IMPLEMENTING AND OPTIMIZING A ROBOT AUDITION SYSTEM

Robotics Research Project Paper

Ben Wolf, s1879227, b.j.wolf@rug.nl,
University of Groningen, Department of Artificial Intelligence

Abstract: Robot Audition Systems are able to improve a robots internal representation of its environment. Key elements of audition systems are Sound Source Localization and Sound Source Separation. Using the principle of Blind Source Separation, robots prove to be able to localize and separate sound sources and recognize simultaneous sound signals. An implementation of a Blind Source Separation framework is presented, along with integration into the Borg architecture and optimization steps. Initial results show that command comprehension boosts from 57% to 93% in an order setting when comparing it with the old framework.

1. The Robot Audition System

Audition contributes to both perception and interaction in agents. Nakadai (2010) states that robots should have a hearing capability equivalent to ours to realize human-robot communication and social interaction, when they are expected to help us in a daily environment. Robots therefore should be able to recognize speech under various acoustic conditions. A mentioned possibility by Breazeal (2001) is to use headsets for direct communication, which solves various acoustics problem, but impedes the natural social interaction with robots.

From a perception perspective, the soundscape, or perceived sonic environment, provides agents with evidence for different sound sources and affordances. The latter can be described as objects or other agents that have some properties that enables the agent to perform an action or complete a goal. Although some affordances can be detected through vision or touch, detection of other types of affordances rely more heavily on audition, e.g. a hissing gas leak or distinguishing between aggressive and playful behaviors among humans (Andringa, 2013).

A proper Robot Audition System should be able to improve a robots internal representation of its environment and improve social interactions between robots and humans.

1.1. Challenges

Three main challenges for these systems are Sound Source Localization, Sound Source Separation and Automated Speech Recognition (Nakadai, 2010).

Sound Source Location (SSL) is usually tackled by using multiple microphones as a microphone array and the concept of binaural recording. The principle is that, with respect to an audition source, some microphones are blocked by the body or head of the robot. Therefore, some microphones will record a dampened and delayed version of the sound source. Using signal analysis and correlation between the microphones, the location of a source can be determined. This last step relies on the principles of Directions-of-Arrival (DOA) estimation (Rascon et al., 2014).

Sound Source Separation (SSS) is usually achieved through implementing a Blind Source Separation algorithm (Nakajima, 2008). This algorithm tries to separate the perceived signal into different source signals. Using the information of SLL, the search space for this algorithm can be limited severely, but this is still a hard problem with multiple target sources. The challenge is to extract a high quality source signal from the perceived mixed signals.

The system can be evaluated on SSS by using labeled data, but for a more reasonable quality measure of separation, the results from the Automated Speech Recognition (ASR) engine should be taken into account. A human interpretation of a 'good' separation does not guarantee a good separation for ASR. Furthermore, since the goal of this project is to optimize speech comprehension, the word recognition rate (wr) of the ASR is a sensible measure.

1.2. Source Separation

The base case for separating multiple sources with multiple channel microphone arrays is that the location of the sound source is known. In the basic form (Belouchrani et al., 1997), one has to process multidimensional observations given by:

$$\boldsymbol{x}(t) = \boldsymbol{y}(t) + \boldsymbol{n}(t) = \boldsymbol{A}\boldsymbol{s}(t) + \boldsymbol{n}(t) \qquad (1)$$

This model is widely used in the field of narrowband array processing. In eq. 1, $\boldsymbol{x}(t)$ is a noisy instantaneous linear mixture of source signals. In our case that would be the soundscape: the perceived sonic environment through the microphone array. The vector $\boldsymbol{s}(t)$ contains the signals emitted by the sound sources, these are the clean recognizable sound source events that a robot audition system should be able to infer from the original input. The $\boldsymbol{y}(t)$ vector contains the array output sampled at time t, and matrix $\boldsymbol{A}$ is the transfer function between sources and sensors. In order to construct the transfer function matrix, one should know the locations of the sources relative to each microphone node. But with robots, this is not always the case. Robots move and some sound sources, especially humans speaking, like to do so too. A Blind Source Algorithm is able to separate the original signals by estimating this transfer matrix without any prior information.

1.3. Blind Source Separation

In order to obtain the original signal, a separation matrix $\boldsymbol{W}$ needs to be created from the transfer matrix $\boldsymbol{A}$ such that $\boldsymbol{s}(t) = \boldsymbol{W}\boldsymbol{x}$ (Nakajima et al., 2008). The optimal separation matrix $\boldsymbol{W_{opt}}$ is estimated by minimizing a cost function $J(y)$ which denotes the mixture degree.

$$\boldsymbol{W_{opt}} = argmin\,[J(\boldsymbol{y})] = argmin\,[J(\boldsymbol{W}\boldsymbol{x})] \quad (2)$$

To obtain $\boldsymbol{W_{opt}}$, BSS updates $\boldsymbol{W}$ to minimize $J(\mathbf{y})$ by using a simple feedback loop (3), where $\boldsymbol{W_t}$ denotes $\boldsymbol{W}$ at the current time step t, $\mathbf{J'}(\boldsymbol{W})$ is defined as the update direction of $\boldsymbol{W}$, and μ is a step-size parameter. Most BSS algorithms use a fixed frequency-independent value as the step size parameter, as in eq. 3.

$$\boldsymbol{W}_{t+1} = \boldsymbol{W}_t - \mu J'(\boldsymbol{W}_t) \quad (3)$$

Six basic types of BSS algorithms are discerned that use this step function to separate the original signals; *DSS, ICA, HDSS, GSS, GICA* and *GHDSS.* Nakajima (2008) showed that by using an adaptive μ, performance in sound separation could be improved drastically.

