Unified Incremental Multimodal Interface for Human-Robot Interaction

Master Thesis, Computer Science
Advanced Level, 30 Credits

Author:
Afshin Ameri E.

Supervisor:
Baran Çürükülü

November 17, 2011
Thanks to Baran for his patience, wisdom and help.

Thanks to Batu for his support during the project.

I would like to dedicate this work my awesome mother and brother
Abstract

Face-to-face human communication is a multimodal and incremental process. Humans employ different information channels (modalities) for their communication. Since some of these modalities are more error-prone to specific type of data, a multimodal communication can benefit from strengths of each modality and therefore reduce ambiguities during the interaction. Such interfaces can be applied to intelligent robots who operate in close relation with humans. With this approach, robots can communicate with their human colleagues in the same way they communicate with each other, thus leading to an easier and more robust human-robot interaction (HRI).

In this work we suggest a new method for implementing multimodal interfaces in HRI domain and present the method employed on an industrial robot. We show that operating the system is made easier by using this interface.
Sammanfattning

Ansikte mot ansikte mänsklig kommunikation är en multimodal och inkrementell process. Människor använder olika informationskanaler (villkor) för sin kommunikation. Eftersom vissa av dessa metoder är mer felbenägen till viss typ av data, kan en multimodal kommunikation dra nytta av styrkan i varje modalitet och därmed minska oklarheter under interaktion. Sådana gränssnitt kan användas för intelligenta robotar som arbetar i nära relation med människor. Med denna metod kan robotar kommunicera med sina mänskliga kolleger på samma sätt som de kommunicerar med varandra, vilket leder till en enklare och mer rubost mänskliga-robot interaktion (HRI).

I detta arbete föreslår vi en ny metod för genomförande av multimodala gränssnitt i HRI domän och presentera den metod som används på en industrirobot. Vi visar att driva systemet underlättas med hjälp av detta gränssnitt.
Preface

It has taken me a long way since I decided to change the path of my life from being a dentist to become a researcher in the field of Computer Science. I have not achieved that goal yet, but this master thesis is a part of the path towards it. During the way there have been many wonderful people who supported me and here I would like to thank each of them for their support, kindness and help.

First, I should express my gratitude towards the angel which is my mother, Iran Nekuamal. Without her kindness, help and unlimited support, I could not make this dream come true. I should also send millions of thanks to my awesome brother, Amin Ameri, the coolest person I know. His wisdom and support have been the light on this path.

I would also like to thank Baran Çürükli, my supervisor, advisor and now a friend. Baran! I would like you to know how grateful I am for all your help and support. I wish I could pay them back one day.

Next thanks goes to Batu Akan, whom without his advices and help during the thesis project, this could not have been achieved.

Last, but not least, I would like to mention all my friends who stood by my side all the way. Mahgol, Shahab, Melika, Björn, Tina, Elham, Thomas, Hubert and Mana, thank you.

November 17, 2011
Afshin Ameri E.
Contents

1 Introduction ....................................................... 1
  1.1 Background .................................................. 3
    1.1.1 Multimodal Frameworks ................................. 3
    1.1.2 Multimodal Systems ................................... 4
    1.1.3 Incremental Parser .................................... 5
  1.2 Published Papers ............................................ 5

2 Architecture ..................................................... 7
  2.1 COLD Language .............................................. 10
    2.1.1 COLD Syntax .......................................... 10
    2.1.2 COLD Parser and Compiler ............................. 11
  2.2 Incremental Multimodal Parsing .............................. 12
    2.2.1 Multimodal Grammar Graph ............................. 12
    2.2.2 Hypothesis ............................................ 13
    2.2.3 Incremental Parser .................................... 13
  2.3 Modality Fusion ............................................. 14
  2.4 Semantic Analysis .......................................... 15

3 Results .......................................................... 19
  3.1 Multimodal Interface Test Application .................... 21
  3.2 Block Stack Up ............................................. 22
  3.3 Kitting ..................................................... 22
  3.4 Block Sort .................................................. 23

4 Discussion and Future Work .................................... 25
  4.1 Discussion .................................................. 25
    4.1.1 COLD Framework ...................................... 25
    4.1.2 Multimodal HRI through Augmented Reality ......... 26
  4.2 Future Work ................................................. 27
Chapter 1

