



## A Human Robot Interaction Toolkit with Heterogeneous Multilevel Multimodal Mixing

B. (Bob) van de Vijver

MSc Report

**Committee:**

Prof.dr.ir. S. Stramigioli  
Dr.ir. E.C. Dertien  
Dr.ir. D. Reidsma  
Dr.ir. J.F. Broenink  
D.P. Davison, MSc

August 2016

035RAM2016  
Robotics and Mechatronics  
EE-Math-CS  
University of Twente  
P.O. Box 217  
7500 AE Enschede  
The Netherlands



## Summary

In the modern world robots are being used more and more for everyday tasks, but mostly they are being used for industrial purposes. A relatively new application field is autonomous conversation, which requires robots to have social interaction with humans. For social interaction it is important to have fluent and lifelike robot behavior, without the need of explicit constant control. This requires us to no longer see the robot as a puppet, that always needs its puppeteer, but to see it as an actor, which has a director that asks for certain behavior.

The aim of this Master project was to create software that is capable to mix several (external) behavior requests (which for example can come from a director) into viable robot movements. The resulting behavior needs to be fluent and lifelike in order to create good social interaction, but it also needs to be without dead time. The robot should not depend on the external controller and therefore it is also in need of autonomous behavior, which is also capable of overwriting the behavior requests. This way, the robot can for instance be startled by something it sees while he is in conversation with someone.

To achieve this goal, Heterogeneous Multilevel Multimodal Mixing (HMMM) is introduced, which mixes the (external) behavior requests into the required viable behavior. HMMM also processes the required autonomous behavior, combining/overwriting it with the external requests when necessary.

The external requests during this project come from AsapRealizer, which is a behavior realizer from the Human Media Interaction (HMI) research chair. A behavior realizer plans the behavior during the interaction: it decides on what to tell, what movements should be made and most importantly when they should be made. AsapRealizer acts as the director for the robot hardware.

A robot and a human robot interaction toolkit from within the Robotics and Mechatronics (RaM) research chair are being used as starting point. The toolkit already allows a puppeteer to fully control the robot, but it has no autonomous behavior with the exception of breathing movements and eye blinks. HMMM is added to the toolkit and both the robot and toolkit are improved. Motion detection is added to be a source for the autonomous behavior.

The result is a re-usable human robot interaction toolkit which has the capability to accept behavior requests from other software, resulting in fluent and lifelike behavior. It does not need constant control and it can autonomously break any requested behavior when necessary. The resulting robot seems to have a positive effect on others and it can be used for social interaction with humans.

A few recommendations are made to further enhance the possibilities of the human robot interaction toolkit. With recommendations ranging from extra features to HMMM improvements, there should be enough content for a subsequent project.

## Samenvatting

In de moderne wereld worden robots steeds meer gebruikt voor alledaagse taken, al worden ze het meest gebruikt voor industriële toepassingen. Een relatief nieuw toepassingsgebied is autonome conversaties, waarbij de robots sociale interactie met mensen moeten aangaan. Voor sociale interactie is het belangrijk om vloeiend en levensecht gedrag te vertonen, zonder dat de robot constant bestuurd wordt. Hiervoor is het noodzakelijk dat de robot niet langer als een pop gezien wordt (die altijd zijn poppenspeler nodig heeft om te functioneren) maar als een acteur waarbij een regisseur vraagt om bepaald gedrag.

Het doel van dit Masterproject is om software te ontwikkelen die verschillende (externe) verzoeken (van bijvoorbeeld een regisseur) voor bepaald gedrag met elkaar mixt om zo correcte gedrag op te leveren. Het resulterende gedrag moet vloeiend en levensecht zijn om goede sociale interactie en het moet nooit zijn dat de robot stil staat. De robot mag dus niet afhankelijk zijn van de externe aansturing waardoor autonoom gedrag ook benodigd is, die ook de huidige verzoeken moet kunnen overschrijven. Op die manier kan de robot bijvoorbeeld schrikken door iets wat hij ziet terwijl hij met een conversatie bezig is.

Om het doel te bereiken wordt Heterogeneous Multilevel Multimodal Mixing (HMMM) geïntroduceerd, wat de (externe) gedragsverzoeken gaat mixen naar het vereiste correcte gedrag. HMMM behandelt ook het benodigde autonome gedrag door het te combineren met de externe verzoeken indien nodig. De externe verzoeken komen tijdens dit project van AsapRealizer, een gedragsplanner van de Human Media Interaction (HMI) vakgroep. Een gedragsplanner plant het gedrag gedurende de interactie: het besluit wat er wanneer gedaan moet worden. Feitelijk is AsapRealizer de regisseur van de robot.

Een al bestaande robot en mens-robot-interactie toolkit van de Robotics and Mechatronics (RaM) vakgroep wordt gebruikt als basis. The toolkit maakt het al mogelijk om de robot als pop te bedienen, maar het mist nog het benodigde autonome gedrag (met uitzondering van adembewegingen en oogknipperen). HMMM wordt geïntegreerd in de toolkit en zowel de robot als de toolkit worden verbeterd. Bewegingsdetectie wordt ook toegevoegd om als bron voor het autonome gedrag te dienen.

Het resultaat is een mens-robot-interactie toolkit die de mogelijkheid heeft om gedragsverzoeken van andere software te verwerken, met als resultaat vloeiend en levensecht gedrag. Het werkt zonder constante bediening en het kan autonoom het gevraagde gedrag onderbreken als dat nodig is. De uiteindelijke robot lijkt een positief effect op anderen te hebben en hij kan uitstekend gebruikt worden voor sociale interactie.

Aan het eind worden er nog een paar aanbevelingen gedaan om de functionaliteit van de toolkit te verbeteren. Met aanbevelingen die gemaakt worden zou er genoeg materiaal moeten zijn om een vervolgproject te kunnen vullen, waarbij gekeken kan worden naar verbeteringen in HMMM tot het toevoegen van nieuwe features.

# Contents

<b>1</b>	<b>Introduction</b>	<b>1</b>
1.1	Context . . . . .	1
1.2	Setup . . . . .	1
1.3	Main goal . . . . .	2
1.4	Approach . . . . .	3
1.5	Report outline . . . . .	4
<b>2</b>	<b>Analysis</b>	<b>5</b>
2.1	Using robots for social interaction . . . . .	6
2.1.1	Control on multiple abstraction levels simultaneously . . . . .	6
2.1.2	Literature on social interaction . . . . .	7
2.1.3	Behavior classification . . . . .	7
2.2	Related work . . . . .	7
2.2.1	Social robots . . . . .	7
2.2.2	Available toolkit . . . . .	8
2.3	Subsumption architecture . . . . .	9
2.3.1	Sub-behaviors . . . . .	9
2.3.2	Strength and weaknesses . . . . .	10
2.4	End-effector control . . . . .	10
2.5	AsapRealizer . . . . .	10
2.5.1	A complete system . . . . .	11
2.5.2	Communication . . . . .	11
2.5.3	Concluding . . . . .	12
2.6	Saliency . . . . .	12
2.7	Requirements . . . . .	13
2.7.1	Control design and software . . . . .	13
2.7.2	Hardware . . . . .	14
2.8	Concluding . . . . .	14
<b>3</b>	<b>Design</b>	<b>15</b>
3.1	Control design (HMMM) . . . . .	15
3.1.1	Emotion mixing . . . . .	15
3.1.2	Gaze mixing . . . . .	16
3.1.3	Sequence mixing . . . . .	17
3.1.4	Animator . . . . .	20
3.1.5	External communication . . . . .	21

3.1.6	Parameters . . . . .	22
3.2	Software . . . . .	22
3.2.1	Existing functionality . . . . .	22
3.2.2	Saccade movements . . . . .	23
3.2.3	System resource usage . . . . .	23
3.2.4	Saliency detection . . . . .	24
3.2.5	Save shutdown . . . . .	25
3.3	Hardware . . . . .	25
3.3.1	Processing unit . . . . .	25
3.3.2	Camera location . . . . .	26
3.3.3	Display driver . . . . .	26
3.3.4	Power supply . . . . .	26
3.3.5	Casing . . . . .	26
3.3.6	Portability . . . . .	27
3.3.7	MIDI controller . . . . .	28
3.4	Final system . . . . .	28
<b>4</b>	<b>Results</b>	<b>29</b>
4.1	Control design (HMMM) . . . . .	29
4.1.1	Gaze mixing . . . . .	29
4.1.2	Validation of HMMM . . . . .	29
4.2	Software . . . . .	31
4.2.1	Performance . . . . .	31
4.2.2	Saliency detection . . . . .	32
4.3	Hardware . . . . .	32
4.3.1	Portability . . . . .	33
4.3.2	Cameras . . . . .	33
4.3.3	Evaluating the hardware changes . . . . .	34
4.4	Final system . . . . .	34
<b>5</b>	<b>Conclusion and recommendations</b>	<b>35</b>
5.1	Conclusion . . . . .	35
5.1.1	Control design and software . . . . .	35
5.1.2	Hardware . . . . .	35
5.1.3	Research questions . . . . .	36
5.1.4	General notes . . . . .	36
5.2	Recommendations . . . . .	37
<b>A</b>	<b>Demonstration instructions</b>	<b>38</b>
A.1	Manual control . . . . .	38

---

A.2	Demonstration mode . . . . .	38
A.3	Autonomous mode . . . . .	38
<b>B</b>	<b>EASEL architecture</b>	<b>39</b>
<b>C</b>	<b>External communication</b>	<b>40</b>
C.1	Communication basics . . . . .	40
C.2	Saliency . . . . .	40
C.3	Emotion . . . . .	41
C.4	Sequence . . . . .	42
C.5	Feedback . . . . .	43
C.5.1	Gaze feedback . . . . .	43
C.5.2	Prediction feedback . . . . .	44
C.5.3	Planning and progress (general) feedback . . . . .	44
<b>D</b>	<b>Bill of materials</b>	<b>45</b>
	<b>Bibliography</b>	<b>46</b>





# 1 Introduction

## 1.1 Context

In the modern world robots are being used more and more to support humans with legions of different tasks, such as production but also dangerous maintenance. Next to the already common tasks, a relatively new application field where robots are being deployed is autonomous conversations with humans. For example, chatbots are used to answer simple questions from webshop visitors without the need of human interaction. However, such chatbots can usually only communicate with text, making it hard to create a positive social interaction.

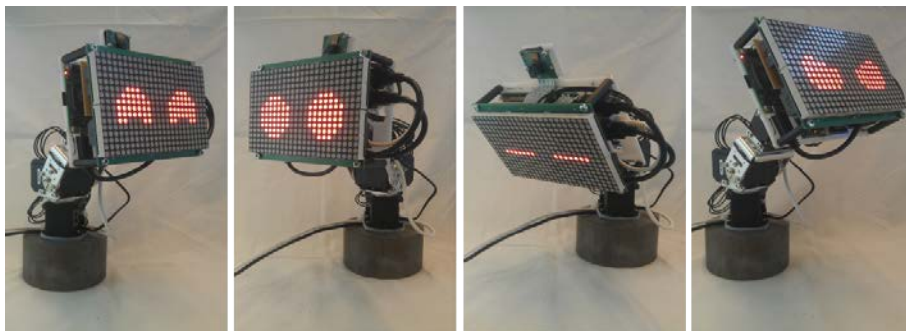
More interesting are physical robots that can directly interact with their environment and are able to communicate with anyone that needs attention. An example of such an application could be a receptionist, that can help visitors with finding the correct office or that can make appointments whenever the requested person is not available. For such a robot it is important that the interaction is fluent and lifelike: this improves the overall (conversational) experience. It should also not be focusing only on the given tasks: when something more important happens, which is not related to the actual conversation, it will need to acknowledge it.

To create a fluent and lifelike interaction it is important to have a robot that has some form of autonomous behavior. Most robots are controlled as a puppet; whenever their puppeteer is not present, they are unable to do anything and appear to be lifeless. If a robot is no longer seen as a puppet that needs constant control, but as an actor instead (where a director takes the place of the puppeteer), the robot will always be able to do something. It will try to act to the commands from the director, but it will interpret them and combine it with its own behavior.

This research focuses on software that can be used to create a fluent and lifelike social robot, that will function autonomously. Many work has already been done to create social robots, but there is currently no easy and reusable software that can be applied for the described goal. Our research chair already has an reusable software toolkit (Oosterkamp, 2015), but it does not contain the functionality to create the social interaction as described. Even though it is currently only used to puppeteer a robot, it can be extended to make the robot behave like an actor, where the director will be positioned outside of the robot.

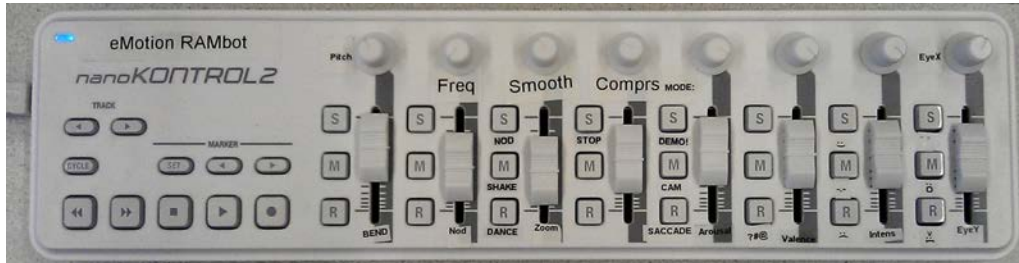
The robot director is called a behavior realizer, which will analyze the scene and request actions accordingly. It decides on the social required interaction steps (the conversational partner and the dialog) and will instruct the actor as such. The actor will try the fulfill the requests, but when there is something in need of attention, it will take the focus instead.

## 1.2 Setup



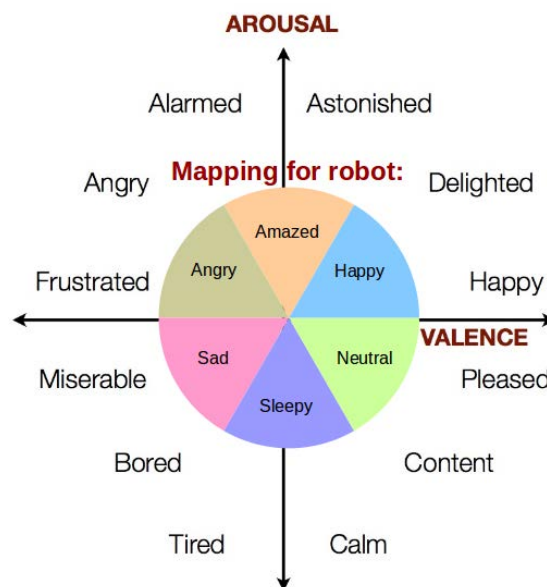
**Figure 1.1:** The small, iconic small robot from earlier work (Watanabe et al., 2013)

An earlier project exploring the notion of ‘anti-social’ robot behavior (shown at the HRI 2013 (Watanabe et al., 2013)) yielded a minimalistic, abstract and mostly iconic small robot, of which a picture is printed in Figure 1.1. Although it is basically not much more than a neck with two eyes, the design proved capable of successfully conveying multiple emotions onto human subjects. It has been the basis for a subsequent MSc project (Oosterkamp, 2015) using the design as a basis for a toolkit for human robot interaction. The toolkit provides a direct control interface for a puppeteer (see Section 2.2.2 for a more in depth description of the toolkit.)



**Figure 1.2:** The MIDI-panel used to control the robot (Oosterkamp, 2015)

The basic controls of the robot are based on commands sent with a MIDI panel such as pictured in Figure 1.2. There are multiple presets and modes available, among which triggering prebuilt animatronic sequences, direct joint control, animations of the LED screen that shows the expressive eyes, and emotion control via 2D arousal and valence space (Figure 1.3). There is some autonomous behavior present, namely a constant breathing motion and eye blinking, being executed without command of the puppeteer.



**Figure 1.3:** Arousal and valence to robot emotion mapping

### 1.3 Main goal

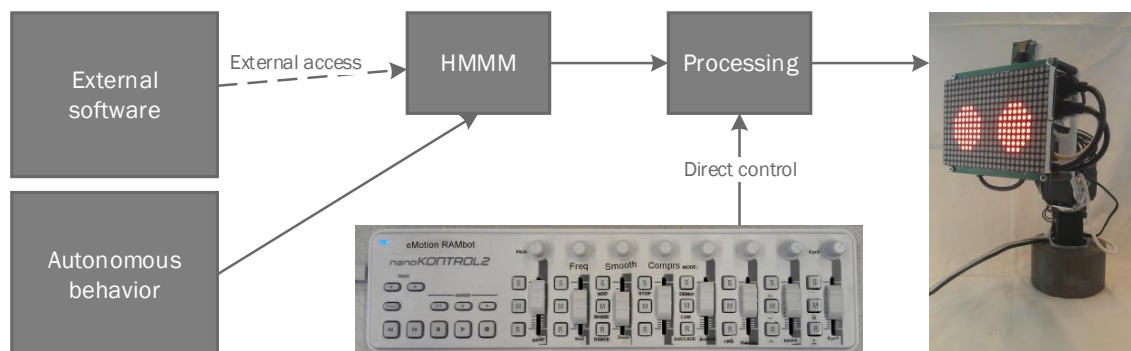
The main goal of this project is to transform the role that the robot fulfills in social interaction from puppet to actor. To achieve this, autonomous behavior needs to be mixed with requests from other software (that is acting as the director), allowing fluent and lifelike behavior without any dead time (time in which the robot does not move at all). Next to this the software should be capable of identifying important events such that it can break the requested behavior to focus on that. The mixing of the control requests and the autonomous behavior needs to be done

for several control types (multimodal) and on multiple levels (multilevel), resulting in Heterogeneous Multilevel Multimodal Mixing (HMMM). Models and algorithms need to be developed together with mechanisms to determine the desirable level of integration for the different types of control, which will all be added to the existing toolkit for human robot interaction.