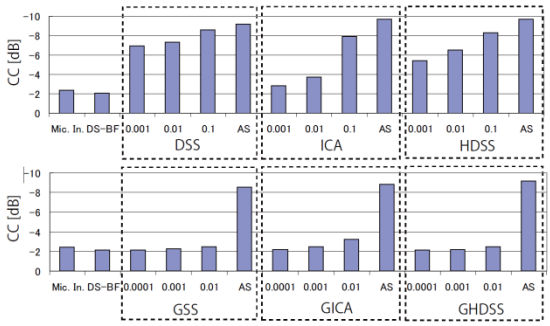

Figure 1: *The correlation coefficients of the sources and synthesized sources. In all algorithms, Adaptive Step (AS) helped improve separating the sound sources.*

These algorithms used to separate sound sources are formally defined by Nakajima (2008); *DSS* denotes Decorrelation-base Source Separation. *HDSS* is a similar node, but uses complex notation to obtain results. *ICA* is a simple independent component analysis based on Kullback-Liebler divergence and natural gradient method. The G-variations of the algorithms (Figure 1, bottom row) denote a Geometric approach to constrain search space. While usually underperforming the regular items, these algorithms are faster and don't suffer when using an Adaptive Step for the update parameter μ.

1.4. HARK

In recent papers, Nakadai (2010) shows that a robot with 8 microphones is able to understand 3 simultaneous speakers in one room. This is without any noise sources, except for the robot itself. The used architecture uses a modified version of the blind source separation to split the mixed perceived signal into source signals and is able to process these signals in parallel.

Hearbo (Nakamura, 2012) is a robot that is capable of ASR while listening to music. In their experiment the operator and sound speaker were positioned 60° apart. Hearbo dances to the playing music and can control the volume. The operator can request facts about the music, request a different song or a different dance. HARK is used as developed by Nakadai (2010) to separate the signals, along with additional modules for beat detection. Hearbo is already able to communicate with humans in a natural way, although Hearbo's lexicon is bounded by a finite grammar.

HARK utilizes the GHDSS algorithm, which calculates a high order correlation between the multiple inputs to separate them. Within HARK it is possible to add expected noise sources to the GHDSS configuration to enhance GHDSS's performance in terms of signal to noise ratio (Nakadai 2010). In essence, the robot is able to expect some noise sources from its own body while stationary or driving. They claim it should boost performance, but it is hard to estimate, since this noise profile also changes over time.

The GHDSS algorithms is fed through the MFMs that HARK creates. MFM is shorthand for *mel frequency cepstrum mask*. For each frequency band over time a confidence level is given as to its predictability for a clean speech source.

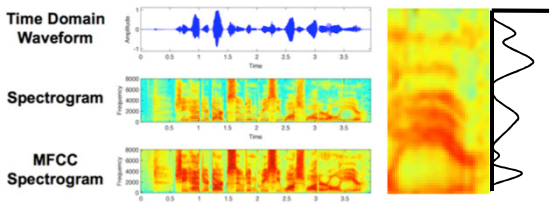

Figure 2: *A depiction of representations of sound and the principle of an MFM. The graph on the right denotes the MFM confidence density.*

This MFM gives a possibility for speech in the sound signal and is dynamically altered based on the output of the MUSIC algorithm (Schmidt, 1986). The MUSIC algorithm outputs the direction of a maximum $n - 1$ possible sound sources, where n is the number of microphones in an array. With these and other modules, HARK is able to track and recognize up to 3 simultaneous sound sources in a noisy environment from about 1 meter from the robot Nakadai (2010), if the sources are separated by 30°. Using an adaptive MFM in detecting words from a closed corpus, the word recognition rate increased from 48% to 95% compared to a fixed MFM. This enables the HEARBO robot to interact in a natural way with human and other sources of sound.

1.5. Conclusion

The impact for human robot interaction can be very significant. With just the option for dialogue with a robot, facing the operator when being talked to acts as a type of bodily feedback which the user will naturally interpret as if the robot is 'putting attention' to him/her (Rascon & Pineda, 2014).

Furthermore this approach to robot audition enables a robot to hear from virtually all directions. Most current domestic

service robots that are being developed right now, still use a directional microphone with a range of about 50 cm. Therefore whenever an instructor want to talk to the robot, he/she should stand in front of the robot and bend towards the microphone, which isn't really a natural way of communicating with the robot. Ideally a robot owner should be able to give the order *come here* with the robot not seeing its owner, but detecting the angle of the speech source and using that information to make an initial navigation plan to move towards its owner.

Research in Robot Audition has primarily been focused on Automated Speech Recognition. Robots up to this point use not all auditive information that is available to humans. The fact that a robot which is able to separate multiple speech sources is exciting. It also paves the way for a system that can use sources' auditive information to enhance the internal representation of the world.

A (for instance domestic or hospital) robot could therefore be more aware of the general mood of an environment and use this context to select tailored behaviors, more suited to the current situation. i.e. a cry for help in a silent home or in a crowded room could require a different approach to conflict resolution.

2. Implementation of the HARK Framework

In order to incorporate the HARK framework into the existing BORG architecture, the audio processing pipeline must alter. In the original setting, the audio signal from a directionally sensitive microphone was sent through a Java compiled CMU Sphinx system. Speech hypotheses were sent through TCP to a speech accepting client and were put into memory.