Introduction

Companies producing mass market products such as car industries have been using industrial robots for machine tending, joining, and welding metal sheets for several decades. However, in small medium enterprises (SMEs) robots are not commonly found. Even though the hardware cost of industrial robots has decreased, the integration and programming costs make them unfavorable for many SMEs. In order to make industrial robots more common within the SME sector, industrial robots should easily be (re)programmable by any employee of the manufacturing plant. Our goal is to give a robot the ability to communicate with its human colleagues in the way that humans communicate with each other, therefore making the programming of industrial robots more intuitive and easy.

One of the main characteristics of human communication is its multimodality. Speech, facial gestures, body gestures, images, etc. are different information channels that humans use in their everyday interaction; and most of the times they use more than one at the same time. At the other hand, humans see robots as objects with human-like qualities [36]. Consequently, a human-like interaction interface for robots will lead to a richer communication between humans and robots.

In-person communication between humans is a multimodal and incremental process [8]. Multimodality is believed to produce more reliable semantic meanings out of error-prone input modes, since the inputs contain complementary information which can be used to remove vague data [35]. For example, if the speech recognition system has “blue object” and “brown object” as its two best hypotheses, the wrong hypothesis can be easily ruled out if there is support from vision system that there is no blue object in the scene.

It is also accepted that some means of communication are more error-prone to special type of information and can transfer other data types more precisely [35]. In a multimodal communication, humans can use the modality which is more reliable for the information to be transferred. For example when someone asks about directions to a special place,
a good way of informing them is using a map (visual channel) rather than explaining it with words. As another example, in an industrial environment, saying “weld this to that” while pointing at the two objects, is more reliable than saying “weld the 3cm-wide 5cm-long piece to the cylinder which is close to the red cube”. That’s because the speech channel is more error-prone when it comes to defining spatial information, while visual channel is more reliable in this case.

Human brain processes different modality inputs incrementally. This means that the processing starts as soon as the inputs start and the semantic meaning of the inputs is build up in the brain incrementally [26]. This also applies to resolving perceivable context and action planning. In other words, people understand and plan their responses incrementally and they perform the action when the complete meaning of the utterances is perceived [26, 8]. This action maybe in the form of a head nod, vocal response or something that the speaker had asked them to do.

In HCI/HRI domain, incremental processing helps to improve response times of computer systems. This is specially beneficial for audio inputs, since the computer can use the speech time for performing most of its calculations, thus resulting a much more improved response time [34, 38]. A fast response time is an important feature for designing a multimodal system. Such systems should be very responsive, because (if not) they may lead to repetition of commands from the user, more ambiguity in the recognition process and user annoyance [13].

Apart from helping in addressing ambiguities in error-prone inputs, multimodal interfaces have other benefits including: interface robustness, reliability, error recovery, alternate communication methods on different situations and more communication bandwidth [19]. These qualities make multimodal interfaces a good choice for most HCI/HRI applications.

In this work we have implemented a framework for rapid development of multimodal interfaces and used it in an industrial scenario. The framework is designed to perform modality fusion and semantic analysis through an incremental pipeline. The incremental pipeline allows for semantic analysis and meaning generation as the inputs are being received. The framework also employs a new grammar definition language which is called COactive Language Definition (COLD). COLD is responsible for multimodal grammar definition and semantic analysis representation. This makes it easy for multimodal application developers to view and edit all the required resources for representation and analysis of multimodal inputs at one place and through one language. We used this framework to develop a high-level augmented reality interface for programming (read it training) of an ABB IRB140 industrial robot.
1.1 Background

Because the presented system is an incremental multimodal framework, in this section we will briefly review previous studies in all the involved fields. They include: multimodal frameworks, multimodal systems and incremental systems. Please note that a multimodal system, does not necessarily propose a multimodal framework. Therefore the frameworks are discussed separately.

1.1.1 Multimodal Frameworks

Designing and implementing a multimodal interface is a high cost process, therefore multimodal application frameworks are needed for rapid development of such systems[19, 13]. Several researchers have proposed multimodal frameworks to address this need.