The existing toolkit already adds breathing motions and blinking actions to the controllable movements and allows these actions to be fluently mixed with the direct joint controls via the manual MIDI-controller. For example, if the user turns the robot via a slider, this is not overwriting the breathing motion, but the motions are combined instead. However, the toolkit does not offer these ‘mixing’ functionality when controlled with high level commands for deliberative behavior.

In order to have multiple inputs available for mixing, it is needed to have a high level communication interface that can be used by external software (the director) in order to send its requests to the robot. In Figure 1.4 the required structure is given. The current toolkit only allows the direct control path; HMMM will add the autonomous behavior and external control through it by mixing them into direct control. Even though the control needs to be self-contained, it will need parameters with which the operation can be tuned.

The external software will be a behavior realizer, that takes the director role in the social interaction. Even though the goal is to design an interface that can be reused, during this project the external software will be *AsapRealizer*. This behavior realizer is a project from the Human Media Interaction (HMI) research chair from the University of Twente, that collaborates with this project.



**Figure 1.4:** A schematic overview of the system goal

The second goal is to update the existing robot hardware in order to improve its portability and design. The current design has a lot of mass positioned in the moving head, requiring a heavy base. As the main processing board is positioned in the moving head, the MIDI control panel needs to be connected there, requiring multiple cables to the moving head. The hardware changes should not negatively impact the look-and-feel of the robot, while the total hardware costs should be similar.

Finally, the possibilities of end-effector control for social robots will be considered. Currently robots are controlled using joint-space, but end-effector control might be beneficial.

## 1.4 Approach

This project focuses on two main aspects: Additional software functionality that implements HMMM and hardware improvements to enhance the portability and looks. For HMMM models and algorithms are developed in collaboration with the HMI research chair, of which the results are implemented and added to the existing human robot interaction toolkit. Concurrently the robot hardware will be updated in order to improve the portability. The existing software will also be updated by identifying problematic structures and improving the general performance.

To order to validate the work done during the project an international workshop, the eNTERFACE'16, will be attended. During a timespan of 4 weeks AsapRealizer will be linked to two different robotic platforms: the robot from this project and the Zeno (Robokind Robots, 2016), which is an humanoid robot. Both robots will receive the same behavior requests from Asap-realizer (tailored for their platform), where the robot with HMMM should move more fluent and more lifelike. It should add different autonomous actions such as eye blinking and breathing movements, but it should also be capable of identifying and focusing important events outside of the conversational scope. When the robot from this project is compared with the Zeno, the Zeno will have dead time (during which the robot does not move at all) as it lacks any autonomous behavior.

### **1.5 Report outline**

This report starts with the analysis of several key-objects related to the topic in Chapter 2. First the research questions will be defined, after which the definition of social interaction will be clarified with the use of abstraction levels en behavior classification. Next, work related to the subject will be mentioned, and some more information about the behavioral realizer (Asap-Realizer) will be given. The subsumption architecture and saliency will be discussed and the chapter will close with a complete overview of the actual requirements based on the analysis.

In Chapter 3, the new design will be discussed, with as subjects control, software and hardware. The complete system will be reviewed in the end of the chapter, which is the actual delivered end-product.

The next chapter (Chapter 4) will focus on the results accomplished with the end-product as described in the chapter before. The control, software and hardware parts will be compared with the old versions, and the communication with external software will be evaluated.

In the final chapter, Chapter 5, the final conclusion will be given and some recommendations for future work will be devised.

## 2 Analysis

As introduced in Section 1.1 this project aims to create a robot which can be used for social interaction, that is fluent and lifelike. In order to achieve this goal HMMM will be created. With HMMM (external) requests from, for example, a behavioral realizer such as AsapRealizer, will be mixed with autonomous behavior. The autonomous behavior is also present when there are no other requests, ensuring that the robot does not have any dead time, but it can also use it to interrupt the requested behavior.

For this project a couple of research questions are formulated which will be used to construct the project requirements. The main research part of the project is related to control design, specifically the heterogeneous multilevel multimodal mixing (HMMM) of all available input signals. The questions are as follows:

1. *What are the direct controls that need to be exposed to a behavior realizer?*  
To be able to control the hardware from a behavior realizer, a certain set of controls must be available.
2. *What are the operation parameters that need to be configurable?*  
With the controls it should also be possible to adjust a certain set of parameters in order to tune the mixing operation.
3. *What is the most suitable communication schedule for the communication between the robot hardware and the behavior realizer?*  
A suitable communication schedule for the behavior realizer and the robot hardware is a must-have.
4. *Will the behavior/executed actions be more fluent when the mixing is done on the robot hardware?*  
When the mixing of the input signals (behavior) is done on the robot hardware, the resulting behavior should be more fluent and more lifelike as there is less need for communication between the different systems.

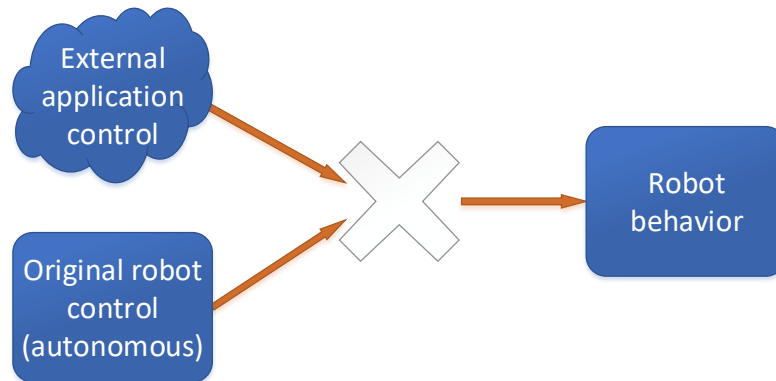
It is expected that with the answers of the first three questions the basis for HMMM communication is known. However, the real main goal of the project is the implementation of HMMM, which yields the following research question:

5. *How can different control inputs be mixed and what are the implications of the mixing on the different control levels?*  
Although the mixing of (external) input signals sounds trivial, it must be decided how the actual mixing behavior will be, which depends on the used controls. When HMMM is used, there might be implications of the current control levels: is everything as available as before, or are there any concessions to be made?

Currently the robot is controlled using joint space control, however there might be other control options that are more suitable. This creates the following extra question:

6. *Is end-effector control more convenient than joint-space control in relation to HMMM?*  
Currently joint-space control is used in the available toolkit. It might be easier to use end-effector control as for that type of control basically only the point to go to needs to be communicated, while the joint positions can easily be calculated. This might be more cumbersome when using joint-space control.

A global overview of the mixing is given in Figure 2.1. The rest of this chapter will analyze related work and existing applications that are used during the definition of the project requirements in Section 2.7.



**Figure 2.1:** The system should mix the external application control with the autonomous controls

## 2.1 Using robots for social interaction

A robot that is designed to be used in social conversational context can use its hardware to realize multiple behavioral levels. These levels, or behaviors, can be classified into four groups:

- *Autonomous*  
Such as idle motions and breathing.
- *Semi-autonomous*  
Such as motions required to keep the gaze focused on a certain target.
- *Reactive*  
Such as reflex responses to visual input.
- *Deliberate*  
Such as speech or head gestures that make up the utterances of the conversation.

Part of the expressions (and especially the deliberate ones) will be triggered by requests from, for example, a dialog manager. A dialog manager for social dialogs typically orchestrates the progress of the social conversation between human and robot, and - based on this progress - requests certain deliberate behaviors to be executed and certain changes to have to be made to parameters of the autonomous behavior of the robot. Such requests are typically specified using a high level behavior script language such as the Behavior Markup Language (BML), which is agnostic of the details of the robot platform and its controls and capabilities for autonomous behaviors. The BML scripts are then communicated to the robot platform by a Behavior Realizer (e.g., AsapRealizer (van Welbergen et al., 2012)), which interprets them in terms of the available controls.

### 2.1.1 Control on multiple abstraction levels simultaneously

The multiple levels that need to be controlled simultaneously are a challenge for the toolkit for human robot interaction and especially for the multiple input mixing extension. While the toolkit can take care of the autonomous and reactive behavior patterns, executed automatically, external inputs for deliberate behavior by a behavior realizer must be accepted and handled as well. The external input might also adjust the parameters for the autonomous/reactive behavior on the fly to adopt them more for the current dialog state.

However, there will be a case that the requested deliberative behavior actually conflicts with the (semi-)autonomous behavior, since the dialog manager does not know the exact current state of the robot and its (semi-)autonomous behaviors. To solve this problem the toolkit for human robot interaction should contain intelligence to prioritize, balance and mix the several, multilevel requests in order to create correct robot control, as the inputs can not be translated directly.

### 2.1.2 Literature on social interaction

Social interaction for embodied conversational agents (virtual or robotic) is a topic in which a lot of research has been done. Therefore a lot of that research can be used during the development of the input prioritization, balancing and finally, the mixing. A research project from within the university (Gennep, 2013) is used as reference guide for gazing behavior. This guide also references an older paper from within the same group (Heylen, 2006), which discusses the principles of conversational structure. There is also an architecture named EASEL (Expressive Agents for Symbiotic Education and Learning) (University of Twente, 2015) available which describes a symbiotic education and learning architecture: it contains all the needed components for social interaction, although based in education.

### 2.1.3 Behavior classification

For this project the final behavior was split into three levels from the defined behaviors in Section 2.1, namely (semi-)autonomous, deliberate and reactive. The two autonomous behavioral levels are combined to a single level, as they will be handled likewise.

#### (Semi-)autonomous

The autonomous domain must be implemented within the robot itself, such that the external control can focus on the deliberate behavior. This also means that there will be no direct controls available for the external controller: only the parameters of the (semi-)autonomous domain can be adjusted on request, but the adjustments will be processed directly.

#### Deliberate

The deliberate domain is reserved for the external controller. This can be a dialog system with AsapRealizer, but the MIDI-controller will also be considered as an external control. However, this deliberate behavior must be mixed with the other behaviors: that is exactly the goal and exactly what the HMMM must do.

#### Reactive

The reactive domain can be considered a special case: even though the given example in Section 2.1 describes a reflex action, the actual control can be implemented on the robot hardware or the external controller, depending on multiple factors such as the available hardware and processing. However, the exact location of the control does not matter, as it still needs to be mixed with the autonomous behavior. It does matter for the type of communication: when it is implemented externally it will be processed as direct control. When implemented on the robot, it is only possible to change the parameters.

## 2.2 Related work

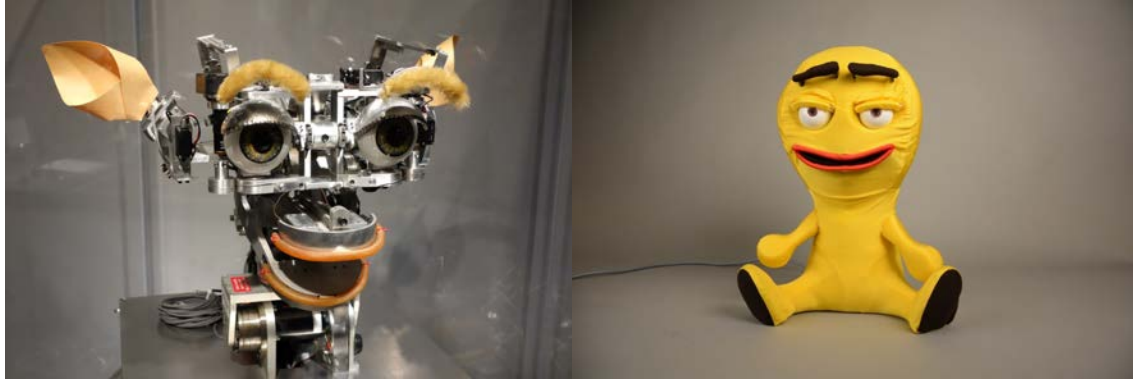
### 2.2.1 Social robots

A book published by MIT (Breazeal, 2002) already presents a vision of the social robots of the future, which mainly describes that robots will socially intelligent in a human like way. It also gives details about a specific robot named Kismet (pictured in Figure 2.2a), which incorporates



insights from the scientific study of animals and people. However note that Kismet had already been build in the late 1990s and can be considered one of the first concepts of a social robot.

Another publication in the field of social robot is about the robot Ono (pictured in Figure 2.2b), which is low-cost open source social robot (Vandeveldel, 2014). The robot hardware is again simple and should be easy to do-it-yourself, but it's face is more complicated as it has a total of 8 degrees of freedom (DOF) for only the facial expression. It also does not have the possibility to move it's head, which makes it rather static.



(a) Kismet (Breazeal and Scassellati, 1999)

(b) Ono (Vandeveldel, 2014)

**Figure 2.2:** Two examples of other social robots

The difference between the two robots is easy noticeable. Where Kismet is clearly build to demonstrate the functionality of movement of in eyes, eyebrows, ears and mouth in relation to social interaction, the Ono (build later) is more meant to be directly socially attractable, which is mostly the result of the resemblance with a stuffed animal, something the Kismet is missing.

However, the movement the Ono can make is limited to its mouth, upper eyelids and eyebrows. It can create facial expressions but there is no body movement to support the emotion. The Kismet however can move it's head and even it's eye's while it has also the ability to actuate the lips, ears and eyebrows. Therefor should the Kismet be able to create more convincing emotions.

The hardware used in this project has combined a simplistic design while retaining as much as possible features. Although it is missing a mouth and ears that can be used to amplify emotions, the flexible display and multiple degrees of freedom in the neck joint create enough possibilities to convey emotions successfully.

### 2.2.2 Available toolkit

The current, available toolkit that has been developed, consist primarily out of a number of software components for input (midi, camera, Behavioral Markup Language), output (servo motors, serial communication), playback and control of animation sequences, a 3D visualizer containing the kinematic model of the robot, etc. These components have been implemented in Robot Operating System (ROS) and are lean enough to be run on a Raspberry Pi 2. The screen is controlled by its own (small Arduino) microcontroller which renders built-in animations based on parameters from the ROS controller. The motors that are used are RS485 bus controlled Dynamixel servos. The primary goal of this toolkit is to provide a robot developer with a well-chosen set of building blocks (software and hardware) to equip (existing) robot platforms with capabilities for social interaction.



## 2.3 Subsumption architecture

The subsumption architecture is a layered control structure (proposed by Brooks (1986)) where the higher level layers subsume the roles of the lower level when they wish to take control. It was proposed in opposition to traditional symbolic artificial intelligence: instead of guiding behavior by symbolic mental representations of the world, it combines sensory information to action selection in a bottom-up manner. Every layer is unaware of the layers that resides above it even though those can interfere with its data path. The basic structure can be observed in Figure 2.3.

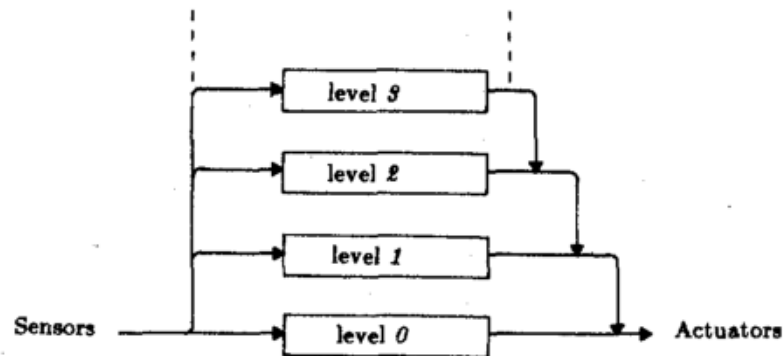


Figure 2.3: The subsumption architecture as defined by Brooks (1986)

### 2.3.1 Sub-behaviors

To make the architecture work, the complete behavior is divided into sub-behaviors which form the aforementioned layers. Each sub-behavior has his own level of behavioral competence and higher sub-behaviors subsume the lower sub-behaviors in order to create viable behavior. As a simple example an exploring robot can be used; three sub-behaviors can be composed. The lowest behavior could be 'avoid objects', the second layer 'wander around' and the last layer 'explore world', as printed in Figure 2.4. To wander around effectively, the robot should avoid objects, and with the subsumption architecture the higher level utilizes the lower level competences. All layers receive sensory information, execute parallel and generate separate output which can be directed to actuators or other layers to suppress or inhibit them.