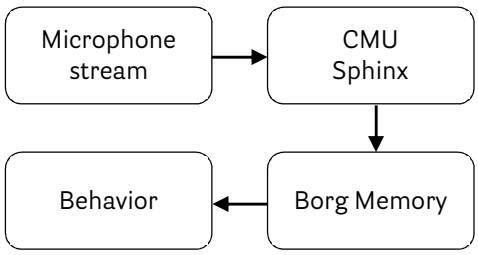

Figure 3: *The low-detail flow diagram of the previous sound system used in the BORG architecture*

The first goal is to replace the microphone stream with a filtered stream, but HARK doesn't produce a continuous stream, only filtered sources which could overlap in time. Furthermore, the behavior has little control on how the ASR functions, i.e. changing the finite state accepting grammar or communicating that the robot is producing a non-focus sound source itself; robot speech synthesis.

2.1. Designing the HARK framework

Through the HARK cookbook, extensive documentation and examples, a network flow file can be made which uses the element of the first section to stream separate sources using the HARK node HarkDataStreamSender. A rendition using the `hark_designer` as a visualization tool for the HARK network is depicted in Figure 4.

The network starts out with a four-channel audio stream from the Kinect (360) microphone array. After a channel selective gain, a Fourier decomposition is conducted to produce frequency bands for the GHDSS algorithm. Continuing downwards, the MUSIC algorithm produces source steering vectors which are compared with previous known sources at the SourceTracker module. When a source steering vector is similar to an earlier vector, the same Source ID will be assigned. If the source vector difference exceeds either the threshold for time or location, a new Source ID will be assigned. The Extender keeps the source 'alive', making sure to listen a while longer in that direction to prevent loss of audio due to small pauses in speech.

After the source separation is complete, white noise can be added (currently set to 0%) and clean sources are synthesized and send through TCP to a recognizer of choice. The default is Julius, with high scores on Japanese speech, but sub-par results for English sentences (Gaida, 2014). In our case, the data stream is connected to PocketSphinx.

The Sphinx system works with utterances in a continuous input signal. A speech signal estimator checks the incoming microphone stream for speech probability. Some features that are used here are tonality and pulsality (Andringa, 2002) which are good predictors of speech signals. For Sphinx, a speech source begins and ends based on this estimator. Whenever a speech utterance ends, Sphinx produces a hypothesis. In the older implementation, this is sent to a memory client to put in memory and for the behavior to use.

The major compatibility problem here is that Hark doesn't allow for continuous output, which is a result from this kind of location filtering. The final node in the Hark network outputs packets of data, SRC_INFO and SRC_ID. The latter contains the ID handed out by the Source Tracker. Since sources can overlap in the time dimension, the output stream sometimes contains no source (in silence or noise); sometimes contains one source (with n sources that don't produce sound at the same time); and sometimes n sources. These partial streams need to be concatenated for each source ID before Sphinx would be able to decode the separated sound signals.

The currently implemented minimal working example (with all modules in place) accomplishes this feature by having n buffers, where n equals the amount of simultaneous perceivable sources, in our case 3.

With an open TCP connection to the HarkData-StreamSender, this script waits for additional data from hark. Whenever a new ID is received, the script assigns an empty buffer to this ID and keeps concatenating incoming data if the ID of new data is matched. If an assigned buffer doesn't obtain new data for a few cycles, the buffer is closed and ready for processing by Sphinx.

At this point, the old java Sphinx4.jar file was replaced by the newest version of a mobile implementation of Sphinx: pocketSphinx. This version allows for a python implementation using Swig, instead of using yet another TCP

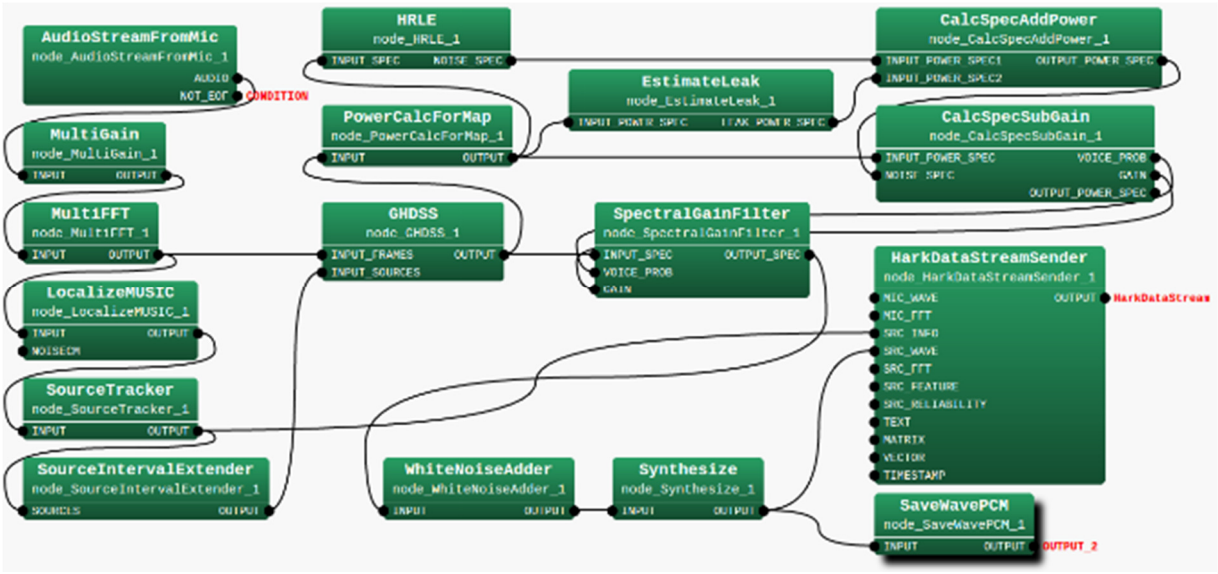

Figure 4: *The HARK network currently in use for sound source separation and parsing*

connection to connect to the Sphinx jar and produce utterances in memory the old way.