W3C has introduced a web-specific multimodal framework [31]. It is a markup language for multimodal systems and does not define system architecture. It is designed to be used as a guideline for implementing multimodal systems. This framework has later been used by some researches for implementing multimodal interfaces [21]. The framework is limited in the sense that it can not handle new innovative ways of interactions easily, mostly due to the fact that it does not support programming languages and most of the programming is only available thought Javascript [41].

Multimodal grammars were proposed in [25, 20] for definition of mixed inputs from different modalities. They have used unification-based approaches and later, finite-state transducers (FST) to parse and fuse different modality inputs [20, 24]. A good feature of multimodal grammars is that they represent the input from different modalities in a single file and can be used for representation of semantic data at the same time. This allows for rapid development of multimodal applications and increases their maintainability.

A more dynamic approach compared to FST model is presented in [42]. The model uses a graph with terminal and non-terminal nodes for grammar representation. This approach allows for on fly expansion of graph during the recognition [42].

A framework which uses spatial ontologies and user context for multimodal integration is described in [18]. Object ontologies are defined in OpenCyc and Microsoft’s Speech API (SAPI) is used for speech recognition. The grammar is defined as a part of SAPI, which means that object and other modalities’ anchors are defined as part of the speech grammar. In other words, non-speech modalities need a specific speech anchor in order to be integrated into the multimodal interaction. For example “put it here” has two words which act as object anchors in the speech: “it” and “here”. The multimodal integration engine then uses these two anchors and object ontologies to perform the integration. This means that without the anchor words, other modalities will be ignored.

A framework for rapid development of multimodal applications is presented in [13]. Although authors accept the fact that system responsiveness is an important factor in such systems, they refuse to perform the fusion during the speech and discuss that such
an approach may lead to constraints on user interaction.

An interesting multimodal system with an incremental pipeline is presented in [8]. Their system which is called RISE, can process syntactic and semantic information from audio/visual inputs incrementally and generates the feedback on-the-fly. RISE’s architecture is based on incremental parsing of speech input with different sub-systems that try to resolve the references in the speech through visual data or the context of dialogue. Therefore RISE is not a fully multimodal system, because non-speech modalities are complementary to the speech and not part of the interaction.

An abstract model for implementing an incremental dialogue manager is proposed in [40]. Although the model is not designed for multimodal interactions, it can easily be modified for that. Other multimodal frameworks can be found at [11, 43, 27, 6].

Like [18] and [8], we also use object references during fusion. The difference is that in order to dismiss the problem of using speech as the main modality, we have defined a multimodal language which allows for definition of any mixture of different modality inputs. Through this language (which we call it COLD) any modality or any mixture of modalities can be used in the interaction. Pure non-speech communications are easily achievable through COLD. We also take a similar approach as [42] for grammar representation and parsing. Our contribution is in expanding this model to include a programming interface that can be used for semantic analysis and feedback generation.

1.1.2 Multimodal Systems

Since the introduction of “Media Room” in Richard A. Bolt’s paper [7], many other systems have been implemented which are based on multimodal interaction with the user. Researchers have employed different methods in implementation of such systems [17, 12, 25]. All multimodal systems are common in the sense that they receive inputs from different modalities and combine the information to build a joint semantic meaning of the inputs. Finite-state multimodal parsing has been studied by Johnston and Bangalore and they present a method to apply finite-state transducers for parsing such inputs [22]. Unification-based approaches are also studied by Johnston [22]. Fugen and Holzapfel’s research on tight coupling of speech recognition and dialogue management shows that the performance of the system can be improved if it is coupled with dialogue manager [14]. A good study on incremental natural language processing and its integration with vision system can be found here [8] and also in [15]. Incremental parsers are also studied for translation purposes [34].

On a higher level of abstraction, speech modality alone has been used by Pires [37] to command an industrial robot through switching between preprogrammed tasks. Also Marin et al. [32, 33] combined both augmented and virtual reality environments together with higher level voice commands to remote operate an industrial robot.
1.1.3 Incremental Parser

Although incremental parsers are not new in computer science [16, 1, 10], they have rarely been used as a tool for parsing of multimodal inputs.