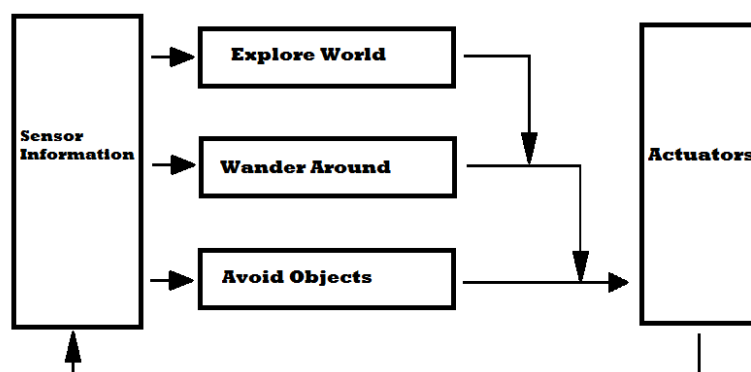


Figure 2.4: An simple example of the subsumption architecture with the different layers filled with sub-behaviors

### 2.3.2 Strength and weaknesses

Although the subsumption architecture has several strengths (such as parallelism, incremental design and generality), it also has a few significant drawbacks. To design the architecture (such as decomposing the layers and choosing the suppress and inhibit signal) takes a lot of work and expertise in the subject. Furthermore, when the decomposition is done, the result is inflexible during runtime; it is usually specifically targeted for a single goal (Granot, 2008).

Although the subsumption architecture is proposed to mix several sub-behaviors in order to create viable behavior, it is not sufficient to be used for the goal of this project. As it is desired to mix several different (external) control signals into embodiment actions, there will be the need of a more complex system which can be configured dynamically. More important, as external control signals will be supported, it is not trivial to classify them into specific layers inside the subsumption architecture. Finally, the (external) control signals from a single source might be impossible to be placed in a separate sub-behavior, as they might be classified as multiple different layers of behavior.

## 2.4 End-effector control

As stated earlier, the toolkit currently uses joint-space control for every joint. This means that every joint position is calculated manually: every actuator angle needs to be known in order to steer the robot. For example, if the head needs to move to the left, it must be decided which joint is the most convenient to use and then its angle must be updated. It might be more convenient to use end-effector control (as formulated in research question 6).

When end-effector control is being used, the state of the individual joints is no longer that important: only the orientation of the end-effector in the space is important. By calculating the transform matrices the joint angles will follow from the requested end-effector orientation: it might even occur that there are multiple possibilities to reach the requested orientation. In that case one of the solutions needs to be chosen. Most commonly this will be the option that keeps the required total angle change (which usually is also the least amount of work) to a minimum. This simplifies the control at runtime as you only need to specify the orientation, but you will need to calculate the transform matrices beforehand (which is something that needs to be done for every different actuator configuration).

Even though end-effector control can be convenient for general robot control, it unfortunately is not as convenient for social robots as might be expected. Due to the nature of social interaction, not only the position of the end-effector (in this case the display with the eyes) is relevant. It is important that you can control individual joints to create, for example, a convincing nodding motion by using a specific single joint. When solved with end-effector control it is unknown which joint is, or even joints are, used.

In conclusion, social robots do not get particular benefits from using end-effector control over joint-space control. It can even be considered inconvenient to use end-effector control, as it is no longer possible to create specific joint movements needed for certain social interactions. This also means that the answer on research question 6 is already known. End-effector control is not more convenient than joint-space control in relation to HMMM and social interaction.

## 2.5 AsapRealizer

The original toolkit already has the possibility to connect with AsapRealizer (Oosterkamp, 2015), but it does not support many different commands. There are only three available commands: dance, shake and stop. Nevertheless, the link layer can be reused for the communication schedule that will be proposed, as it is a standard component. This chapter will discuss the link layer and the internals of a system using AsapRealizer.

### 2.5.1 A complete system

When the goal is to build a complete dialog system, only AsapRealizer and an embodiment (robot or virtual) will not be sufficient. Since AsapRealizer only interprets BML in terms of the available controls of the embodiment, there is also a need for a producer of those scripts. This producer is in the need of an intent and behavior planner, which plans the actions and communicates with the realizer using the afore mentioned BML scripts. Figure B.1 in Appendix B contains an global overview example of a complete system which contains all necessary parts to form a complete dialog system.

The system can be divided into four core groups:

- *Perception*  
The perception group is used to perceive the external world. This can have multiple forms, like speech or scene recognition. The data is communicated to the next layer.
- *Reasoning, memory and interaction management*  
Based on the information from the perception and the current dialog state (conversational) actions are planned, which are expressed in BML. The BML scripts are send to the next layer.
- *Behavior generation*  
The behavior generation (in this case AsapRealizer) interprets the BML scripts from the interaction management and converts them into the control primitives available at the embodiment.
- *Presentation and motion control*  
The presentation and motion control uses the motion primitives generated by the behavior generation and processed them into the action action of the embodiment.

Taking the complete system into account, it is clear that the focus of this project is on the communication between the last two groups and the conversion of the primitive controls to actual actions of the embodiment.

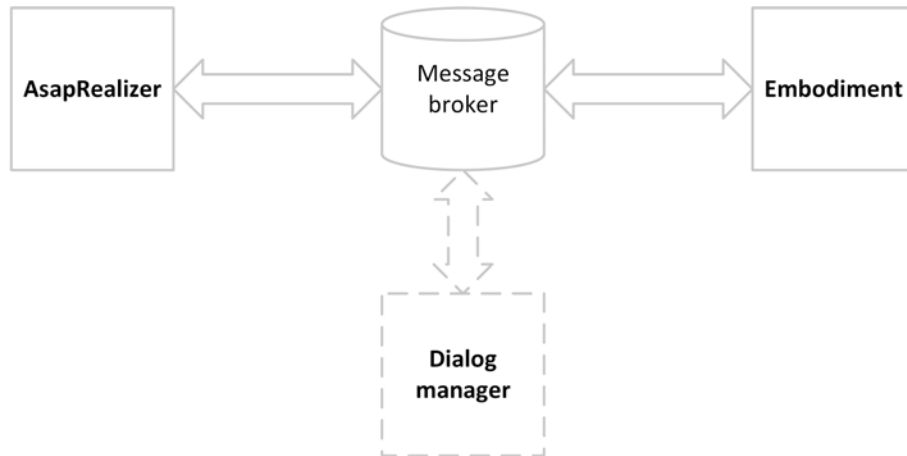
### 2.5.2 Communication

The EASEL architecture is build around modularity, such that all parts of the system can be interchanged. Every part can also be run separately and can thus distributed over multiple machines. To be able to realize such distribution, a central message broker (in this case Apache Apollo) is required which fulfills the role the coupling server for every separate part. The same holds for AsapRealizer and the embodiment (in this case ROS): the messages between the two parts all go though the central server. Due to this communication structure the separate parts do not need to know the complete system topology but only their own input and output channel; this simplifies the node configuration. A graphical representation of the communication is given in Figure 2.5.

The behavior realizer and the embodiment communicate through a message broker using a body containing a XML-message. The used XML-template is printed in Codeblock 2.1. As stated earlier, currently only three actions are supported which can be substituted in the `${animationName}` variable. Combined with message body are a few headers, indicating the destination (the ROS channel) and message- and subscription-id (used internally by the broker).

```
1 <data>
2 <sequence type="str">${animationName}</sequence>
3 </data>
```

**Codeblock 2.1:** A sample XML-message template used for communication



**Figure 2.5:** A possible communication network when using AsapRealizer: AsapRealizer and the embodiment are required for this project, but it can be expanded with any component needed, such as a dialog manager.

### 2.5.3 Concluding

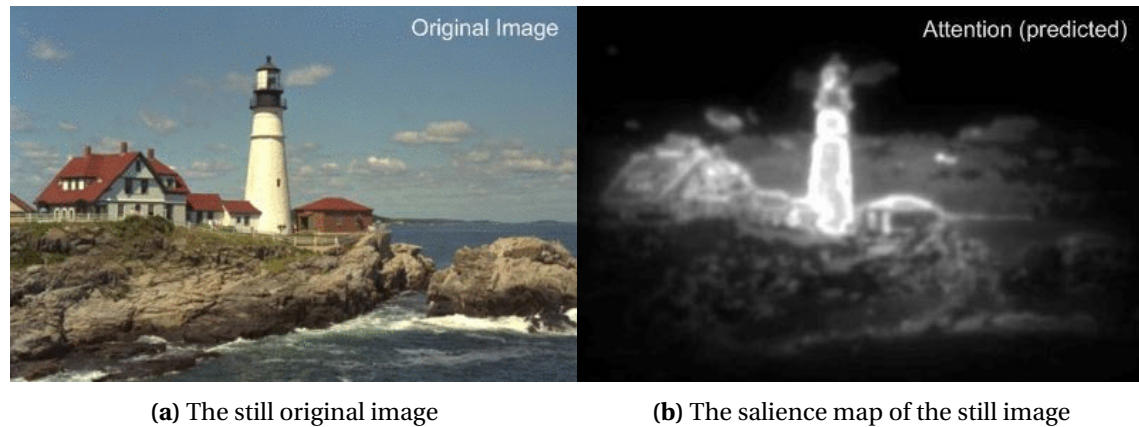
To construct a complete dialog system more than only the embodiment and AsapRealizer is needed. There is also the need for an interaction manager and some sort of perception. The message templates help to make the communication between the embodiment and AsapRealizer as flexible as possible, such that almost anything can be communicated.

## 2.6 Saliency

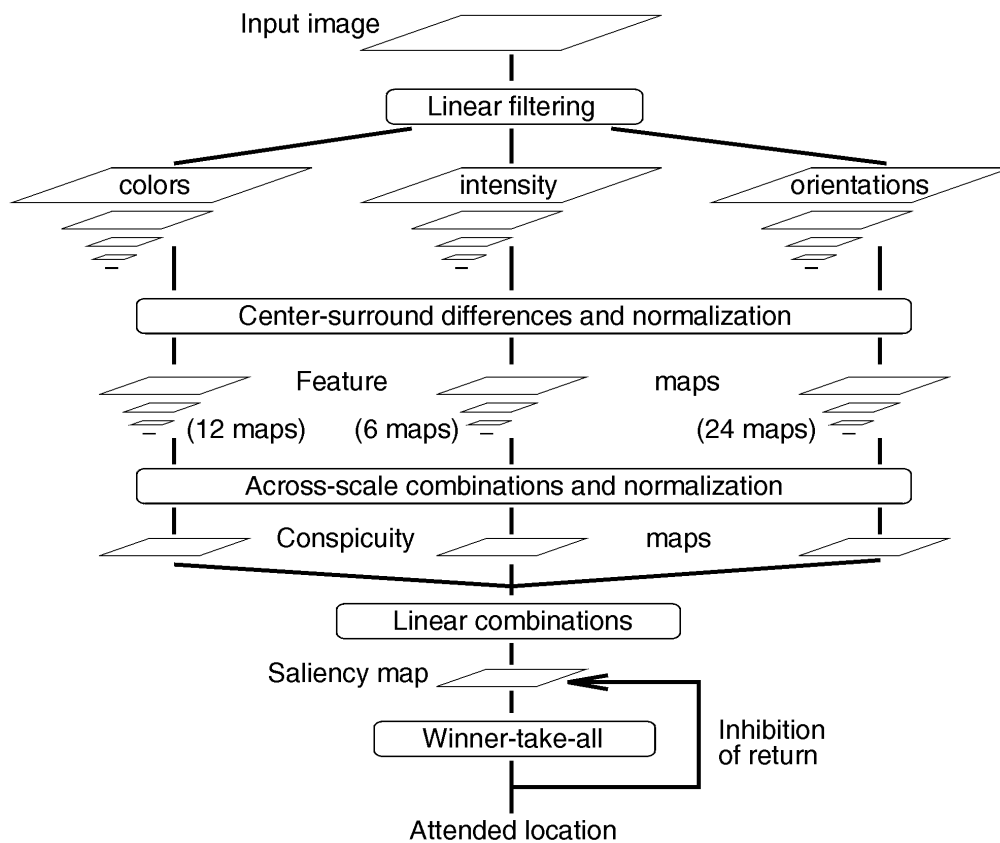
The saliency of an item is the state of quality by which it stands out relative to its neighbors. Items can be anything in the current field of sight, such as objects, persons and more. Saliency typically arises from contrasts between items and their neighborhood: examples can be the flickering lights in a room or loud sudden noises. In robots saliency is usually focused on the visual domain, even though others domains are possible, such as the auditory domain. The saliency of a specific item can be influenced by training: this means that for everyone/everything the saliency map of an environment may differ (Schneider and Shiffrin (1977) & Shiffrin and Schneider (1977)).

Multiple articles are available about saliency mapping of images and videos. The most challenging part lies within the salience detection of an video with a moving camera. From within the research chair there is an article available discussing the saliency-base humanoid gaze emulation using a moving camera setup (Reilink, 2008). It uses an existing saliency map algorithm and adjusts it to work in a moving camera setup. There have been multiple improvements in saliency detection since then, which for example now use manifold learning (Jiang and Crookes, 2012).

An example of a salience map of a still image is given in Figure 2.6. It can be seen that specific parts of the image are more salient than others: they are more interesting than the other parts of the image. In Figure 2.7 an algorithm (Reilink, 2008) for creating the salience map and eventually the location to attend is given. It is important to note that any mixing of saliency mapping must be done before the ‘winner-takes-all’ action as that final step decides the point at which the gaze must be directed. The inhibition of return feedback makes sure that regions which have been attended to become less salient over time. It also initially enlarges the salience of a specific area after the first selection to prevent rapid eye-movement as the most salient place might change quickly over time otherwise.



**Figure 2.6:** Saliency detection of a still image (University of Mons - NumediArt Institute - Attention Group, 2015)



**Figure 2.7:** Architecture of a saliency model (Reilink, 2008)

## 2.7 Requirements

Based on the previously formulated research questions and the analysis on the different subjects, it is possible to formulate the project requirements. Note that some of the requirements are added while they have not been discussed before: however, they are still relevant. The requirements are given in order of importance (per group) and are as follows:

### 2.7.1 Control design and software

1. External software must be able to control the hardware with exposed high level controls

2. Autonomous behavior needs to be available from within the robot hardware
3. The autonomous behavior needs to be combined with external controls
4. The resulting behavior needs to be fluent and natural (lifelike)
5. The available camera needs to be used for autonomous behavior
6. Operation parameters for the autonomous behavior and mixing need to be adjustable on runtime when feasible
7. The external control connection should not be tailored to a specific application
8. The software needs to be reusable in other projects

### **2.7.2 Hardware**

9. The portability of the robot needs to be improved
10. Changes made to the hardware may only have a minimal negative impact on the look-and-feel of the robot
11. The total cost of the robot should remain about the same.

## **2.8 Concluding**

This chapter started with the research questions that are relevant for this project, after which some work on related fields is discussed. There is currently no easy reusable (software) toolkit for social interaction between humans and robots that meets the requirements set. However, the toolkit developed from within the research chair already has a good starting point and it will be extended with the proposed HMMM. With the basic knowledge of social interaction and behavior classification, it can also be concluded that the external controls need to be high level abstractions of the robot. The robot hardware itself should handle the low level controls and the autonomous behavior. This resulted in the requirements 1, 2, 3 and 6.

Note that the main goal is to create fluent and lifelike behavior, which is important during social interaction. It is given as an extra requirement here in item 4.

With the end-effector control concluded as infeasible for social robots due to their animation nature, research question 6 does not lead to any requirements.

With saliency as the most convenient method of controlling the gaze direction due to its mixing capabilities, the onboard camera needs to be used for the autonomous behavior (corresponding to requirement 5). When combined with an external saliency input, it should be able to create fluent, lifelike behavior.

As the software is still aimed to be an easy reusable toolkit for human-robot interaction, the software should be easily reusable by every application (hence the requirements 7 and 8). While the connection with AsapRealizer will be the primary test case, it is convenient that it uses a message broker which can be easily be reused.

For the hardware it is important that the original look-and-feel will be preserved, while the overall portability needs to be improved, resulting in the requirements 9 and 10. The final requirement (11) is set to limit the available budget.

## 3 Design

This chapter will focus on the design of HMMM, the update of the existing toolkit and the improvements of the hardware. The old version of the robot had no specific name and was mostly referenced as the (anti-)social robot of the RaM research chair. During this project the robot has gotten a new name: the EyePi. This name will be used in the remaining of this report to identify the new version of the robot hardware.

### 3.1 Control design (HMMM)

The robot platform used is relatively simple: it has three degrees of freedom with the neck and a 24x16 led-matrix display forming the eyes. However, the toolkit and more specifically HMMM, needs to be able to abstract that into simple control primitives that can also be applied on other robots.