2.2. Decoding

With the implementation of PocketSphinx, a decoder is initialized once with a full language model. Optionally we can restrict the search space for utterances by using a JSGF or java scripted grammar file. This grammar notation allows for an extended finite state grammar (FSG) by having optional and recursive definitions.

When a buffer has been closed in the previously mentioned script, we manually start an utterance in the Sphinx Decoder. We trust that Hark has separated the source correctly – i.e. it contains a whole utterance – and request from Sphinx that this source is decoded. Since the buffer contains sound information from one source, and the next source could be a different speaker or command, the utterance is closed after processing this buffer.

Details on the operating of Sphinx are outside the scope of this report. However, the key part of Sphinx's operation is that the list of hypotheses of one decoded utterance contains duplicates and parts of sentences governed by the grammar. The beam search algorithm at the core of Sphinx terminates when an optimal route through the FSG is found and lists all the paths – in our case hypotheses – still in consideration.

2.3. Closing the loop

To recreate the original diagrams functionality, the best two hypotheses (according to a nonlinear internal scoring mechanism) is returned to the script and put into Borg memory directly by treating this script as a ROS node.

3. Basic Behavior Integration

In order to improve the system in terms of Natural Human Machine Interaction, one would ideally want to control the speech recognition system from the behavior, i.e. communicating that the robot is speaking or that it is expecting an answer to a question. Another very useful feature is to change the closed grammar that the system is using on the fly; different scenarios require different expected utterances and therefore different FSGs.

3.1. Speech Recognizer features

The features described here are implemented in the SpeechRos python script.

Pausing

The Speechrecognizer thread has a public Boolean – Pause – which is set to True through the SpeechRos class. This class checks the memory and RosTopics for speech. Whenever the robot speaks, the Speech Recognizer can pause the recognizer, nulling all incoming sound data.

Set_responses

From a behavior, a grammar file could be set including all the utterances that the robot should understand. Sometimes though, the behavior requires the robot to listen for something he hasn't heard before. The set_responses routine makes a temporary grammar file and replaces the present grammar for one detection. A common situation is to listen for expected answers to a posed question.

Ask

Implemented in the speechcontroller a function Ask completes a set of tasks that are useful for a behavior. Ask takes two arguments, the question as a string and the set of responses as a list of strings. The interplay between the behavior, speechcontroller, and SpeechRecognizer can be shown through an example scenario; a restaurant task.

3.2. Speech Recognizer Example

The robot starts out with a general grammar, accepting commands, orders and the emergency stop. At first the robot could ask a question to the operator using self.body. say("what

would you like to order?")

>>>What would you like to order?

We now expect the human operator to say something that can be parsed through the general grammar of the situation, namely one of the selected dished on the menu.

self.m.get_last_observation("voice_command")

I would like to have a pizza hawai<<<

Now this speech observation has several fields containing useful information for the behavior. The observation has a message, *time*, azimuth, 2best. The first is the first hypothesis from the decoder, the azimuth is the angle of the source relative to the Kinect and the last field – 2best – contains the best two hypotheses.

If for any reason the two best hypotheses differ, e.g: "I would like to have a pizza hawai", or " I would like a pizza funghi"; then the behavior should ask the kind of pizza that the operator really wants. The robot can make a guess and try to confirm the first hypothesis.

self.speech.ask("Do you want the pizza hawai?",\
["yes [please]", "no"])

>>>Do you want the pizza hawai?

While the robot is speaking, the speechcontroller pauses the recognizer (so it doesn't listen to itself) and reconfigures the grammar to only accept the expected answers. When the robot is done speaking, the bodycontroller sets the Boolean Pause back to False and allows the Speech Recognizer to listen for the selected responses.

... yes please<<<

The grammar should change only for one utterance and change back to the original grammar after an answer has been decoded. This is solved by another Boolean – asking – and a temporary storage place for the grammar. The decoded utterance is send to memory which allows the behavior to formulate a response:

self.body.say("Okay, I will get the Hawai")

>>>Okay, I will get the Hawai

There are some intermediate steps, checks and conversion, but this covers the pipeline so far. In this example, the behavior now correctly guesses that the first hypothesis was the correct order, but would've been stuck if that wasn't the case. An often used workaround is to ask the operator to repeat the whole order, but that seriously breaks the emergence of communicating with a relatively smart device.

To fix this issue, a crude Natural Language Processing module has been set up which takes a highly abstract world model into account. Welcome additions are an ontology which allows for hypothesis adjusting. This allows for a distinction between a grammatically correct sentence and a likely one.

3.3. Part of speech tagging

Back to the topic at hand, what *is* feasible with some generic word knowledge, is to compare Sphinx hypotheses with each other and extract confidence information about parts of sentences. This allows us to produce meaningful questions to resolve any leftover uncertainty about the decoded utterance.

If we look back at what the recognizer should be able to deal with, namely a closed set of utterances, there's no ambiguity in the commands that a domestic service robot should deal with during the test setting. In essence, we *know* the verbs that the robot should understand, we *know* which objects can replaced, we *know* what orders to expect. These are all defined beforehand. That's why speech utterances are not parsed with an automatic part-of-speech tagger, but rather a rule base system specific for the closed grammar set is proposed and used in the behavior specific Natural Language Processing File.