In HRI domain incremental parsers have been successfully used for dialogue management by [29]. They use statistical models to perform parse selection during an incremental parse and use the system for dialogue management in HRI domain. Their results show significant improvements in parse times. In [39], ADE (Agent Development Environment) is proposed. It is an infrastructure with natural language processing capabilities that allows for integration of several intelligent devices through a goal manager to reach a goal. A sample design based on ADE for development of a "smart home" is then presented.

Incremental parsers are also used for generating online feedback during conversational interaction by [28]. Other incremental systems in HRI domain can be found here [12, 30, 9].

1.2 Published Papers

The work on research and development of this system has also resulted several papers that have been published in different conferences. We present these papers alongside their abstracts here.

- **Incremental Multimodal Interface for Human Robot Interaction**
  A.Ameri, B.Akan, B.Çürükli, L.Asplund - ETFA 2010, p.1–4 - Bilbao, Spain [3]
  
  **Abstract:** Face-to-face human communication is a multimodal and incremental process. An intelligent robot that operates in close relation with humans should have the ability to communicate with its human colleagues in such manner. The process of understanding and responding to multimodal inputs has been an interesting field of research and resulted in advancements in areas such as syntactic and semantic analysis, modality fusion and dialogue management. Some approaches in syntactic and semantic analysis take incremental nature of human interaction into account. Our goal is to unify syntactic/semantic analysis, modality fusion and dialogue management processes into an incremental multimodal interaction manager. We believe that this approach will lead to a more robust system which can perform faster than today’s systems.

- **Intuitive Industrial Robot Programming Through Incremental Multimodal Language and Augmented Reality**
Abstract: Developing easy to use, intuitive interfaces is crucial to introduce robotic automation to many small medium sized enterprises (SMEs). Due to their continuously changing product lines, reprogramming costs exceed installation costs by a large margin. In addition, traditional programming methods for industrial robots is too complex for an inexperienced robot programmer, thus external assistance is often needed. In this paper a new incremental multimodal language, which uses augmented reality (AR) environment, is presented. The proposed language architecture makes it possible to manipulate, pick or place the objects in the scene. This approach shifts the focus of industrial robot programming from coordinate based programming paradigm, to object based programming scheme. This makes it possible for non-experts to program the robot in an intuitive way, without going through rigorous training in robot programming.

• A General Framework for Incremental Processing of Multimodal Inputs

Abstract: Humans employ different information channels (modalities) such as speech, pictures and gestures in their communication. It is believed that some of these modalities are more error-prone to some specific type of data and therefore multimodality can help to reduce ambiguities in the interaction. There have been numerous efforts in implementing multimodal interfaces for computers and robots. Yet, there is no general standard framework for developing them. In this paper we propose a general framework for implementing multimodal interfaces. It is designed to perform natural language understanding, multi-modal integration and semantic analysis with an incremental pipeline and includes a multimodal grammar language, which is used for multimodal presentation and semantic meaning generation.

• Augmented Reality Meets Industry: Interactive Robot Programming

Abstract: Most of the interfaces which are designed to control or program industrial robots are complex and require special training for the user and the programmer. This complexity alongside the changing environment of small medium enterprises (SMEs) has lead to absence of robots from SMEs. The costs of (re)programming the robots and (re)training the robot users exceed initial costs of installation and are not suitable for such businesses. In order to solve this shortcoming and design more user-friendly industrial robots, we propose a new interface which uses augmented reality (AR) and multimodal human-robot interaction. We show that such an approach allows easier manipulation of robots at industrial environments.
Chapter 2

Architecture

Developing a multimodal interface for HRI involves several sub-systems (i.e. parsers, modality fusion, etc.). Communication between these systems and their data modeling as well as design of each sub-system and it’s role in the whole interaction process are important factors in designing and developing multimodal frameworks and interfaces. Choosing suitable methods for addressing the aforementioned factors affects behavior and properties of the system; therefore defining the required characteristics of the system in early stages of development can act as a guideline for making those choices. With this regards we defined the desired characteristics of a multimodal framework as follows:

1. The framework should allow for integration of new modalities to the pipeline easily and without changing much of the previous code. This can help developers to add new modalities to the system in a more organized fashion.