Therefore it is chosen to not give direct control access to the implementation specific hardware, but to work with globally defined requests, simplifying the external interface. For the interface three specific types are distinguished:

1. *Gaze direction*

Instead of controlling the joints relevant to the gaze direction directly, only the gaze direction can be controlled. The request will not contain an simple setpoint: the request will contain a saliency map which will be used together with the other available map to determine the actual gaze direction.

2. *Emotion*

The displayed emotion on the led-matrix display will be controlled with a abstracted emotion mapping, based on valence and arousal (see Figure 1.3 in Chapter 1).

3. *Predefined sequences*

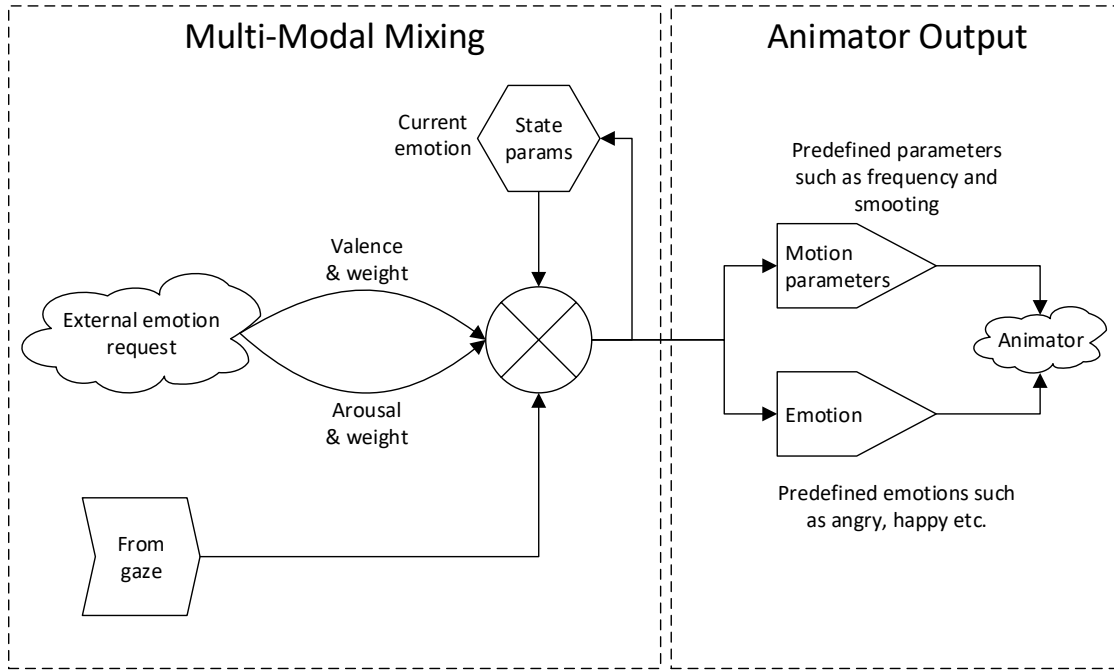
Robot usually have predefined sequences that can easily be reused, such as nodding, shaking and dancing. It is convenient to have predefined sequences available, as they are created specifically for the platform in use. Sequence request will support timing.

It is the task of the HMMM-part to integrate multiple external requests into smooth behavior, combining it with autonomous behavior. Motion(speeds) can depend on the internal parameters, which can depend the current emotion state. Detailed schematic overviews of the proposed HMMM-mixing are given in figures 3.1 (gaze), 3.3 (sequence) and 3.4 (emotion). All figures start with the (external) requests on the left, processing them to the output on the right.

#### 3.1.1 Emotion mixing

The emotion mixing part of HMMM (Figure 3.1) can be considered the simplest mixing part. It processes input from both external requests and requests from the gaze part (see Section 3.1.2). The requests are directly mixed and new output values are calculated based on the previous state and the requested values.

Requests that describe large sudden changes in emotion will be processed instantly: for other requests the emotion state will gradually change into the requested state. Two outputs for the animator are generated: motion parameters and the current emotion. The motion parameters are generated in the emotion mixing part as they are directly related to the current emotion: For example, a cheerful person has sharper and faster motions than a sleepy person has. Due to time constraints the connection with the motion parameters has not been implemented.



**Figure 3.1:** HMMM emotion mixing

### 3.1.2 Gaze mixing

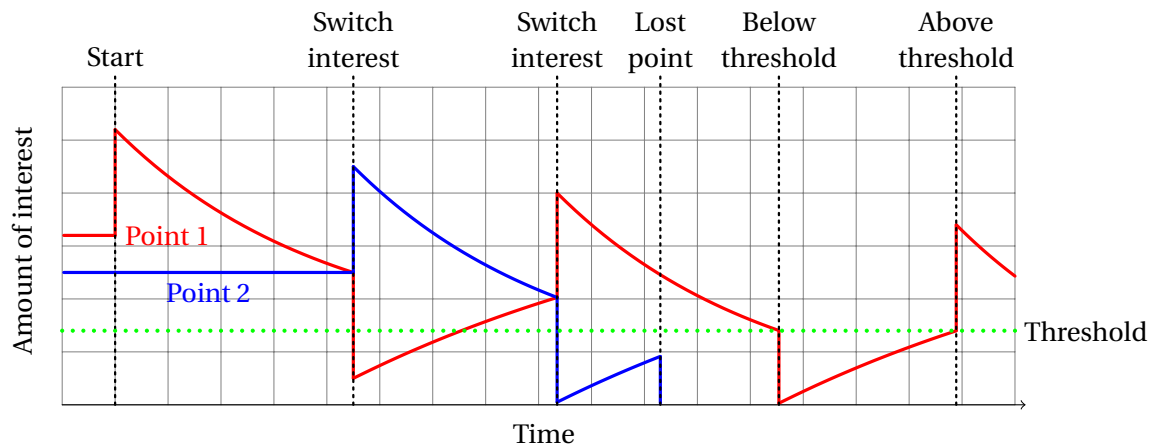
For the gaze mixing part of HMMM (Figure 3.3) two mixing types are used: single-modal and multi-modal. The single-modal mixing purely processes the incoming saliency maps. Although all maps have the same data structure, they can be defined as autonomous or deliberate maps, but that data is not used during the single-modal mixing. All maps are combined into a single map (keeping their original data intact) and fed into the mixer. The mixer chooses the most salient point from the map (which is the point with the highest value), but it uses the current state to update the map first.

Before the selection of the most salient point is done, the mixer computes the difference with the previous map to find related points. Due to small camera movements, but also movements of the object self, the centers of the salient points moves from frame to frame. In order to not see every changed center as a new independent salient point, small changes are detected and the data from the previous point is combined with the updated data on the new point. This method prevents the ghosting of the different points when there is a moving body in front of the camera and it also functions as a smoothing function for the final EyePi movement.

To create a more lifelike behavior, the mixer will loose interest in active salient spots by applying a logarithmic penalty that is configurable during runtime. Whenever a spot looses the interest because another spot is more interesting, the first spots receives an instant penalty for loosing interest to prevent fast interest flipping between two spots. After loosing interest and receiving the last penalty, another logarithmic process will reward the spot to make it more interesting again. The rate is again reconfigurable and the total amount of interest will never be higher than the original calculated amount. The last instant reward in the system is given when a spot receives interest, again to prevent fast switching between interests. Finally, there is a threshold value that needs to be met to be able to retrieve interest. The described behavior is sketched in Figure 3.2: the exact behavior is configurable with parameters.

After selecting the most salient point, the state is updated and the point will be send as feed-back. Although the gaze direction is expected to be directly connected to the selected point, it will first be mixed again in the multi-modal part. In this part, the autonomous generated map





**Figure 3.2:** An sketch of the interest for two interest points during time

(from the internal camera) can induce shock behavior, which is communicated to the emotion and sequence mixing parts that should create the shock emotion and sequences.

This second state mixing can also block a gaze direction from being forwarded to the animator in case specific sequences are active. The mixer is therefore connected to the sequence database and it also receives the output from the sequence mixing part (see Section 3.1.3). When such a blocking sequence is active, the gaze direction output will be frozen until the sequence is deactivated.

### 3.1.3 Sequence mixing

The final mixing part of HMMM, sequence mixing (Figure 3.4), handles both external requests and request from the gaze part. Sequences are pre-defined motions, which have specific motion definitions and requirements for every available actuator. Every robot platform will need it's own definitions for all sequences in order to complete the mixing step. The definitions are specified on actuator level and they have one of the following classifications:

- *Required absolute motion*  
This absolute motion it required to complete the sequence. If it is not possible to do this, the sequence request must be rejected. It is impossible to mix this actuation with any other other that controls the required actuator.
- *Not required absolute motion*  
This motion is still an absolute motion, but on conflicts it can be dropped.
- *Relative motion*  
As this motion is relative, it can be added to almost every other moved by adding it's value. When an actuator is near it's limit, the actuation can be declined.
- *Don't care*  
The actuator is not used, so the sequence does not care about it.

Every sequence request has its own identifier, which is used in the feedback message in order to identify the feedback for the external software.

The first possible rejection is done based on these classifications: the current queue will be checked and the information from the requested sequence is retrieved from the database. If there are any conflicts in actuator usage that can not be solved, the request will be rejected. The second possible rejection is based on the timing of the requested sequence: if the timing can not be met, the request will be rejected. Both rejections will be send back to the requester

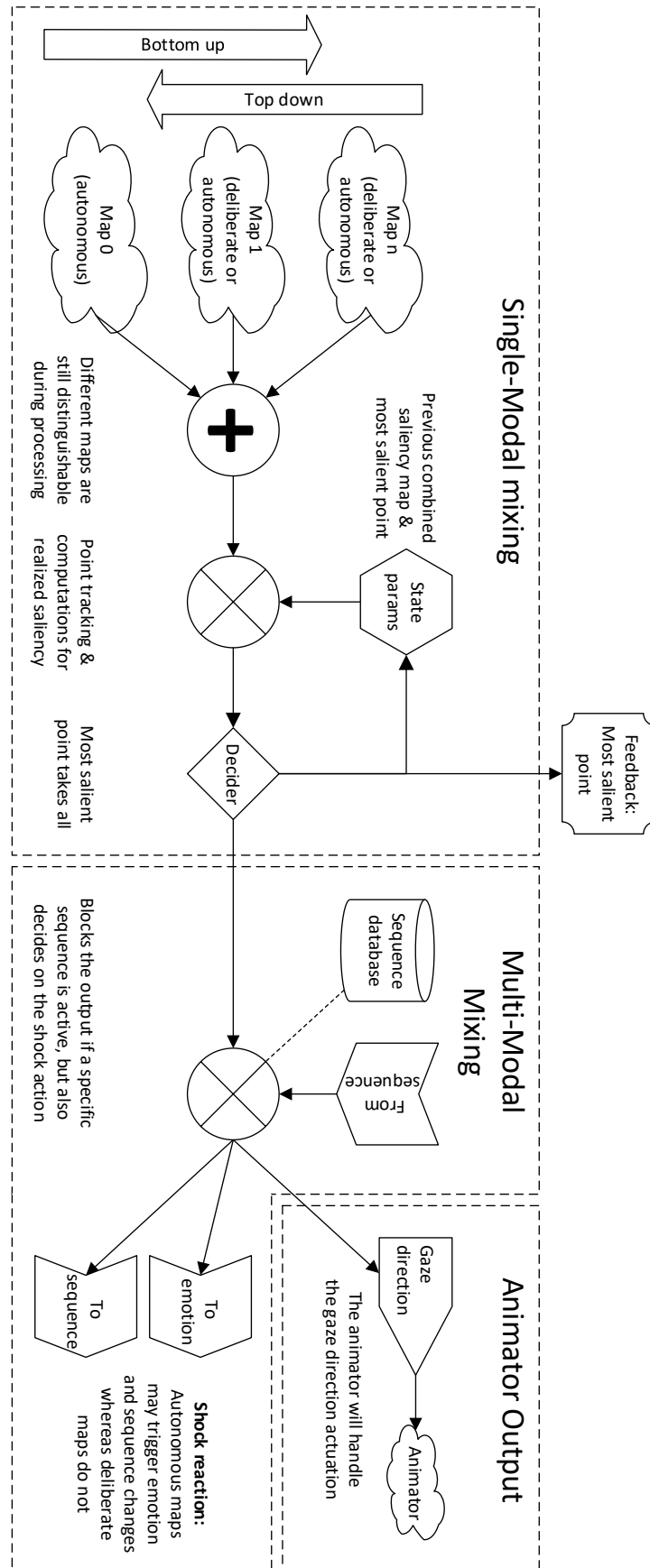


Figure 3.3: HMMM gaze mixing

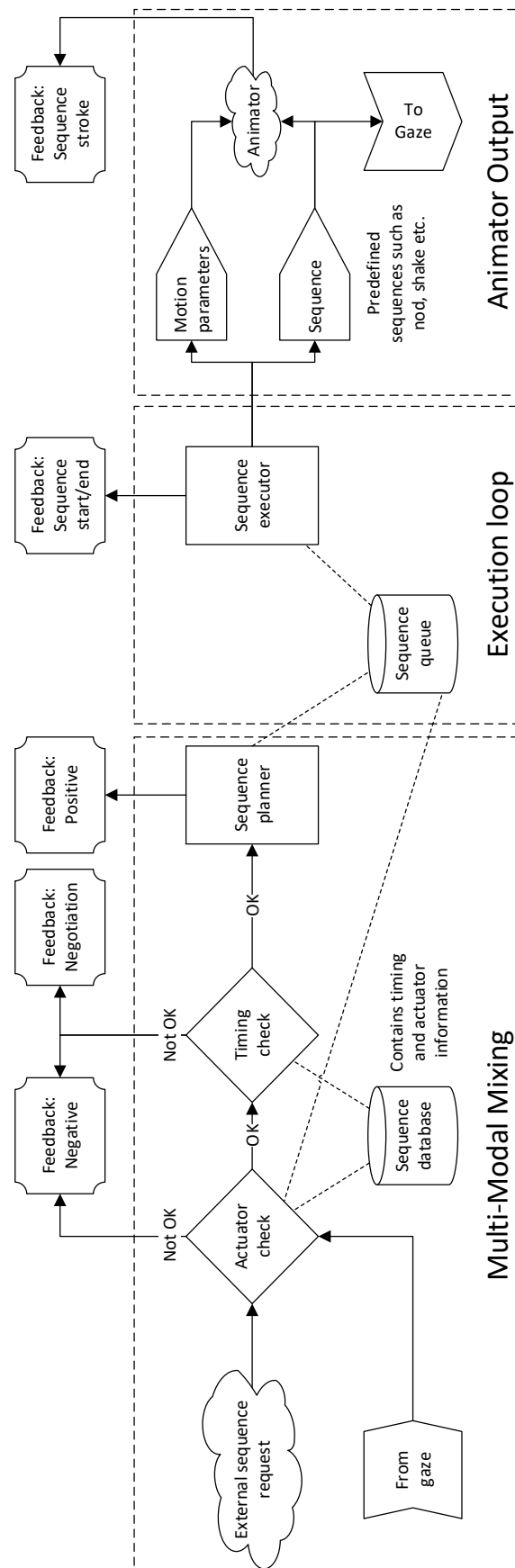


Figure 3.4: HMMM sequence mixing

using a feedback message. More information about the feedback messages can be found in Section 3.1.5.

If the sequence has passed both the actuator and timing check, the sequence planner will put it in the queue for execution. An acknowledgment request is send back to the requester and the processing of this specific request stops for the moment.

The second part of the sequence mixing is no longer direct part of the mixing process itself. There is a constant running process which will activate sequences when there are allowed to start. When a sequence is started, feedback is send to the original requester that the sequence is started. The output to the animator contains both the sequence and possibly adjusted parameters in order to make the timing. The sequence is also transported to the gaze mixing part, in order to operate the blocking behavior there. The animator itself also sends feedback requests on animation strokes. Once a sequence is stopped, the sequence executor also provides that action as feedback.

Due to time constraints it is not possible to implement the complete described gaze mixing. The sequence database and the actuator check have not yet been implemented, but even without those the mixing is still functional.

### 3.1.4 Animator

All mixing parts have one output in common: an output to an animator part. The animator is implementation specific and will differ per robot, but it needs to take the generated output from HMMM as input. These are the same as the input of the HMMM-part, although mixed and they should not clash. An extra data channel is added: motion parameters to adjust speeds of the movements. Note that the animator also has an feedback output, required for progress feedback (see Section 3.1.5). Figure 3.5 provides an schematic overview of the animator as implemented for the EyePi.