4. Advanced Behavior Integration: Parsing, Producing Responses & NLP

The final part of the behavior integration is mostly designed to accommodate the General Purpose Service Robot task and Speech tasks in the RoboCup@Home setting. The latter comprises of answering questions and producing meaningful responses, while the first is to allow the robot accomplish a General task through speech. Several files and scripts work together to accomplish this feature, the grammar, jsgfparser, nlp and finally the behavior. In this section the company demo behavior is treated as an example.

4.1. Grammar file

The grammar file denotes useful items for the Sound Recognition algorithm, but also contributes (partly) to the crude part-of-speech tagging that will be used by the nlp.py script. These are the contents of the grammar file:

```
#JSGF v1.0;
grammar vcc;
public <all> = <start> [please] <order> (and
<order>)*;
<start> = (alice);
<order> = <get> | <second>;
<get> = get [the] (cup |can )from <location>;
<second> = (empty [it] [in] the basket on <location>
| bring [it] to <location>);

<verb> = (get | bring | empty);
<object> = [the] (cup | can | basket);
<location> = (table one | table two | table three |
operator | the operator );
```

It starts off with a public <all> that lists all accepted speech utterances. 'alice' is the required start word and is followed at least one order. There are two types of orders, a <get> and a <second>, the first notes getting an object, the latter is doing something with the object.

The top part defines the grammar structure for the Speech Recognizer, the last three lines provide information about tags that we're interested in for parsing. If we are uncertain about which member of a class we heard, we can compile a question for that class, e.g. "What *object* should I get from table one?

4.2. JSGFParser class

This class loads a grammar file and can parse and search the grammar in various ways. Orders with optional words [denoted by brackets] are reduced to a version without Optionals in the function filterOptionals. In this way, "Alice

please get ...” and “Alice get ...” can be parsed to be equal. Furthermore we can look up for any word which class it belongs to, for any class the list of possible words and check for choices, let’s say (option 1 | option 2) which is useful for sentence generation.

4.3. nlp class

This class separates the speech processing from the old GPSR behavior which had all the processing happening in one file. The goal of this class is to convert a speech observations into an action structure and check whether some things need to be asked before confirming and executing the command. This broad requirement makes the responsible class a bit open and not well defined.

The nlp class uses the JSGFParser class to traverse and identify word types like *verb* and *object*. It has access to the variable items of importance. These can be different for each task. For example the GPSR has 13 variable classes that need to be parsed differently.

Parsing the observation

As a short reminder, a speech observation holds the two best hypotheses in the form of strings. These two strings need to be compared. The process is again illustrated with an example:

```
obs[‘2best’] ::
["alice get cup from table one and bring it to table
two", "alice get the cup from table one and bring to
table three"]
nlp.get_actions(obs) #gets options
```

The philosophy of the get_actions function is to first remove the optional words, then cut the long command up into shorter elements, then per element the two best hypotheses are compared. The order is first split on “and”. The – in this case – two parts with each two hypotheses are partially parsed. This function is specific to a grammar and makes a dictionary of the two hypotheses, reducing them further in dimensionality. Through regular expressions created based on the variable items of importance, the partial order is converted into lists of action dictionaries. The first part of the order will result in this python structure:

```
parsed_list ::
[{"verb":“get","item":“cup","location":“table one"},\
{"verb":“get","item":“cup","location":“table one"}]
```

The NLP class checks for differences and formulates a question to resolve the difference at hand using the known information. In this case the resulting question compilation result is easy and predictable:

```
{"diff" : False ,\
"question" : "No difference found", \
"understood_options": None,\
"action" : parsed_list[1],\
“unknown": None}
```

This action structure is added to the yet empty list of action structures and the second order is checked for differences. The returned action structure then looks something like this for the second part of the order:

```
{"diff" : True ,\
"question" : "Where should I bring the item to? table
two or table three",\
“understood_options": [“table one", “"table two"],\
"action" : {"verb" : “bring", "item" : “item",
"location" : None,\
“unknown": “location"}
```

The action structure lists the understood options, which consequently are the expected answers to the formulated question. The parsed list currently holds None as its location, which should be resolved, indicated by the dictionary item “unknown”. When `nlp.get_actions(obs)` is called from the behavior, NLP returns a list of these two action structures.

Confirming the order

An action structure can only be confirmed if there are no unknowns left. With this confirmation (after a possible question dialogue) the robot can show the operator that the command has been understood correctly and explain what it is about to do. This confirmation is built based on the action structures and is useful when resolving an unknown important variable. If previous parts of the command can be confirmed early on, it is more clear to the operator which parts are unclear for the robot. Which brings us to the most important file, the behavior.

4.4. Behavior

For action selection like behaviors, like the Borg company demo and the GPSR challenge, a standard profile of behavior is used. The behavior has several states which are consistent throughout the variants and reused, the other states refer to the actions or subbehavior selected. The behavior normally kickstarts with a speech command while in the “waiting for command” state. If a new speech observation is found in memory, the nlp class is used to convert the observation into action structures, the state is changed to “processing actions”.