2. The framework should include a data modeling system that is used by all sub-systems. This means that all the sub-systems will act based on a central data modeling system, leading to easier interaction between them. With a universal data model the modules can consult each other for removal of vague data, when such information might be needed.

3. Reprogramming and rapid prototyping of multimodal interfaces should be done easily and with little to no changes in the main program.

4. The framework should be able to use other information channels during an interaction to remove ambiguities and build up the response to the inputs.

The first requirement can be directly achieved by modular design of the pipeline and language parser. We will discuss this later in this chapter.

The last requirement is similar to human way of processing multimodal inputs and is in accordance with definition of incremental parsers. Thus the final design should involve an incremental pipeline that passes trough all the modules in the framework. This way,
all the modules from input to response will act accordingly while the inputs are being perceived. Such a design will not only allow for on-the-fly action planning, but also (with a careful design of the data model) partially helps to achieve our second goal: the connected modules in the pipeline can incrementally get the results from other modules and help to remove ambiguity on vague data before the inputs are completely perceived.

In order to implement the third factor in our vision, we chose to use a multimodal language for presenting different multimodal inputs and their related runtime code. Multimodal languages provide a fast way for expressing inputs from different modalities, their relations and combined multimodal semantic representation [23]. Desired properties of a programming language are discussed in [41] as follows:

- **Modality Agnostic**: The language should not be specific to any of the involved modalities. Any new modality should be easily added and programmed when needed.
- **Structuration**: The language should support a mechanism to define the structure of UI. This will allow for easier integration of UI and interaction routines, leading to easier development of multimodal systems.
- **Control**: Such as “if” statements, loops and procedures. These features are essential parts of any programming language and in the case of multimodal interfaces they can be used to develop richer interaction experiences.
- **Events**: They are needed for binding UI with runtime routines. This will also help developers for rapid development of multimodal interfaces.
- **Data Model**: The language should contain a clear data model.
- **Reusable Components**: The components should be reusable.

The above list makes it clear that the data model is an important part of the language too. As mentioned before, a multimodal language not only is used for processing semantic related information, but is also responsible for expressing different modality inputs and relations [23]. This means that, the data model should be designed in a way that can be easily represented by the language as well.

Finite-state grammars (FSG) have proven to be the method of choice for developing such data models [20, 23, 11, 13]. FSGs can be expressed by multimodal languages and are represented by a graph internally, which in turn, is one of the methods used for development of incremental parsers. This means that the language file will define a FSG with all the modalities included and their related semantic processing code. With this approach, developers can add any functionality to the grammar when needed. For example the programmer can attach a function to a phrase in the grammar which consults other modalities for confirmation during the incremental parse.

The multimodal language that we have designed for the framework is called COLD (COactive Language Definition). COLD’S design tries to fulfill all the aforementioned
requirements. The proposed COLD framework is completely modality agnostic, since it handles any number of modalities without any restrictions or preferences. It is easy to add a new modality to the code and define its role in the semantic representation of inputs alongside its functional role in the runtime routines.

UI structuration and UI events can be handled via any .NET programming language in COLD framework. Because the framework is developed as a .NET class library, it is possible to add it to a .NET project and use .NET capabilities for the UI structuration and events. Modality events are a special case and are handled both by the top-layer .NET programming language and also COLD scripts. When a new modality input is received, it is first captured by .NET event handlers and converted to a framework specific format, at this point COLD scripts take control and process the information according to COLD code, which might raise other inter-component events and/or handle those events.

Data models are handled both by the C# and COLD language. C# is responsible for UI related data while COLD is used to handle variables related to semantic analysis and representation. COLD also has the support for calling Prolog routines. It is also possible to write any Prolog code within COLD, which justifies the reusability and control properties mentioned before.

Apart from the aforementioned properties of a multimodal framework, COLD is also an incremental system. Parsing, modality fusion, semantic meaning generation and execution are all performed through an incremental pipeline. With an incremental system it is easy to build up a response to user inputs before they are finished. For example when a person points to an object and is giving commands about it, a robot can turn it’s camera towards the direction and “look” at the object before the command is complete. This helps to build a more human-like interaction between the robot and its human colleague [8].