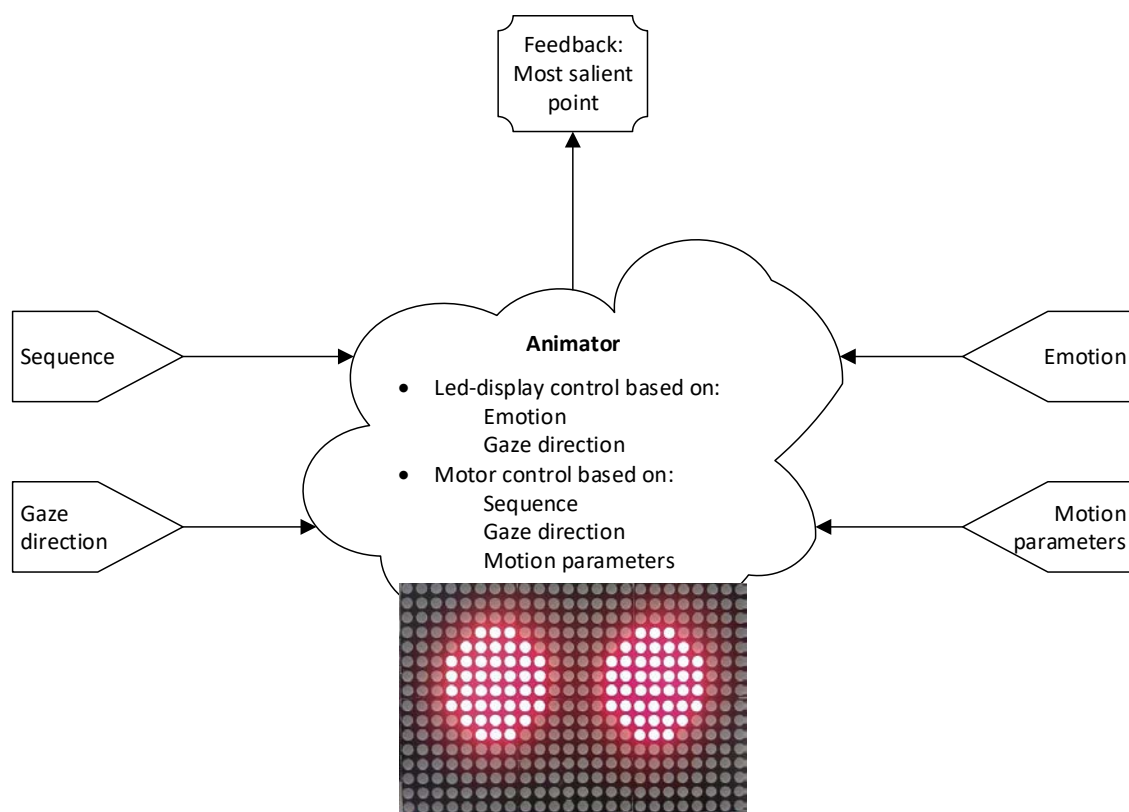


Figure 3.5: Animator

### 3.1.5 External communication

The communication between the robot and the external source (in this case AsapRealizer) will be done using an intermediate message broker, in this case Apache Apollo. Both AsapRealizer and ROS already have support for this communication method: for AsapRealizer it even holds that the larger system (as in Figure B.1 in Appendix B) communicates using the same message broker.

The communication with external software is bi-directional, but the two defined message types only work in one direction. The request message type is only possible towards the HMMM-part, while the feedback message type is only send outwards.

There are three request types (corresponding to the three HMMM parts):

- *Gaze*  
Contain saliency mappings, including an deliberate parameter to prevent a scaring reaction on deliberate behavior
- *Emotion*  
Contain valence and arousal data representing the emotion
- *Sequence*  
Contain certain predefined movement sequences, including their requested timing

There are three types of feedback messages defined:

- *Gaze*  
Provides a streaming feedback of the point that has the current interest.
- *Planning*  
Being send on sequence requests, it contains either an acknowledgment (ack) or an rejection (nack).
- *Progress*  
Progress feedback is send when specific states of an sequence is achieved. These states are start, stroke and end.

The exact message templates are described in Appendix C. For the planning feedback we can distinguish different subtypes:

1. *It is impossible to plan this request (nack)*  
It is not possible to execute the requested sequence. This might be the case if the requested time is too soon or has already past, or if the actuators are not available.
2. *Exact negotiation (-)*  
This feedback type will be used when the requester wants to know when a specific sequence can be planned such that it will be executed. The requester will need to send a new request with the required timing based on the negotiation result.
3. *Negotiation (ack)*  
This feedback will be used if the requester specified that the sequence should have a start on or after the requested time. This is a weak request, on which the feedback will contain the computed planning.
4. *Try to execute, but motion parameters are updated (ack)*  
If it is possible to achieve the timing by updating the motion parameters (within configured bounds), the parameters will be updated and will also be send as feedback.

5. *Will execute, but it will be late (ack)*

If the requested timing can not be met, but it can be met if it is within the configured flexibility limit (for example 50-100 ms), the sequence will be executed.

6. *Will execute on time (ack)*

If the requested timing can be met without problems.

Note that the timing requests can be made on the start, stroke and end synchronization points and that all feedback holds for every possibility. To be able to handle the stroke and end timing requests, the sequence mixer will calculate the expected duration of every sequence request from arrival as they are based on the currently active motion parameters.

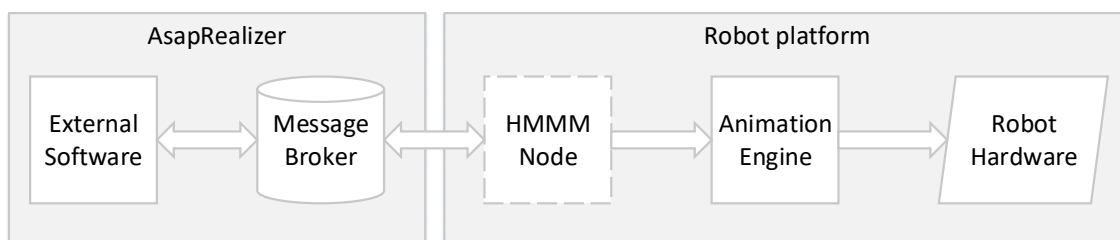
### 3.1.6 Parameters

Next to the internal motion parameters there are several other parameters that need to be configurable: the first one is the robot configuration which needs to be known beforehand and will not be adjustable on runtime. Other parameters are focused on the autonomous behavior: examples are the speed and amplitude of the breathing and blinking (which are actually the motion parameters in the EyePi). There are also some HMMM parameters, which are used to control for example the weight of the (external) inputs. The latter parameters need to be adjustable on runtime, as they can influence the final behavior of the robot significantly and can help with social interaction.

The parameters are implemented with the Dynamic Reconfigure component of the ROS, which means that they will not adjustable with the uses of the bridge yet. However, as it is a standard component which also use the ROS-messages, it should be possible to expose the parameters over the communication bridge in a later stage. Note that as the parameters values are generated with the help of a configuration file: these changes would probably be made twice. There is a graphical interface available to adjust the parameters, which is sufficient for now.

## 3.2 Software

The toolkit software that was available from previous work was sufficient for the old requirements, even though it stressed the processing unit unnecessarily. However, the existing functionality should remain intact, and the new functionality will also use the existing animation engine (in extended form) to avoid code duplication. The HMMM part will be placed in front of the animation engine, but should control the hardware through it (as also described in Section 3.1.4). This results in the following, simplified block scheme (Figure 3.6), in which the HMMM node is to be build and the platform to be improved as whole.



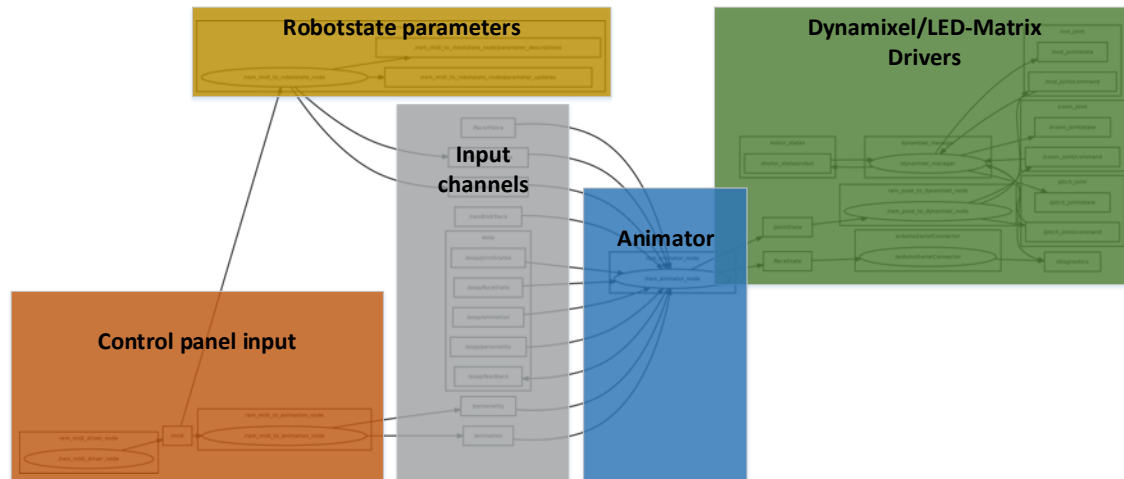
**Figure 3.6:** Simplified block diagram of the designed software solution

### 3.2.1 Existing functionality

The existing functionality was not build with a clear structure in mind: all nodes were defined separately and the data flow between them was not clearly defined. Next to that there are a lot of

differences in code-style between the different modules. However, as the software is functional, only the larger structure and naming will be changed.

The old structure is depicted in Figure 3.7, in which the application is not streamlined: it is not directly clear what the different modules are designed for and deeper code inspection is required to analyze the actual functioning.



**Figure 3.7:** The software structure as currently utilized in the existing toolkit

To improve the scalability, maintainability and re-usability of the toolkit has been refactored into logical blocks, which are easy recognizable and re-usable. There were nodes which had as their only function to forward a specific message type as another message type, resulting in the data being send twice. These nodes are removed and in the case they had any significant functionality, that functionality has been transferred to one of other sending or receiving nodes.

The existing functionality of the toolkit will be extended with the HMMM functionality (see Section 3.1) and the inputs needed for the autonomous behavior will be added. The possibility for external control will also be implemented.

### 3.2.2 Saccade movements

The original toolkit already features saccade-functionality, but that only works well when only the xy-direction of the eyes were being controlled. When any sequences such as dancing or nodding were activated, the saccades became unnatural eye movements. As the HMMM part will benefit from the saccade movements as it increases the lifelike behavior, the saccade behavior will be fixed to be more lifelike. The xy-motion will no longer be the same and the saccades have been made less intense. They also disabled on certain sequence to prevent constant eye-spinning. By default the eye saccade will have some smoothing, to prevent fast oscillations around the center.

### 3.2.3 System resource usage

The improvements in the list below were made to the the toolkit to reduce the system resource usage. At the same time these improve the stability and allow the addition of extra components on the same hardware.

- *New minimal Raspbian system image*  
By using the newly available minimal Raspbian system image, less resources are used by processes that are not needed for the functioning of the EyePi. The old installation

did not use this minimal image and was also bloated with software not needed for EyePi operation.

- *Changed MIDI-controller behavior*

The old toolkit used polling to retrieve the data from the MIDI-controller. Although it is not possible to change this into event driven behavior from the toolkit code, the library used to retrieve the data does use an event driven model which saves the data points ordered on time. By lowering the polling frequency (a factor 10) and handling multiple data points from the library in a single poll, the CPU usage has been significantly lowered (compared to the earlier full core utilization) while retaining the same experience.

- *Reduced logging to minimal*

By default, the toolkit would log every action and movement to the standard output, which is saved on disk. It did not use the ROS-API for logging, resulting in large log files, causing IO-waits and eventually a complete lockup due to a full disk. By using the ROS-API for logging, the default is to no longer log everything which results in less disk-usage and thus lower latencies. It is still possible to enable the logging if required.

- *Removal of dead code*

There were pieces of code that were active in the old toolkit, but were never used. Together with general code optimizations in other parts again a significant improvement is made.

With these improvements, the system currently does not fully utilize the available processing power of the Raspberry Pi computer, even with the new software components that are being added. By leaving this small margin there is more room for solving unexpected high loads during normal execution (which can happen). It also provides more possibilities for debugging when needed and the complete system responds faster when not fully utilized. Compared with the old hardware, on which any small action would result in a delay in the animator, the current system performs faster and is more stable.

### 3.2.4 Saliency detection

For the saliency detection two different methods were explored: one is based on using advanced algorithms that can predict the saliency of any (still) image, while the other is based on the detection of movement. The first option is classified as real saliency, the second option can not be classified as such, even though it is part of the conditions that make something salient (see Section 2.6).

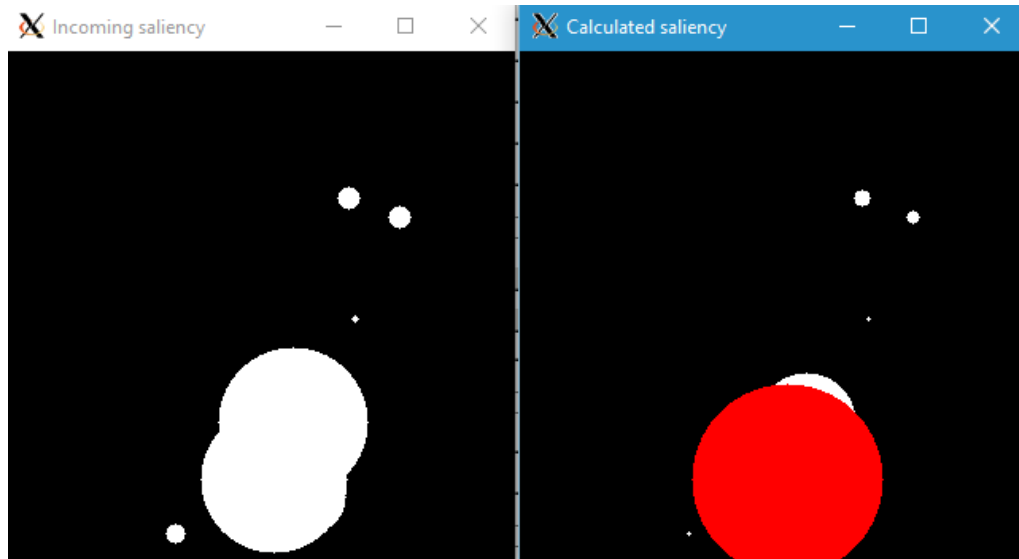
For the implementation on the EyePi hardware, the second option is chosen as it is the only viable option. The real saliency implementation needs too much resources to be able to run on the available hardware, especially when live footage is being used together with the requirement that the detection is semi-realtime.

In Figure 3.8 the output of the detection can be seen. The circles represent the multiple interest points (which form the saliency map when combined), while the size of the circle represents the amount of interest the corresponding point has. When an interesting point is no longer detected, the circle will shrink and will be removed after a predefined amount of time. The left image depicts the unprocessed output of the motion detection in the form of a saliency map. The right image shows the output of the calculations of the HMMM part, including the most salient point that is filled with red.

The internal camera abstracts the motion detection data into a saliency map. The weights of the points will depend on the size of the detected motion, as every motion detected will be in the map communicated to the HMMM part. Note that motion detection is not the same as saliency detection, although motion is considered part of the stimuli to make something salient. The



choice was made to have only motion detection running and not a complete saliency algorithm due to the limited amount of processing power available on the platform itself.



**Figure 3.8:** The output of the saliency detection of the autonomous EyePi

### 3.2.5 Save shutdown

In order to protect the operating system of the EyePi, a safe shutdown button has been implemented on the MIDI-panel. The older revision could only be turned off by removing the power completely, without shutting down all processes, risking the integrity of the data on the SD-card. Even though no problems occurred before, the shutdown button allows for easy and clean shutdown, which also speeds up the boot time as it is no longer needed to check the dirty filesystem after an unclean shutdown.

## 3.3 Hardware

The first hardware version had, despite its effectiveness in conveying emotions and being easy to build, some minor drawbacks in its design. By tackling the minor drawbacks it will be more complicated to construct (thus taking more time), but it is not expected that it will be problematic as construction is a one-time task, while the result will be easier in use.

### 3.3.1 Processing unit

#### Old situation

The processing unit (a Raspberry Pi 2B) was located in the moving head, causing more mass and the need for cables which are connected to the side of the head (see Figure 3.10). It also placed the complete power distribution in the head and it was not build to last a lot of movement.

#### Updated design

By locating the processing unit (still the same Raspberry Pi) outside the moving head, it looses mass and there is no longer the need for cables connected to the side of the head. The power distribution can be relocated to a more secure place, next to the power supply.

### 3.3.2 Camera location

#### Old situation

The camera was also located on the moving head, complicating image processing on the low-grade hardware. However, as the camera moves together with the head, the field of view is only limited by the head movement itself.