It pops the first actionstructure (AS) from the list and checks whether there’s a difference with the diff Boolean. And will confirm the first order:

```
>>> “Okay, I will get the cup from table one”
```

The state hasn’t changed, allowing the second AS to be dealt with. The behavior uses self.speech.ask to ask:

```
>>> “but, where should I bring the item to? table two
or table three?”
```

The state has now changed to “waiting for answer”. If there’s a new speech observation the behavior first checks whether the first hypothesis variable type matches the previously unknown type, in this case location. If these types don’t match, the behavior repeats the question. If the types do match, for instance the operator answers with “table two”, which is value of the unknown variable, the previous AS gets updated with this new information and the AS can be confirmed.

```
>>> “Okay, I will bring the it to table two”
```

Now all AS have been confirmed, a list of states is generated to be traversed. This statelist is the central control of the behavior. The behavior checks the main verb of the action structure and decides what subbehaviors should be ran with

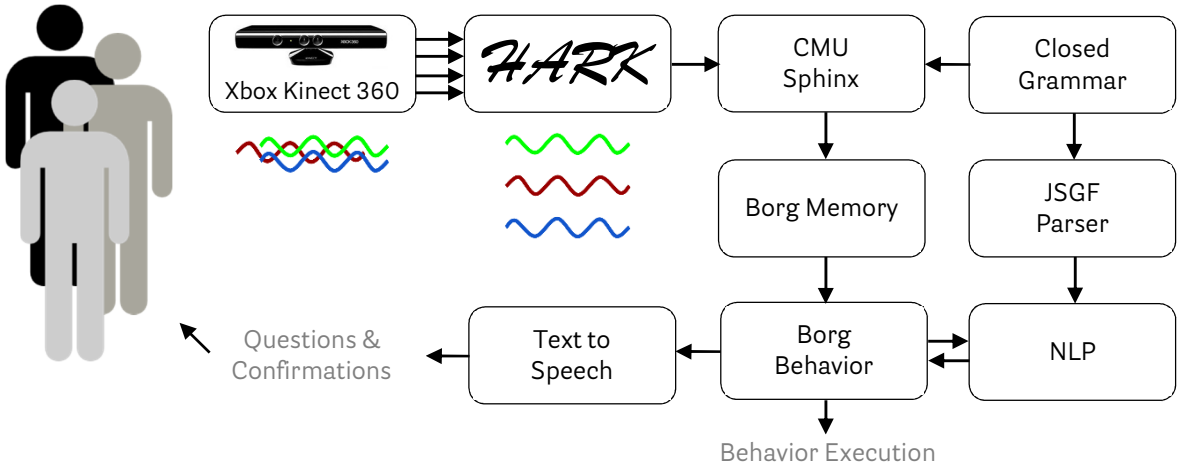

Figure 5: *The sound processing framework with its major components and relations to the behavior architecture and the Natural Language Processing module. Sound sources are separated with HARK, then decoded into hypotheses by PocketSphinx and put into Borg memory. During an interplay between the Behavior and the NLP class, the command is parsed into steps of actions to be taken. Any unknowns in one of the steps is resolved through questions, clear commands are confirmed through text to speech.*

which parameters and puts these on the list. In case of our example, we will get two action states in the statelist:

```
self.statelist ::
[["get_x_from_y",current_AS],\
["place_on_y",current_AS]]
```

After the statelist is populated, the user at hand can decide whether the AS should be printed, or whether the behavior should finally start with its first order. In any case, the state is switched to `self.statelist[0][0]`.

Action Execution

This marks the end of the human interaction part, since the robot will now execute its orders. In order to present a complete picture, the rest of the behavior is explained here as well. The state that is set in the last section is "get_x_from_y". Within the demo behavior, the subbehavior `grabitem fromlocation` is called with two parameters, the item and location. This behavior gets added to the `self.selected_behaviors` list with the precondition `True`. The behavior remains in this state and waits until the `behavior.is_finished()` and empties the selected behaviors, and retrieves the next state.

5. Optimizing ASR with environment evaluation

There are a lot of parameters in the Hark system which influence the quality of source detection and separation. Ultimately better separated signals have a higher chance to be recognized correctly. This will be shown in this section. The most influential parameters for sound source separation are those linked to Sound Pressure Levels (SPL), often measured in dB. Even for humans it is harder (but still feasible) to understand a person standing a meter away in a crowded environment than for instance the same person a meter away in an office room.

In the next subsections an experiment is described to ascertain the importance of several influential Hark parameters and how they influence speech comprehension in a two-step procedure.

5.1. Methods

To estimate the performance of the proposed framework, a quality measure is needed. It is very hard to come up with a measure for sound separation, unless one is willing to tag the sentences in the time domain as well. The most interesting quality measure of the framework overall is the recognition score.

Labels

The utterances which are used for this experiment are generated from a special grammar file which resembles the GPSR situation. The grammar is more complex compared to the examples shown earlier. The robot's name can either be Sudo or Alice, and three types of utterances can be discerned. Furthermore commands can be concatenated indefinitely, but for this experiment they are limited to three.

```
#JSGF v1.0;
grammar vcc;
public <all> = <start> (<question> | <request> |
<order>);
<start> = (alice|pseudo);

<question> = (what is your teams name | what is your
favorite color);

<request> = [please] (follow me | enter the elevator
| give me a drink | pick up the [<color>] cup);
<color> = (red | white | green | purple | blue |
yellow);

<order> = [please] <verb> [the] <object> (and [the]
<object>)* to [the] <location> (and [<verb>] [the]
<object> (and [the] <object>)* to [the] <location>)*;

<verb> = put | place | bring | take | deliver | get;
<object> = (orange juice | ice tea | coffee | beer |
chocolates | chewing gums | peanuts);
<location> = table one | table two | table three |
object shelve one | object shelve two | kitchen |
bedroom;
```

From this, 10 sets are generated each consisting of 10 utterances, which yields 100 spoken utterances. Sets of 10 are chosen to allow the operator to make a mistake and possibly

Figure 6: a. *Setup for recording foreground speech signals in the robotics lab.* ***b.*** *Setting of an ambience sound recording during a break of a lecture in the main lecture hall of the Bernoulliborg.* ***c.*** *Measurement setup during the recording of* ***b****.*

redo 9 utterances of recording, when the pronunciation is sub-par instead of the whole dataset.