The COLD framework includes a dedicated parser for each modality and a central parser. The central parser is responsible for modality fusion and performs it incrementally based on COLD definitions. Semantic analyser, which is next in the pipeline, also uses COLD definitions to extract semantic meaning of inputs. It has a Prolog engine at its core to handle Prolog code related to semantic analysis. Context resolver and dialogue manager can be next in the pipeline but they are not part of this project.

According to the above, COLD is an important part of the framework. COLD definitions are used to (1) generate separate grammars for each modality, (2) define the fusion pattern, (3) define the semantic variables and calculations and (4) define dialogue patterns and dialogue turns. This means that COLD affects the whole process from input processing to modality fusion, semantic analysis and dialogue management.

The next sections will continue by discussing COLD language’s design, syntax and usage. Then we describe the multimodal graph (data model) which represents multimodal inputs. It will be followed by the incremental parser and modality fusion modules. In
the last section the semantic analysis and execution will be covered.

2.1 COLD Language

In a similar approach to multimodal grammar definition represented in [20]; COLD language is capable of defining the grammar of multimodal sentences and their respective semantic representation. We have expanded Johnston’s grammar to (1) include optional phrases, (2) support semantic variable handling and (3) include semantic and interaction related code. Please also note that COLD uses a different method for parsing multimodal inputs which will be discussed later in this paper.

2.1.1 COLD Syntax

A sample of COLD code is shown in Figure 2.1. COLD’s syntax can be divided into two subcategories: (1) grammar definition and (2) semantic representation. Semantic representation code is always inside curly braces and in the picture is seen in violet color. It can be any Prolog code which is syntactically correct. The rest of the code defines the grammar. It is build up of blocks with the following syntax:

\[
\text{predicate (variable list)} \rightarrow \text{atom/predicate list}
\]

In a Prolog-like syntax, the “\(\rightarrow\)” sign, separates a predicate from its definition. Predicates can define variables which will later be used in semantic analysis. These variables are defined inside parenthesis that follows predicate’s name. Predicate definition is a space-separated list of atoms or reference to other predicates. In order to separate atoms for different modalities a colon ‘:’ is used. For example the first definition of “pickup” in Figure 2.1 is \(\text{pickup:singleObjectSelect}\); it means that \(\text{pickup}\) is the atom for first modality and \(\text{singleObjectSelect}\) is an atom for the second modality. Please note that the COLD code has no information on what these atoms mean and what modality they represent. It is later in the input registrars and parser that each modality translates the information into its related meaning. In this case, our modalities were speech and mouse input respectively. This means that the speech parser is responsible for parsing \(\text{pickup}\) and mouse parser is responsible for parsing \(\text{singleObjectSelect}\), whatever they mean.

When a predicate definition list contains references to other predicates, it must be enclosed within “\(<\)” signs with all the variables that should be passed to it. For example the second definition of \(\text{pickup}\) contains \(\text{<object(TargetObj,Props)>}\) which refers to \(\text{object}\) predicate and passes \(\text{TargetObj}\) and \(\text{Props}\) variables to it. It should be noted that variables are passed in a call-by-reference manner and therefore, in this case, \(\text{Props}\) is a variable that will be assigned a value later.

Finally, optional phrases are enclosed in square braces “\([\]” within COLD code. Optional phrases are atoms or predicates which do not have to be present in the input for
Figure 2.1: Some Sample code of COLD

```
command(ObjetAnchor, Retval) --> <pickup(ObjetAnchor, Retval)>
{call(Retval)}.
command(ObjetAnchor, Retval) --> <putin(ObjetAnchor, Retval)>
{call(Retval)}.
command(ObjetAnchor, Retval) --> <execute>.

pickup(TragObj, Retval) --> pickup:singleObjectSelect
{TragObj=mouseReturnvalue, Retval=pickup(TragObj)}.
pickup(TragObj, Retval) --> pickup <object(TragObj,Props)>
{Retval = (Props,pickup(TragObj))}.

object(TragObj, Retval) --> a wooden block {Retval - property
(TragObj,name,woodenblock)}.

putin(TragLoc, Retval) --> put it [here]:singleObjectSelect
{TragLoc=mouseReturnvalue, Retval = (holding(CurObj), putin
(CurObj, TragLoc))}.
putin(TragLoc, Retval) --> put it in <object(TragLoc,Props)>
{Retval = (holding(CurObj), Props, putin(TragLoc, CurObj))}.

execute --> execute {execute}.
```