#### Updated design

With the processing unit moved, it is no longer feasible to have the camera on the head as the ribbon cable will most probably break with the movement. As the camera needs to be near the processing unit when using a Raspberry Pi and a ribbon cable, the best location is on the front of the casing, combined with a wide-angle lens. The lens creates a larger field of view, while image distortion is minimal compared to a fish-eye lens (which increases the field of view even more).

### 3.3.3 Display driver

#### Old situation

The display driver (an Arduino Micro) is positioned directly behind the display and connected to the processing unit with an USB-cable, resulting in a small cable loop at the side of the head.

#### Updated design

As the processing unit moved, the USB-cable needs to be extended to be able to connect again. However, with more space on the back of the display, the display driver location will be optimized, removing the visible cable loop at the side of the head.

### 3.3.4 Power supply

#### Old situation

The power supply exists of a small desktop power supply, which is permanently connected to the head with a simple cable. It converts AC into the needed 12V DC for the Dynamixel servos, while the voltage needed for the Raspberry Pi and peripherals is converted in the head itself.

#### Updated design

The power requirements should have not changed, but it will be relocated to be integrated on the same spot as the processing unit. This means that there will no longer be the need of a power cable to the head, while the power conversion can still be done near the processing unit.

### 3.3.5 Casing

#### Old situation

There was no specific casing: everything was combined between two plastic plates forming the head of the robot, while the base existed of a simple weight. The weight proved to be insufficient as base: it was possible to rock the robot over using the normal MIDI-controls.

#### Updated design

By relocating the processing unit (and peripherals) into a closed casing at the bottom, the robot should come over as more finished and be more consumer ready. With a larger footprint, although it has less weight, the robot should have a more stable base. Figure 3.9 contains an image of the case internals.



**Figure 3.9:** The internals of the new casing

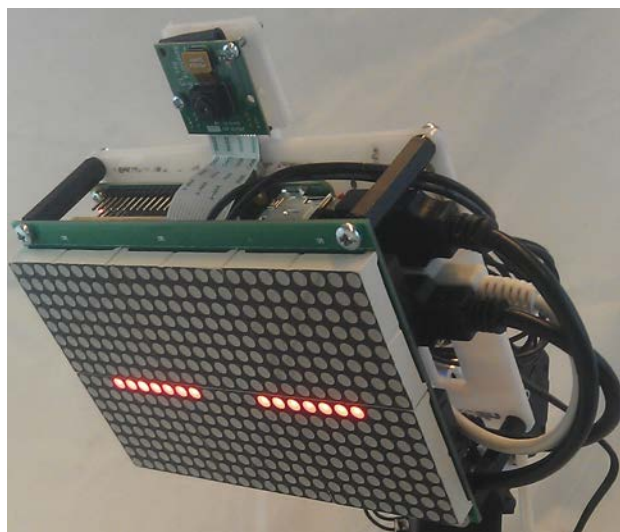
### 3.3.6 Portability

#### Old situation

With the cable loops at the side of the head and the permanently connected power supply cable it is inconvenient to move multiple robots in a single box. Power cables will tangle as they have loose ends and the hardware is vulnerable as it is basically in the open (mostly due to the cable loops, see Figure 3.10). The weight of the base is too high and it does not provide enough stability.

#### Updated design

With most components not directly accessible in a single casing, the hardware itself will be less vulnerable. At the same time there will be no unnecessary permanently connected cables to minimize the change of damage. When a cable can have a loose end during transport, it will be detachable. Without the base weight its total mass should have lowered significantly.



**Figure 3.10:** The original, vulnerable hardware

### 3.3.7 MIDI controller

#### Old situation

The MIDI controller had some labels indicating the function of the buttons, but they were already falling off. The internal LEDs of the controller were not used.

#### Updated design

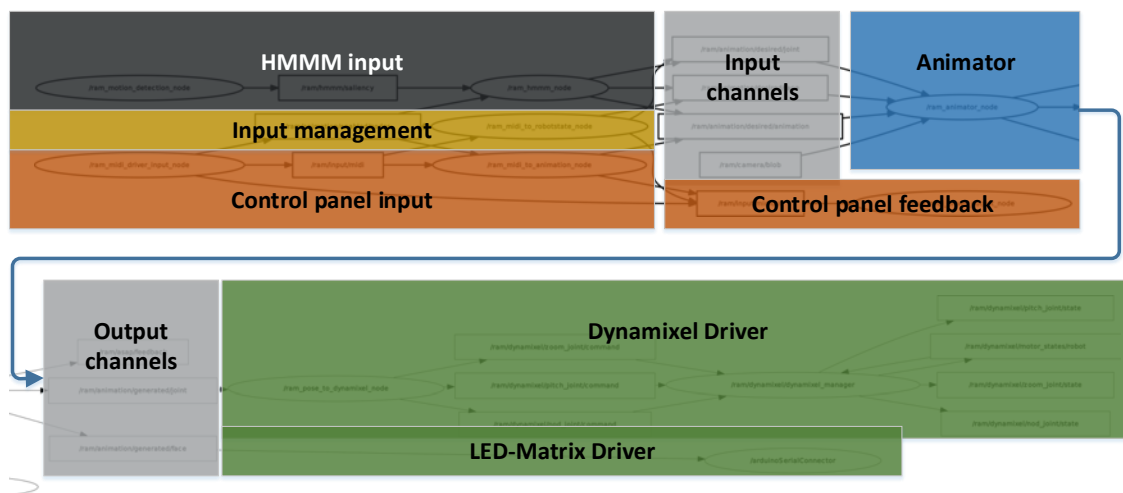
The new controller now has an full overlay indicating the button functionality and the internal LEDs are used to indicate the current status.



**Figure 3.11:** The MIDI-panel has been updated with an full overlay indicating the available controls

### 3.4 Final system

The final system will combine the existing and new software functionality and should significantly be improved on the hardware. At the same time the stability, scalability, maintainability, re-usability and system resource usage of the software will be improved, but the updated hardware will also affect those aspects. An overview of the final software is given in Figure 3.12, which shows a cleaner and straight to the point program flow. In total, the complete system should deliver smoother behavior, especially when combined with external software control, with minimal impact on the global look-and-view of the hardware.



**Figure 3.12:** The designed software structure, which minimizes communication channels

## 4 Results

### 4.1 Control design (HMMM)

Most parts of the design of HMMM as described in Section 3.1 have been implemented in the EyePi. The actuator check, the global sequence database and the motion parameters based on the emotion state are missing from the current implementation, but their absence has limited impact on the functionality of the whole system.

#### 4.1.1 Gaze mixing

While it is hard to test the gaze mixing quantitatively, it can be verified that the saliency maps are mixed correctly and that the corresponding EyePi motion confirms with the expectations. Points are chosen correctly and the EyePi indeed loses its interest in the currently most salient point gradually, to eventually select another point. There is no fast switching between two equally salient points detected due to the rewards and penalties.

#### 4.1.2 Validation of HMMM

To validate the final behavior when using HMMM a simple experiment has been held which should show the difference between a robot without HMMM and a robot with. Next to the EyePi robot, an humanoid robot called the Zeno is used. The Zeno can receive the same behavior requests as the EyePi, but it does not have the capability to autonomously react to other inputs or mix any request as it lacks HMMM.

For the test the EyePi has been used twice: in the first series (A) it uses its full capabilities with HMMM, but with the second series (B) the autonomous capabilities have been turned off. The Zeno has been used in the third test series (C). Stills from the tests are given in Figure 4.1.

There was a complication during the test execution: it appeared to be impossible to retrieve reliable joint state information from the robots without severely impacting the performance causing the complete interaction to become out of sync. It should also be noted that even though it appears the Zeno (series C) is not moving at all, that is not the case. The Zeno unfortunately only shows minimal movement with during the same test due to limitations in its control.

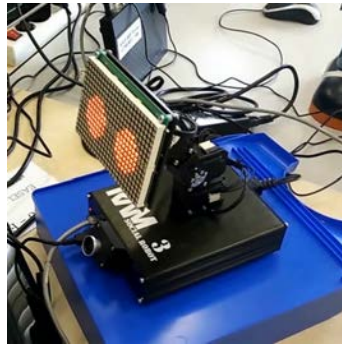
The executed scenario contains the following numbers steps, which correspond to the numbers in the figure. The robot is told to greet every person and tell them to take a seat.

1. Nobody in sight. All robots are in their initial position.
2. Person enters the field of view. All robots focus their view on the person and tell the person to take a seat.
3. Person still in view, but now in front of the robot. The robots finish their text. The EyePi with HMMM automatically blinks.
4. Something happens on the left (seen from the robots), while the person is still in the field of view. Only the robot with HMMM has the capability to react, as the others are still focused on the person.
5. The movement is gone, so the robot with HMMM will again focus on the person that is still there. The others robot did not move, except to follow the person in view.





A1



B1



C1



A2



B2



C2



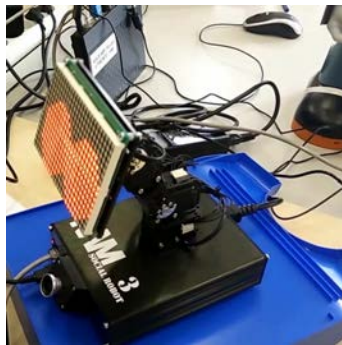
A3



B3



C3



A4



B4



C4



A5



B5



B5

**Figure 4.1:** The EyePi in series A uses HMMM, while the other two series (both an EyePi (B) and a Zeno (C)) do not use HMMM. In the scenario a person is greeted (which comes from the right in frame 2), while there is a sudden movement in the left at frame 4 (seen from the robots).

Even though the HMMM shows the requested behavior, a user study should be done to conclude if this robot is received as a fluent and lifelike robot. Due to time constraints such a study is not done, but it is still recommended to have one in a later stage as it might indicate any improvement possibilities. The experience until now (combined with the responses on the eINTERFACE'16) is however that the robot is received as such.

## 4.2 Software

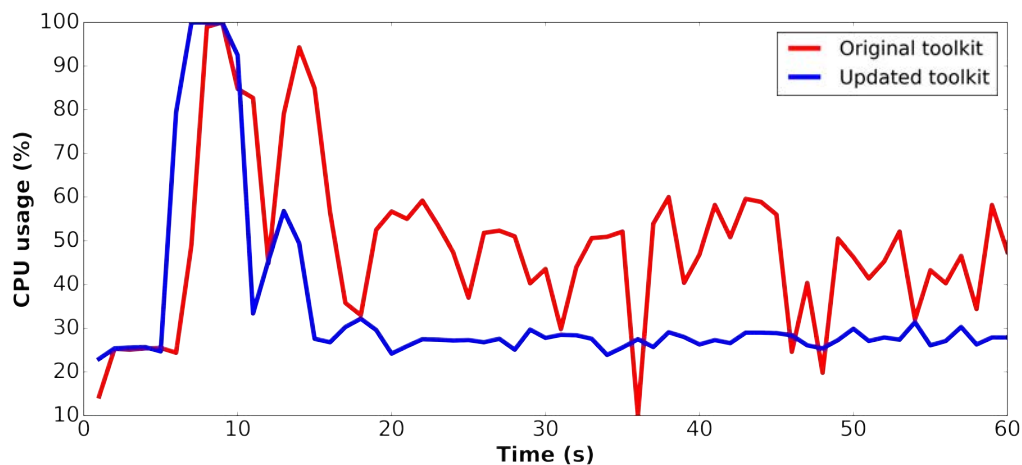
The software has been updated in order to improve the scalability, maintainability and re-usability. The ability of motion detection has been checked by trail and the performance of the updated toolkit is compared to the performance of the original toolkit.

### 4.2.1 Performance

The changes made to improve the system performance have been validated by running a monitoring tool during the execution of the complete toolkit. The monitoring tool takes a resource usage snapshot every second for 60 seconds. Both the original and the updated toolkit are started after the monitor has started its measurements. After the toolkit has been booted it is put into demo mode manually using the MIDI control panel. The results are given in Figure 4.2, Figure 4.3 and Figure 4.4.

The CPU related results show a large improvement in CPU-usage and process switch rates. The first peaks in the CPU-usage graphs are similar and are caused by the initialization of the toolkit, but directly after initialization the improved toolkit is less resource consuming and more constant during execution.

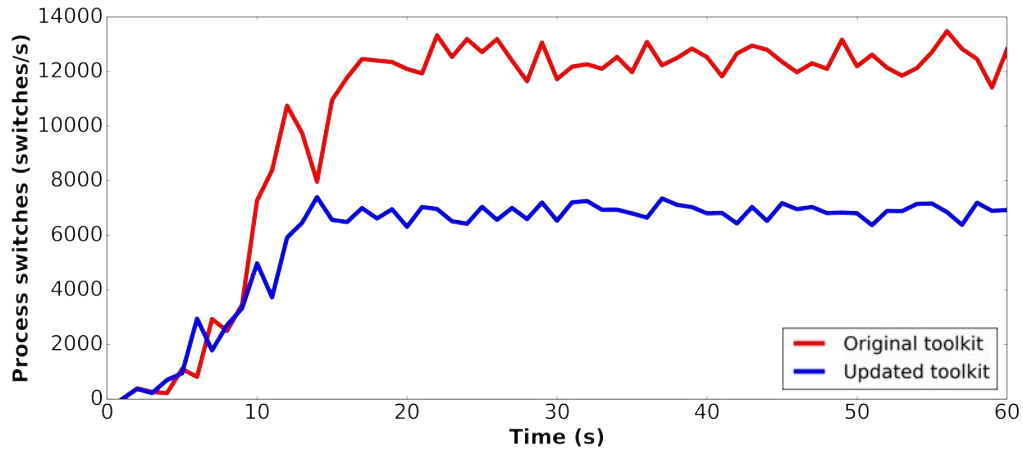
An important note is that the motion detection node is also running with the updated toolkit, which is not the case with the original toolkit. ROS is not build for dynamic node activation and as the initialization of the camera node takes a relatively large amount of time, it was chosen to always execute it. The processor graphs show that this has been a correct choice.



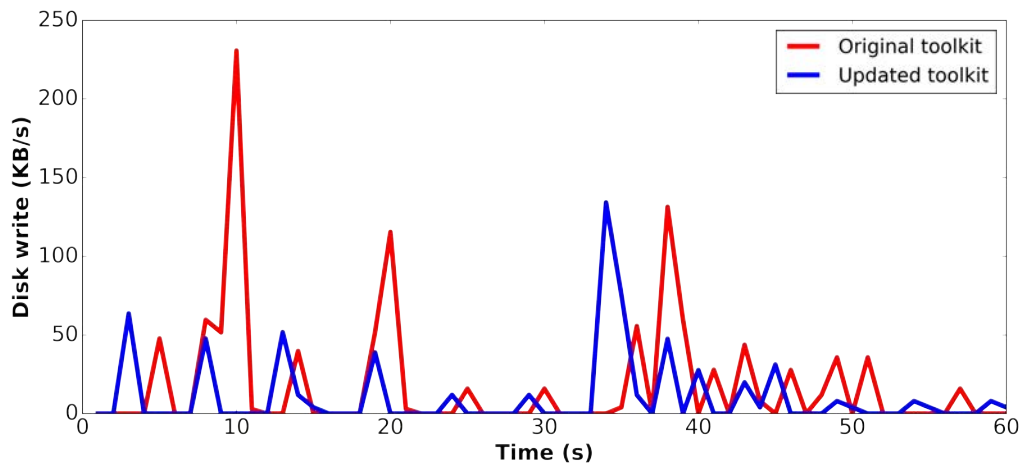
**Figure 4.2:** CPU usage of the original and the improved toolkit

The expected improvements in disk usage can not be concluded from the data in Figure 4.4. This is caused by the used monitoring tool, which writes the monitoring data to a file on the disk. Even though the graph is not conclusive, the created log files by the improved toolkit are significantly smaller as all `printf` statements from the code are replaced with ROS log calls. As these statements are only executed when the toolkit is run in debug mode, the disk usage is less and it is no longer possible to have a full disk after a day.

A final side note needs to be given on the comparison of the performance. Both tests are run on the newly installed minimal Raspbian system image. Even though the test results are therefore



**Figure 4.3:** Process switches per second of the original and the improved toolkit



**Figure 4.4:** Disk writes of the original and the improved toolkit

better comparable, the new system image should have reduced the system load even further as there are less OS-services running. This means that the improved toolkit might even perform better than the current graphs suggest. Unfortunately, there is no data available from the old installation. Secondly it should also be noted that the improved toolkit also constantly computes a saliency map based on the movement data, which are extra calculations compared to the existing toolkit. The calculations are always done as it is not feasible to start and stop the nodes during runtime.

#### 4.2.2 Saliency detection

While it is hard to test the motion detection quantitatively, it is verified that motion is detected correctly and that the correct weights are given based on the size of the movement. Even though it is not pure saliency detection, it works sufficient for the required autonomous behavior.