Recordings

At first, clean recordings are made in a silent environment, in this case the robotics lab. The utterances are read one by one and voiced when looking into the kinect microphone array. This ensures that the voice recording is clear. See also Figure 6a.

The recognition scores through the framework on this set of noise free recordings will act as the upper baseline for the performance when background sounds are added. Background recordings are made in the same constellation, a two minute long recording of silent operator standing in front of the Kinect to make sure that no other (speech) sources originate from the same position.

Several locations have been selected that lie within the natural habitat of our domestic service robot, Alice. The ambience during a break of a 300 attendees lecture was recorded, for instance (Figure 6b,c); as well as the cafeteria during lunch break, the local study associations member area and several classrooms. Needless to say, some ambiences had a higher SPL than others, a dependent variable in the experiment.

All 19 background signals are combined with the ten foreground signals to form a set of 200 recordings, 190 mixes and the 10 noise-free recordings. Using an adapted version of the SpeechRos speech recognizer, these mixes are decoded and the recognition results are compared with the labels. This comparison checks whether the understood command, so after NLP generalization, matches the utterance denoted in the label.

Adapting network

Three settings are compared in this experiment, with three different kind of networks. In the first case, the classifier has as input mono-converted input signals and cannot use hark to separate the sound sources, but rather relies on Shpinx function to detect the speech utterances and send its hypotheses to NLP.

In the second case, HARK parameters are estimated without an dependent variable, so kept constant regardless of the actual SPL. The selected parameters `TRACK_THRESH`, `SEPARATE_GAIN`, `TRACK_PAUSE` and `MULTI_GAIN` are found through a grid search on values close to the default values.

Since these parameters mostly deal with SPLs, the idea is to make a linear regression model that makes these parameters dependent on the average SPL of the set. Again, the following results are obtained through a grid-search process.

5.2. Results

Figure 7 shows the results of the most influential parameter sweep. When there is little to no background noise, the word recognition rate is not radically influenced by the parameter. In more noisy environments, a higher threshold can drop the recognition rate to 0.12 in the worst case.

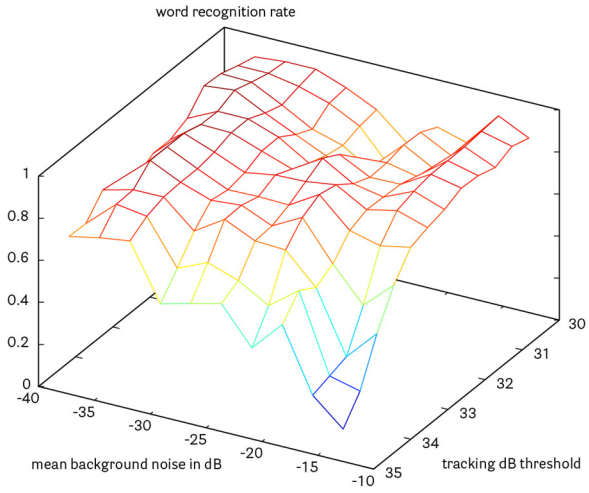

Figure 7: *Word recognition rate for background noise level and tracking threshold. The calculated mean SPL of the background recordings is used as a measure of background noise. The tracking threshold varies from 30 to 35 with a step size of 0.5. In this case the pause parameter was set to 800 frames ~ 1.5 seconds and no gains.*

Another sweep (Figure 8) shows the performance of the system based on the `track_pause` parameter. It shows that de default parameter (800 frames) is sufficient, but the optimum denoted by the blue lines is SPL dependent.

If a constant parameter for tracking should be chosen, 32.5 seems the threshold with the highest word recognition rate, based on this sweep. The dependent parameter value should decrease if the background noise increases for an optimal

word recognition rate in this setting. The optimal parameters for each of the cases and their word recognition rate is presented in Table 1.

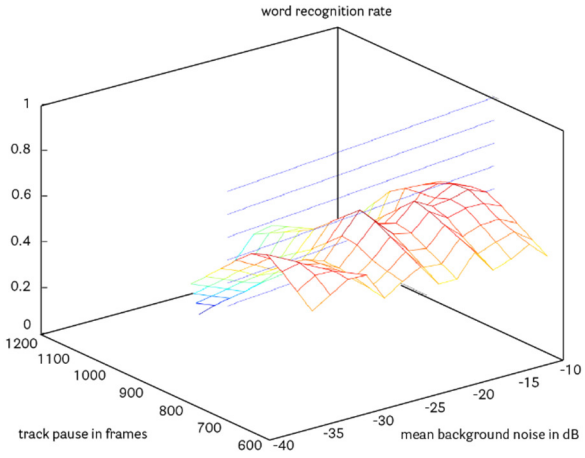

Figure 8: *Word recognition rate for background noise level and tracking pause parameter. The calculated mean SPL of the background recordings is used as a measure of background noise. The tracking pause varies from 600 to 1200 with a step size of 50.*