generation of a complete multimodal sentence. For example the first definition of putin
in figure 2.1 contains “[here]”. Since this refers to the first modality which in this case is
speech, it means that the user can omit the word “here” and still get the same response.
Thus saying “put it here” and “put it” will generate the same result. Please note that
the mentioned putin definition also includes a mouse modality atom that is not optional
and therefore, user has to click the desired location in order to complete the sentence
(but not in that same exact order).

2.1.2 COLD Parser and Compiler

An open source parser has been used to parse COLD scripts that we write it in our
integrated development environment for COLD (COLD IDE) . COLD IDE is an envi-
ronment for writing and compiling COLD code. Since we perform speech recognition
through Microsoft’s SAPI, we also included a converter in COLD IDE that can convert
COLD code to SAPI specific XML file for speech grammar definition. Obviously it only
extracts speech related data from the code in its conversion.

The IDE’s compiler generates a multimodal graph that is later used by the parsers.
The graph and its role in parse are discussed in the next section. Figure 2.2 shows A
screenshot of the development environment.
2.2 Incremental Multimodal Parsing

In order to achieve an open design that allows for integration of new modalities into the system, COLD performs parsing of inputs through dedicated parsers for each modality. All unimodal parsers have access to a shared copy of the multimodal grammar graph and a pool of different hypothesis on multimodal inputs.

2.2.1 Multimodal Grammar Graph

Multimodal grammar graph, which we will call simply “graph” afterwards, contains all of the data defined in COLD code. It includes three different type of nodes: (1) rules, (2) rule references and (3) phrases. Rules contain a list of all the phrases and rule references that define its grammar. The list is ordered in the same manner as COLD code. Rule references are simply references to other rules in the graph and phrases are simple atoms that can be matched and parsed by unimodal parsers.

Each phrase or rule reference node contains a Prolog code block object. The Prolog object stores related semantic code for that node. Prolog code objects are optional and can be defined for a node when that node has a special role in semantic analysis or interaction. This design allows for incremental semantic analysis since each node can execute its related prolog code as soon as the parser moves to the next node.
Optional phrases are defined and included as normal nodes into the graph, but contain a boolean property that marks them as optional phrases. Parsers can later decide if they want to include or exclude the node in their final parse outcome. Please note that no extra graph nodes or edges are added for optional phrases.

All the nodes also include a list of variables that they can access. Since variable values may be different for different competing hypotheses during a parse, the values are not defined at graph level.

Different modalities are simply represented by an integer value for each node that acts as an ID for that modality. Parsers later use this value to decide if they are allowed to process the node or should they wait for responsible parser to process it.

2.2.2 Hypothesis

Hypotheses are objects that include parse related data. Their most important job is to store values for different semantic variables during a parse. Each hypothesis also contains a list of references to graph nodes that build up that hypothesis. Since rule-reference nodes of the graph only include a reference to their respective rules, hypothesis objects have the ability to expand them when needed. This means that if a rule reference is to be parsed and it gets a match on the first atom of the referred rule, its representation in the hypothesis is replaced by the complete representation of referenced rule. In situations like this, it may be possible that a rule expands into several different lists. Therefore, new copies of the hypothesis object and its related semantic variables are created and added to the pool.

Apart from semantic variables and graph references, hypothesis objects also store values that indicate parse time, responsible unimodal parser for each node and parse type (if a node is parsed as an optional phrase or not).

2.2.3 Incremental Parser

Different components are involved in parsing, these components and their relations are shown in figure 2.3 and include:

- Grammar: The grammar component is a copy of the multimodal grammar graph discussed before.
- Hypothesis Pool: Contains a list of all active competing hypotheses.
- Input Registrars: Each modality needs at least one input registrar. Their job is simply to capture the data, pack it and send it to the central parser.
- Unimodal Parsers: Each modality has a dedicated parser. These parsers only communicate with central parser. They have read access to grammar graph and read/write access to hypothesis pool.
Figure 2.3: A diagram of different components involved in parsing.