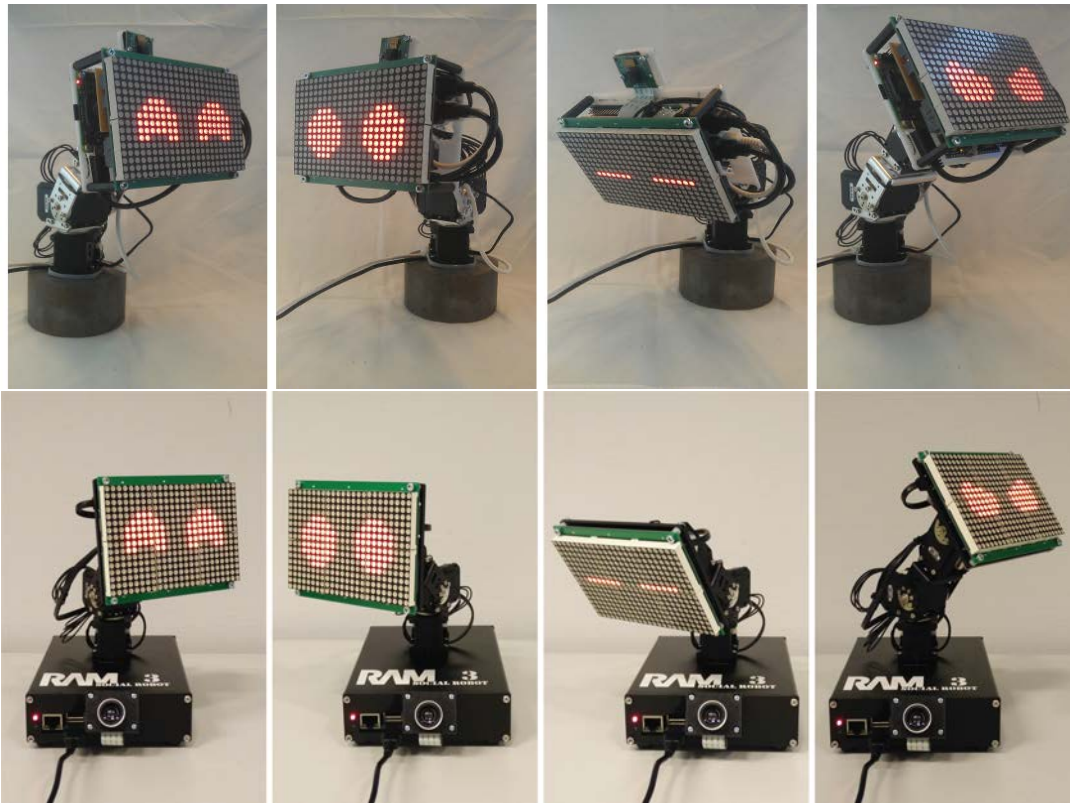
### 4.3 Hardware

The functional hardware has not significantly changed during this project: the robot still contains the same basic hardware. Even though some minor changes have been made, most of them were done to the location of the individual components (see Section 3.3). The changes in appearance can reviewed in Figure 4.5.

Below is an overview of the changed hardware:



(a) The original hardware (Watanabe et al., 2013)



(b) The improved hardware based on the original hardware (Figure 4.5a)

**Figure 4.5:** Comparison of the two hardware versions

- **Camera:** The camera is now fixed on a steady base and has a new wide angle lens.
- **Casing:** There was no casing containing the Raspberry Pi and power supply before.
- **Dynamixel servos:** Due to the lower weight in the head smaller servos are used.
- **New power supply:** The power supply that was used before did not fit in the new casing.
- **Controller LEDs:** Some buttons now give feedback with the build in LED.
- **Control labels:** Full overlay with labels, as the old labels loosened.

### 4.3.1 Portability

With all hardware being positioned save within the casing, there is less chance of damaging any of it. At the same time, the external power supply is integrated, resulting that only a simple power cable is needed to power the whole system, instead of having the power supply external an permanently connected to the EyePi head. The total weight of the robot has also been reduced significantly. While the base weight of the previous build had a mass of 2.5 KG (which is excluding everything else, such as the adapter, robotic head and dynamixels), the total weight of the EyePi is now only 1.26 KG (including everything that was excluded before).

### 4.3.2 Cameras

The three EyePi's are now fully functional with the new hardware and they should be completely identical. However, one of the EyePi's has a different camera module mounted, namely version 2 of the Raspberry Pi camera. The Raspberry Pi camera version 1 was no longer available, forcing use the newer version and creating a small difference between the otherwise identical three EyePi's. However, even though the specifications of the two versions are not equal, it is possible to create the same pictures with the same code without any problems. This results in

the conclusion that there is no significant difference between the two cameras when they are used with the current toolkit.

#### 4.3.3 Evaluating the hardware changes

Evaluation of the new hardware design is not a trivial task to do: both designs have their positive and negative points. Easy to conclude improvements are a higher stability due to the larger footprint of the base and a smaller, more agile head due to the moved processing unit.

During an open day for elementary school students of the research chair lab the response to the new hardware revision was enthusiastic (people asked if they could take it home with them). The same result is achieved during the eNTERFACE'16 workshop: researchers from across the world were interested in even acquiring one for their own research.

Even though the older hardware version is no longer available, qualitatively it can be said that the new hardware design is being liked in general (fulfilling requirement 10). It also proved to be more stable: it did move across the table surface when being operated roughly, but it moved a lot less than the old hardware would have done. Another point of interest are the cables of the power supply and MIDI-controller, which are no longer connected to the moving head itself, allowing easy control without worrying about the state of the cables and it improves the sleekness of the design.

The portability (requirement 9) has fulfilled: not only has the weight been reduced by more than 50%, it is also less vulnerable during transport. Without any external components and cables that can be easily removed multiple EyePi's can be packed in the same box without any chance of cable tangling.

For the costs requirement (11) the same conclusion can be made: the EyePi is cheaper in construction as the Dynamixel servos used in this version are less expensive due to the fact that they are smaller. A complete bill of materials can be found in Appendix D.

It can be concluded that the new hardware design has successfully fulfilled the requirements as formulated in Section 2.7.

#### 4.4 Final system

The final result of this project is HMMM, an algorithm to mix multiple (external) control inputs into viable robot behavior, that is fluent en lifelike without only executing the request. HMMM allows the robot to listen to the director (the source of the behavior requests) and executing the requested behavior, but when there is a more pressing matter it will (temporarily) defer from its task to focus on it. The functionality has been implemented in a human robot interaction toolkit from within the chair, during which the performance of the toolkit has been improved significantly. Finally, the hardware that is being used to demonstrate the toolkit has been improved successfully.

## 5 Conclusion and recommendations

### 5.1 Conclusion

The requirements as formulated in Section 2.7, for both HMMM implementation and hardware improvements, have all been fulfilled.

#### 5.1.1 Control design and software

1. *External software must be able to control the hardware with exposed high level controls*  
External software can control the hardware using the exposed high level emotion, gaze and sequence parameters.
2. *Autonomous behavior needs to be available from within the robot hardware*  
Autonomous behavior is available in the form of motion detection, breathing, blinking and saccade movements. There is also a shock reaction implemented.
3. *The autonomous behavior needs to be combined with external controls*  
Due to the implementation of HMMM, the autonomous behavior can seamlessly be combined with external requests.
4. *The resulting behavior needs to be fluent and natural (lifelike)*  
With the result of the EyePi running HMMM compared to the results from the Zeno, it is concluded that the resulting behavior is indeed fluent en lifelike. The EyePi does not have any dead time and is always doing something.
5. *The available camera needs to be used for autonomous behavior*  
The available camera is used for the motion detection, which is used for autonomous gaze behavior.
6. *Operation parameters for the autonomous behavior and mixing need to be adjustable on runtime when feasible*  
All parameters are currently adjustable on runtime, even though not through the message bridge.
7. *The external control connection should not be tailored to a specific application*  
By using the portable XML-format which abstracts the ROS-messages every application should be capable to communicate with the EyePi. Together with the intermediate message broker, a connection should always be possible.
8. *The software needs to be easy reusable in other projects*  
Every part in the project can be reused in other code. When ROS is not used in the other project, it will be necessary to rework the code for the different platform.

#### 5.1.2 Hardware

9. *The portability of the robot needs to be improved*  
Due to the weight loss and the lack of external cables and adapters, the portability has been greatly improved.
10. *Changes made to the hardware may only have a minimal negative impact on the look-and-feel of the robot*  
Even though no there was no questionnaire to verify the new design, it can qualitatively be said that the EyePi is still being liked in general due to the positive reactions on it during the eNTERFACE'16.

11. *The total cost of the robot should remain about the same*

Since the motors have been replaced with smaller variants, the motor costs are lower. Extra material has been used for the construction of the robot and the original weight has been replaced with a casing, but the changes in costs are probably not significant. However, the exact costs of the old hardware are not available, but they are expected to be lower.

### 5.1.3 Research questions

In the start of Chapter 2 the initial research questions were formulated, which have been answered during the course of this research.

1. *What are the direct controls that need to be exposed to a behavior realizer?*

There are no direct controls needed for a behavior realizer to control robot hardware. Abstracted controls are sufficient.

2. *What are the operation parameters that need to be configurable?*

It is convenient to have parameters configurable that directly influence the robot motion. However, it is not required to have any parameters exposed as the HMMM part can already handle most of them.

3. *What is the most suitable communication schedule for the communication between the robot hardware and the behavior realizer?*

As ROS uses its own message format, a bridge that uses a standard format is required. This implementation uses XML-messages, which abstract the ROS-message format. The exact implementation does not matter, but it should be a format that is easily generated in several programs.

4. *Will the behavior/executed actions be more fluent when the mixing is done on the robot hardware?*

The behavior is more fluent when the mixing is done on the robot hardware. The robot has direct access to its internal state and always has access to the autonomously generated data. This means that the robot will always work, even if there is no external connection, allowing it to maintain its fluent and lifelike behavior.

5. *How can different control inputs be mixed and what are the implications of the mixing on the different control levels?*

With the definition described in Section 3.1 it is possible to mix the different control inputs. Due to the mixing, full control of the robot hardware will be lost, but it is no longer needed to add lifelike behavior yourself (such as breathing and gazing).

6. *Is end-effector control more convenient than joint-space control in relation to HMMM?*

In this use case it is more convenient to use joint-space control than end-effector control, due to the fact that you want to have the possibility to control one actuator specifically.

### 5.1.4 General notes

This project resulted in a functional HMMM, which proved to generate more fluent and lifelike behavior than another robot which is being controlled by the same commands. Changing the robot from a simple puppet that can only be controlled completely into an actor that does what it is being asked, but uses its own interpretation, has been successful. As the models and algorithms created during the project are not yet fully implemented, it might even be possible to improve the fluency and lifelikeness.

## 5.2 Recommendations

Even though the result of this project complies with the formulated requirements, there is always room for improvement and extra features. For the EyePi a couple of recommendations can be made, ranging from HMMM improvements to hardware adjustments, but also extra features. The recommendations are as follows:

- *Full sequence mixing implementation*

The sequence implementation of the EyePi has not yet been fully implemented. It currently lacks an actuator check, which it has been described extensively in Section 3.1.3. Together with the implementation of the actuator check, a well designed database should be added that can easily be filled and reused.

- *Redesign sequence animation*

Even though the current implementation has a sufficient functioning sequence animator, it is relatively hard to define a new sequence, especially when they need more than one distinct and repeating movement. Combined with the database from the previous recommendation, it should be possible to describe and implement new sequence relatively straightforward.

- *More autonomous actions*

Currently, the EyePi only features an autonomous shock reaction, activated on large sudden movements. However, more emotions can be triggered autonomously such as falling asleep or simple looking around when no motion is detected.

- *External parameter interface*

For certain parameters it might be convenient to have them exposed on the outer level. Currently, all parameters are configurable when you have SSH-access to the system by using `rqt_reconfigure`, but not all of them should be available for adjustment on the outside.

- *Computing unit upgrade*

The EyePi uses an outdated Raspberry Pi 2 Model B as computing unit. There is already a newer version available, the Raspberry Pi 3 Model B. The newer model should be 50-60 percent faster than the older one and an extra advantage of the update would be the access to a builtin WiFi-module. However, with the current metal enclosure the usage of the module might be challenging.

- *Moving camera*

Currently, the camera that is used for motion detection is mounted stationary to ease the motion detection. The disadvantage of this method is that the field of view is limited in comparison to the motions the head can make, even with the wide angle lens that is used. However, when the camera is moving the current motion detection will no longer work, so new algorithms need to be used to determine the motion correctly.

- *Autonomous MIDI-controller*

The MIDI-controller is currently only used for emotion control when the robot is in autonomous mode. However, it would be interesting to use the MIDI-controller to create multiple stimuli for the HMMM which can be combined with external software (if there is any).

- *User study*

As the toolkit is used for social interaction, it is recommended to conduct an user study to verify the fluency and lifelikeness of the result. Due to time constraint this was not done, but it might deliver interesting improvement points.

## A Demonstration instructions

The EyePi can be used for an (interactive) demonstration. There are three modes in which the EyePi can operate, one of which is interactive:

- Manual control (interactive)
- Demonstration mode
- Autonomous mode

To boot the system, first plug in a power cord and connect a MIDI-panel to one of the USB ports. If you want to use autonomous mode with external input, you will also need to connect an ethernet cable. All systems are registered for the university network with static configuration, which means that they can be accessed (user `pi`, password `SocialRobot`) with their hostname (note that this only works in the Carré building):

- EyePi 1: `eyepi1.ewi.utwente.nl`
- EyePi 2: `eyepi2.ewi.utwente.nl`
- EyePi 3: `eyepi3.ewi.utwente.nl`

Flip the power switch on the back into the on state to boot the system. It will automatically boot all components.

To safely shutdown the system, press the 'SHUTDOWN' button. It will automatically shut down all components after positioning the head in a safe position. When the activity led (green) stops blinking, it is safe to use the power switch on the back.

### A.1 Manual control

Use the sliders and control the head!

### A.2 Demonstration mode

To put the system in demonstration mode, press the 'DEMO' button after boot. It can be stopped by pressing 'STOP'.

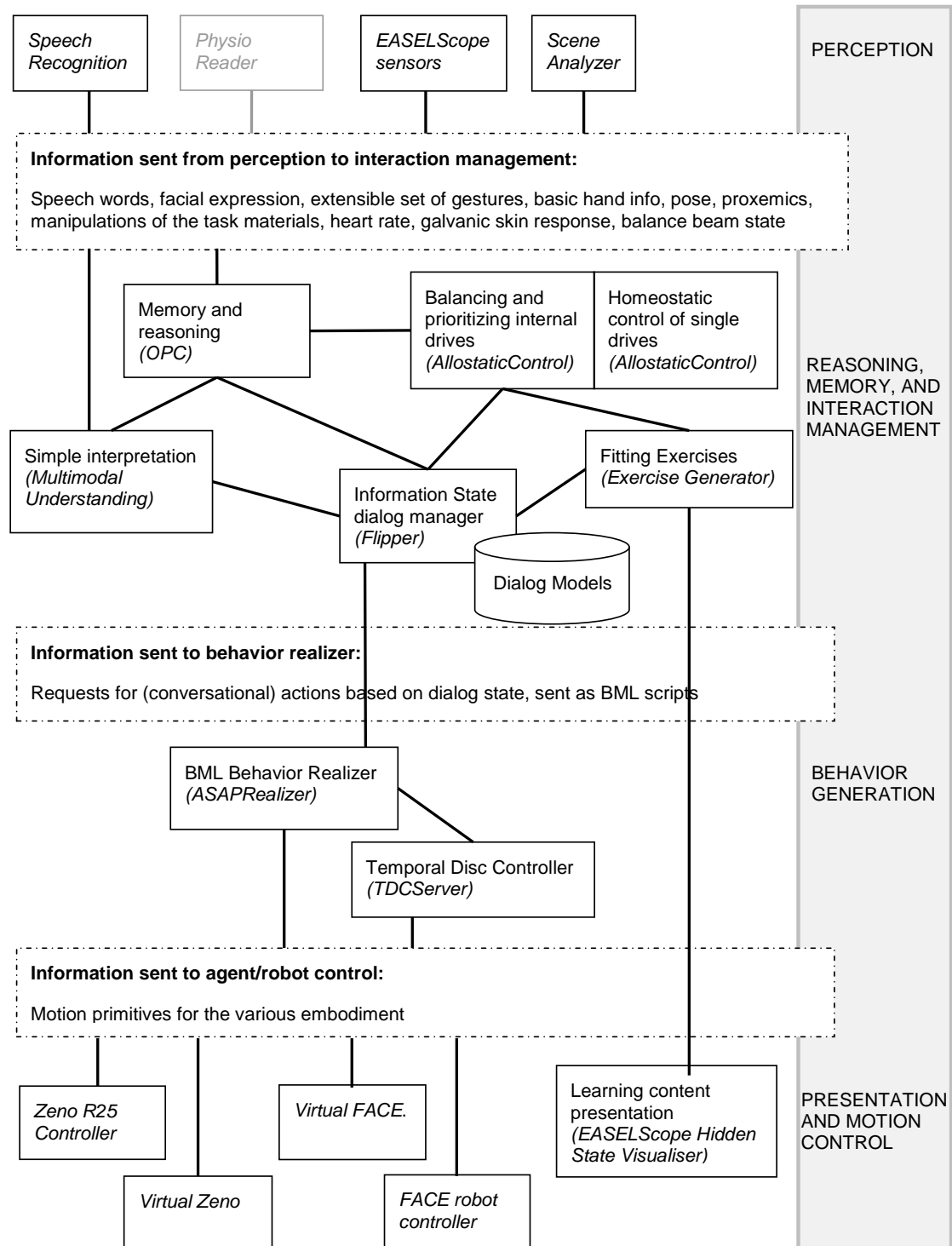
### A.3 Autonomous mode

To put the system in autonomous mode, press the 'CYCLE' button after boot (it should be lit for this mode). You can switch back to manual control/demonstration mode by pressing 'CYCLE' again. Note that the arousal/valence sliders can still be used.