	TRACK THRESH	SEP GAIN	TRACK PAUSE	MULTI GAIN	Words rate	Order rate
1	n/a	n/a	n/a	n/a	.5434	.5752
2	30	1.0	800	1.3	.8466	.9124
3	$30 - \frac{SPL}{11}$	1.1	$900 - \frac{SPL}{3}$	1.1	.8226	.9393

Table 1: *Word recognition rate and order recognition rate for the three methods of processing. 1. Left-most microphone stream fed into Pocket-Sphinx. 2. Static parameters for HARK separation. 3. SPL dependent parameters for HARK separation.*

6. Discussion

The robot audition system HARK has proven to be a useful tool to make use of microphone array based sound separation and its potential has not been met yet. Tweaking the MUSIC algorithm or trying to optimize the Adaptive Step parameter settings could yield better word recognition rates or – just as important – better estimates on certainty of the ASR hypotheses. Other developments are native ROS and Python support of several HARK modules, allowing for a potentially cleaner and more reliable pipeline.

Switching from Sphinx to PocketSphinx in the Borg architecture allowed for a Python SWIG implementation fully compatible with the BORG memory and behavior system. This enables real-time native communication between a behavior and ASR, having dynamically inter-changeable grammars and more input about what should be done about and with utterances from operators.

The separation in the top-level behaviors between language processing and action selection into separate dedicated classes allows for easy reuse of generic functions. This greatly reduces the lines of code inside a behavior and imposes a structure of top-level behaviors, speeding up behavior development and promotes re-using tested sublevel behaviors.

Furthermore, questions and answers can be structurally formulated by the NLP class and answered by the operator though the use of the adaptive grammar on the ASR. This allows for a more natural Human Machine Interaction.

Against expectations, a dynamic separation system based on the SPL of the soundscape, was outperformed by static separation in terms of word recognition rate. The order comprehension however, is slightly better using dynamic SSS.

One could argue that improvements above 90% rate are hard to achieve, regardless of the methods used. This might not improve the robots ability to act on speech commands, but having a better confidence level of decoded utterances will prevent the robot to act on falsely assumed correct understood orders.

Potential improvements could be obtained through dynamic alteration of HARKs parameters on the current soundscape. Up to this point, only the average SPL is taken into account since HARK has only recently been reconfigurable through its ROS implementation. Then again, case 2 (Table 1) shows that fixed parameters allow the Robot Audition System to perform with comparable order comprehension rates.

7. Conclusion

The goal of the project was to enhance the speech system with a microphone array using soundscape information and natural language processing; evaluating every step using a meaningful measure.

The microphone array's abilities to separate sound sources from backgrounds greatly improved the speech recognition and the technique is essential for understanding commands in noisy environments

Large steps have been made to improve the human machine interaction and the structure of sound processing within the robot. Dynamic configuration of ASR enables spawning a natural dialogue with the operator.

Parsing and extracting information from multiple hypotheses boosts the order comprehension and proves a useful method to evaluate robot audition systems.

8. References

Andringa, T., & Lanser, J. (2013). How Pleasant Sounds Promote and Annoying Sounds Impede Health: A Cognitive Approach. *International Journal of Environmental Research and Public Health*, *10*(4), 1439–1461.

Andringa, T. (2002). Continuity Preserving Signal Processing. *PhD thesis,* University of Groningen

Belouchrani, A., Abed-Meraim, K., Cardoso, J.-F., & Moulines, E. (1997). A blind source separation technique using second-order statistics. *Signal Processing, IEEE Transactions On*, *45*(2), 434–444.

Breazeal, C. (2010). Emotive qualities in robot speech, in: *Proc. IEEE/RSJ Int. Conf. on Intelligent Robots and Systems*, Maui, HI, pp. 1389–1394

Gaida, C., Lange, P., Petrick, R., Proba, P., Malatawy, A., & Suendermann-Oeft, D. (2014). *Comparing Open-Source*

Speech Recognition Toolkits. Technical report, Project OASIS, DHBW Stuttgart, Stuttgart, Germany.

Lavate, T. B., Kokate, V. K., & Sapkal, A. M. (2010). Performance analysis of MUSIC and ESPRIT DOA estimation algorithms for adaptive array smart antenna in mobile communication. In *Computer and Network Technology (ICCNT), 2010 Second International Conference on* (pp. 308–311). IEEE.

Nakadai, K., Takahashi, T., Okuno, H. G., Nakajima, H., Hasegawa, Y., & Tsujino, H. (2010). Design and Implementation of Robot Audition System "HARK" — Open Source Software for Listening to Three Simultaneous Speakers. *Advanced Robotics*, *24*(5-6), 739–761.

Nakajima, H., Nakadai, K., Hasegawa, Y., & Tsujino, H. (2008) Adaptive step-size parameter control for real-world blind source separation, in: *Proc. IEEE Int. Conf. on Acoustics, Speech and Signal Processing*, Las Vegas, NV, pp. 149–152

Nakamura, K., Nakadai, K., & Ince, G. (2012). Real-time super-resolution sound source localization for robots. In *Intelligent Robots and Systems (IROS), 2012 IEEE/RSJ International Conference on* (pp. 694–699). IEEE.

Rascon, C., & Pineda, L. (2014). Multiple Direction-of-Arrival Estimation for a Mobile Robotic Platform with Small Hardware Setup. In *IAENG Transactions on Engineering Technologies* (pp. 209–223). Springer.

Rascón, C., Avilés, H., & Pineda, L. A. (2010). Robotic orientation towards speaker for human-robot interaction. In *Advances in Artificial Intelligence–IBERAMIA 2010* (pp. 10–19). Springer.