The external software bridge is automatically booted during startup and tries to connect with the last configured IP. You can edit the IP by logging on and editing  
(`~/ramSocialRobot/src/ram_asap/asap_ros_bridge/scripts/relay.py`).  
After a change, the node can be restarted with the new IP-address by executing  
`roslaunch asap_ros_bridge relay.py`.

## B EASEL architecture

The Expressive Agents for Symbiotic Education and Learning (EASEL) architecture (Figure B.3) is a software architecture developed at the University of Twente (2015).



**Figure B.3:** Overview of the EASEL architecture, divided into four groups of modules. This structure can be used to realize a complex dialog system.

EASEL-D-WP4-UT-150917-D4.2 v2 Final

## C External communication

### C.1 Communication basics

This document shortly describes the possibilities the Human Robot Interaction Toolkit offers with respect to external inputs. Within the system, there are three inputs channels and three feedback channels defined, which can be used by any external program to send wanted values to the hardware. The several (external) inputs are mixed by the HMMM component of the EyePi.

#### Input

- Saliency (streaming) (ROS-topic: `/ram/animation/hmmm/saliency`)
- Emotion (id-based) (ROS-topic: `/ram/animation/hmmm/emotion`)
- Sequence (id-based) (ROS-topic: `/ram/animation/hmmm/sequence`)

#### Output

- Gaze (streaming) (ROS-topic: `/ram/asap/feedback/gaze`)
- General (id-based) (ROS-topic: `/ram/asap/feedback/general`)
- Prediction (id-based) (ROS-topic: `/ram/asap/feedback/prediction`)

The input and output channels can be accessed with on of the following two methods:

- A ROS-node which publishes to or reads from the corresponding ROS-topic
- Settings up a connection with the Apollo broker middleware

Both methods should deliver equal results, although a higher latency can be expected when using the Apollo broker. There are two main reasons for the expected higher latency: there is an extra conversion step to convert the messages and there is a network layer involved.

When numbered lists are used to list certain types, the list numbers are also used to encode the type.

### C.2 Saliency

It is possible to send saliency maps to the EyePi, which contains information about the detected salient points. The map is communicated with the message template defined in Codeblock C.1. Multiple salient points can and should be communicated in the same message. An update rate of at least 15Hz (which is the same as the internal motion detection) is preferred.

The `locationX` and `locationY` attributes must have a value  $\geq -1$  and  $\leq 1$ . The `weight` attribute must have a value of  $\geq 0$  and  $\leq 10,000$ . Internally, the location values are mapped to an 640x640 frame, which is located in front of the robot, seen through it's own camera. The camera on the robot uses a 640x480 stream, which means that not everything in the Y-direction is mapped.

The `inputId` parameter is used to distinguish the separate input streams. It should have an static id for every connection and it is used internally to calculate the interest decay.

The `deliberate` parameter defines if the saliency map should be treated as an autonomously generated map, or as a deliberate one. When a map is marked as deliberate, it can not trigger an autonomous shock reaction.

```
1 string inputId
2 uint8 deliberate
3 float32[] locationsX
4 float32[] locationsY
5 uint16[] weights
```



**Codeblock C.1:** Saliency message definition

The Apollo example (Codeblock C.2) communicates two salient points and their weights (in a single update) to the toolkit. In the example the salient points are  $(-0.6, -0.6)$  and  $(0.9, 0.9)$ , with a weight of respectively 1000 and 700. They are being send with the id `walle1` and are set to be deliberate.

```

1 <data>
2   <inputId type="str">walle1</inputId>
3   <deliberate type="int">1</deliberate>
4   <locationsX type="tuple">
5     <value type="float">-0.6</value>
6     <value type="float">0.9</value>
7   </locationsX>
8   <locationsY type="tuple">
9     <value type="float">-0.6</value>
10    <value type="float">0.9</value>
11  </locationsY>
12  <weights type="tuple">
13    <value type="int">1000</value>
14    <value type="int">700</value>
15  </weights>
16 </data>

```

**Codeblock C.2:** Saliency Apollo message example

**Note** that when there are no updates, the salient point will be removed after a configurable timeout. When a point keeps getting reported, the saliency will be updated with the new data. When a point is used, it will lose saliency after getting an initial bonus. After loosing focus, a penalty will be given, after which the saliency can recover. More information about the process can be found in Section 3.1.2.

Every time a new most salient point is calculated, a gaze feedback message will be published containing the salient point data.

Property	Value
Channel	/ram/animation/hmmm/saliency
Channel type	Streaming
Minimum refresh rate	15Hz
ID	String
Deliberate	Boolean
X-range	float: [-1, 1]
Y-range	float: [-1, 1]
Weight-range	int: [0, 10.000]

**Table C.1:** Properties of the saliency input**C.3 Emotion**

The robot emotion is defined in valence and arousal, which is illustrated in Figure C.1. By adjusting the values, the exact emotion can be communicated. Currently, new values will be mixed with the current value, allowing emotion fading. However, this only works for small changes. If the change in emotion is large enough (which is configurable), the emotion change will be done directly.

The message is defined as given in Codeblock C.3, with an Apollo example in Codeblock C.4. The arousal and valence values can be anything between -1 and 1. The weights can be used to speed up the mixing process if required. By default, the MIDI-controller uses a weight of 40 for both properties.

```

1 string id
2 float32 arousal
3 uint16 arousalWeight
4 float32 valence
5 uint16 valenceWeight

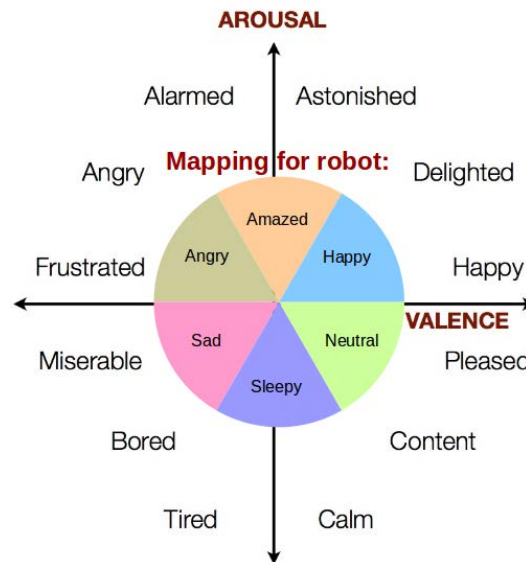
```

**Codeblock C.3:** Emotion message definition

```

1 <data>
2   <id type="str">#unqid</id>
3   <arousal type="float">0</arousal>
4   <arousalWeight type="int">10</arousalWeight>
5   <valence type="float">0</valence>
6   <valenceWeight type="int">10</valenceWeight>
7 </data>

```

**Codeblock C.4:** Emotion Apollo message example**Figure C.1:** Arousal and valence to robot emotion mapping


The incoming emotion message will trigger a feedback message of the acknowledged type.

Property	Value
Channel	/ram/animation/hmmm/emotion
Channel type	ID-based
ID	String
Arousal-range	float: [-1, 1]
Valence-range	float: [-1, 1]
Arousal-weight-range	float: [0, 100]
Valence-weight-range	float: [0, 100]

**Table C.2:** Properties of the emotion input

## C.4 Sequence

The last communication method can be used to request predefined sequences. The definition is given in Codeblock C.5. The following sequences are supported:

1. Dance
2. Nod 
3. Shake
4. Point at the left
5. Point here
6. Point at the right

For the timing request the following types are supported:

1. Start at.

2. Stroke at.
3. End at.
4. Predict timing.

Note that the ‘Predict timing’ type will only return the predicted timing as a special feedback message, but it will not process the sequence any further. The format of the message is given in ??

The value given in the `request` attribute is parsed as seconds and is added to the current time to form the requested time. With `requestCount` it is possible to specify how many times a sequence needs to be repeated.

```
1 string id
2 uint8 sequence
3 float64 request
4 uint8 requestType
5 uint8 requestCount
```

**Codeblock C.5:** Sequence message definition

An Apollo example is given in Codeblock C.6, which requests that the EyePi executes one nod in 400ms.

```
1 <data>
2   <id type="str">sequence_dance_1</id>
3   <sequence type="int">2</sequence>
4   <request type="float">400</request>
5   <requestType type="int">1</requestType>
6   <requestCount type="int">1</requestCount>
7 </data>
```

**Codeblock C.6:** Sequence Apollo message example

Sequences requests can trigger two types of planning feedback, containing either acknowledged or rejected. After a sequence has been acknowledged and planned, there will be progress feedback during the execution of the sequence, on the start, stroke and end synchronization points of the sequence.

Property	Value
Channel	/ram/animation/hmmm/sequence
Channel type	ID-based
ID	String
Sequence	Sequence ID
Request	float: [0.1, $\infty$ ]
Request type	float: Request type ID
Request count	float: [1, $\infty$ ]

**Table C.3:** Properties of the sequence input

## C.5 Feedback

There are three types of feedback messages being used: gaze, prediction] and general feedback. The general feedback is a group of two feedback types which use the same communication channel and template: planning and progress.

### C.5.1 Gaze feedback

Gaze feedback is send on every iteration of the HMMM node, which runs on 50Hz. The gaze feedback message contains the current most salient x,y-coordinate.

```
1 float32 x
2 float32 y
```

**Codeblock C.7:** Gaze feedback message definition

### C.5.2 Prediction feedback

Prediction feedback is only send when it has been requested explicitly. The feedback message contains the id of the request and the stroke and end timing of a single sequence.

```
1 string id
2 float32 stroke
3 float32 duration
```

**Codeblock C.8:** Gaze feedback message definition

### C.5.3 Planning and progress (general) feedback

The planning and progress feedback feedback messages share the same message definition to be able to share the same communication channel:

```
1 string id
2 uint8 type
3 uint8 message
4 uint8 reason
```

**Codeblock C.9:** Planning and progress feedback message definition

They both contain the identifier that corresponds with the request that triggered the feedback message. The feedback type can either be 1 (feedback) or 2 (progress).

#### Planning type

For the feedback type, one of the following messages can expected:

0. Nack
1. Ack

The planning feedback can have one of the following reasons:

0. Empty. No reason given.
1. Fluency. The requested behavior conflicts with other behavior.
2. Too soon. The requested timing can not be met.
3. Invalid. The requested behavior is invalid.

#### Progress type

The progress type does not use the reason attribute, as the messages are always triggered by the execution of a specific sequence on its start, stroke or end synchronization points. The synchronization points are encoded as follows:

0. Start
1. Stroke
2. End

## D Bill of materials

The bill of materials below contains all mechanical parts required (excluding common cabling, crews, nuts and custom PCBs) to build one EyePi. Note that stickers and overlays are also excluded, but they can be made within the research chair. Prices are estimates and rounded up, dollar prices are converted one-to-one to euro prices.

**Table D.1:** Mechanical parts list for one EyePi

Part	Description	Amount	ECPP <sup>i</sup>	ETC <sup>ii</sup>
Servo motors	Dynamixel MX-28	3	200 €	600 €
Midi controller	Korg Nanokontrol 2	1	45 €	45 €
Raspberry Pi	Newest version available	1	35 €	35 €
Display	Adafruit 16x24 RED LED Matrix Panel	1	25 €	25 €
Camera	Raspberry Pi Camera Module	1	25 €	25 €
USB-RS485	FTDI USB-RS485 converter	1	25 €	25 €
Power supply	12V 5A (7773146 at RS)	1	20 €	20 €
Display driver	Arduino Micro	1	20 €	20 €
Enclosure	(B)1304 Full Aluminum	1	20 €	20 €
DC-DC converter	XP Power SR10S05	1	10 €	10 €
Lens	0,67x wide angle lens (magnetic)	1	10 €	10 €
SD Card	At least 8GB	1	5 €	5 €
Power plug	230V power plug (1516058 at Farnell)	1	4 €	4 €
Micro-USB Cable	Flat (VLMP60410B1.00 at Nedis)	1	2 €	2 €
Leds	Red & green 3mm led	2	1 €	1 €
IO plate	IO plate on the front (lasercut)	1	0 €	0 €
Power plate	Power plate on the back (lasercut)	1	0 €	0 €
Camera housing	IO plate on the front (3d-printed)	1	0 €	0 €
<b>Total costs</b>				<b>847 €</b>

<sup>i</sup> Estimated cost per piece    <sup>ii</sup> Estimated total cost

The largest part of the total costs originates from the servo motors. The Dynamixel servo motors can be controlled over RS485, by simply sending the wanted position to the servo itself, without the need of an dedicated controller as it is embedded in the servo. It is possible to create the robot without these advanced servo to reduce the costs, but it will require creating extra controller code.

Furthermore, some parts are estimated on no cost at all and some of the standard parts (nuts, bolts, wires and more) were excluded. In Table D.2 the total costs of one EyePi is estimated.

**Table D.2:** Total costs for one social robot

Part	Amount
Parts	± 850 €
Miscellaneous	± 50 €
<b>Total costs</b>	<b>900 €</b>

## Bibliography

- Breazeal, C. (2002), *Designing Sociable Robots*, MIT Press, Cambridge, MA, USA, ISBN 0262025108.
- Breazeal, C. and B. Scassellati (1999), How to build robots that make friends and influence people, in *Intelligent Robots and Systems, 1999. IROS '99. Proceedings. 1999 IEEE/RSJ International Conference on*, volume 2, pp. 858–863 vol.2, doi:10.1109/IROS.1999.812787.
- Brooks, R. (1986), A robust layered control system for a mobile robot, **vol. 2**, no.1, pp. 14–23.
- Gennep, B. v. (2013), *Believability is in the Eye of the Beholder*, Master's thesis, University of Twente.
- Granot, R. (2008), Introduction to Robotics: Chapter 11. Subsumption architecture, Course slides.  
[http://math.haifa.ac.il/robotics/Presentations/pdf/Ch11\\_Subsumption.PDF](http://math.haifa.ac.il/robotics/Presentations/pdf/Ch11_Subsumption.PDF)
- Heylen, D. (2006), Head gestures, gaze and the principles of conversational structure, **vol. 3**, no.3, pp. 1–27.
- Jiang, R. and D. Crookes (2012), Visual Saliency Estimation through Manifold Learning.  
<https://www.aaai.org/ocs/index.php/AAAI/AAAI12/paper/view/5068/5364>
- Oosterkamp, J. (2015), *Creating a toolkit for human robot interaction*, Msc report 015ram2015, University of Twente.
- Reilink, R. (2008), *Realtime Stereo Vision Processing for a Humanoid*, Msc report 019ce2008, University of Twente.  
<http://essay.utwente.nl/58108/>
- Robokind Robots (2016), Zeno - R25.  
<http://www.robokindrobots.com/zeno-r25/>
- Schneider, W. and R. M. Shiffrin (1977), Controlled and automatic human information processing, *I. Detection, search, and attention. Psychological Review*, pp. 1–66.
- Shiffrin, R. M. and W. Schneider (1977), Controlled and automatic human information processing: II. Perceptual learning, automatic attending and a general theory, *Psychological Review*, pp. 127–190.
- University of Mons - NumediArt Institute - Attention Group (2015), Computational Attention - Saliency Modeling and Applications, Web.  
<http://tcts.fpms.ac.be/attention/>
- University of Twente (2015), Software prototype, FACE robot interfaces and 2D/3D representation using Temporal Disc Controllers for low- and high level behaviour controls, expressive Agents for Symbiotic Education and Learning (EASEL).
- Vandeveld, Cesar, e. a. (2014), Ono, a DIY open source platform for social robotics, *Proceedings of the 8th International Conference on Tangible, Embedded and Embodied Interaction*.
- Watanabe, T., A. Mader and E. Dertien (2013), Exploring Anti-Social Behavior as a Method to Understand Aspect of Social Behavior, *8th ACM/IEEE International Conference on Human-Robot Interaction (HRI)*.
- van Welbergen, H., D. Reidsma and S. Kopp (2012), An Incremental MultiModal Realizer for Behavior Co-Articulation and Coordination, *12th International Conference, IVA 2012, Santa Cruz*.