- Central Parser: Receives data from registrars and sends it to its respective parser. It also detects completely parsed hypotheses and fires events in such cases.

Figure 2.3 shows the components as they are in our current setup used for the experiments. Of course there may be more input registrars and unimodal parsers for other scenarios. In fact, for each modality, one parser and at least one input registrar should be implemented. All the unimodal parsers inherit their functionality from a base class which is included in the framework. This makes implementation of new modality parsers easier, since most of the parsers won’t need to change the behaviour of the base class.

A unimodal parser, only considers the nodes in graph which are related to it, such nodes have the same modality value as the parser. Unimodal parsers are allowed to jump over other modality nodes, only if a match can be found. This feature is added to support fusion of inputs even if they are not exactly in the order as the grammar. It will be discussed more in modality fusion section.

2.3 Modality Fusion

A direct result of using multimodal grammar graph is that parsers will also contribute to fusion during their parse. Because different modalities are already represented in a unified format in the graph; as the incremental parsing goes forward, the results from different parsers, will be fused together consequently. This means that there will be no need for a separate fusion system. Parsers and the central parser, all are part of the fusion system. Therefore, parsers should be implemented with fusion constraints in mind.

An important factor in modality fusion is synchronization of different modalities. Inputs from different modalities will not exactly be in the order defined by multi-
modal grammar. For example the \texttt{put in} command (Figure 2.1) is defined as \texttt{put it [here]:singleObjectSelect}, but it should not mean that the object selection can only be accepted after saying the word “here”. Related inputs from different modalities have a relation in time also. The time relation between speech and its related gesture is reported to be 1000 ms in [42] and between 1000 ms to 2000 ms in [24]. We have used a 2000 ms time difference for resolving related multimodal inputs.

Because of on-the-fly design of the fusion in the framework, the two seconds time difference between related multimodal inputs should be implemented in parsers. For this reason, parsers are allowed to postpone a parse request by central parser, if the next node in the graph does not have the same modality as the new input. In such cases, parsers store the information in a queue and return a value to the central parser which indicates a postponed parse. Upon arrival of a new input, the central parser first sends the new input to the respective parser and asks the postponed parser to check for a new parse. The parser then decides to (1) accept the data, if the input from other modality has been satisfied; (2) remove the data if the two seconds time difference does not approve; or (3) request for another postpone if data can not be parsed but the two seconds difference has not been met yet.

2.4 Semantic Analysis

Semantic analysis can be referred to as a search on the set of possible target objects and actions. In this sense each phrase of each modality can affect the final outcome of semantic analysis. An easy way of performing this search is to define different variables and narrow down their values based on the inputs from different modalities. Prolog language has proven to be the language of choice in such cases for many researchers, mostly due to its ease of use.

The COLD framework supports Prolog predicates and variables within COLD language for semantic analysis. Each phrase or rule reference can have its related Prolog code and variables. When they are successfully parsed, the respective Prolog code is executed and variables are updated. Processing of Prolog codes is performed through SWI-Prolog engine.

Because rule-references act like a function call to a rule, all the variables that are defined within the rule reference should be passed to the rule as well. This happens as a call-by-reference function call, meaning that further changes to the values of the variables are also visible to caller functions as well. This design helps us to define semantic variables in the highest level of the grammar and allow lower parts of graph to manipulate them. The final value of the variable - which is defined by all the subsections of the containing rule- is the semantic representation of that rule.

As discussed before, hypotheses contain a variable table that stores a list of all semantic variables and their values. At runtime, when a new hypothesis is formed, its high
level variables are defined and added to its variable list. As the parse continues, execution of semantic Prolog statements, assigns new values to these variables. A special case happens when a rule-reference is parsed and expanded; in this case, the rule-reference may be expanded to several different smaller hypotheses. Therefore the whole variable table and its values are copied and assigned to them.

Prolog statements in COLD language are not used for semantic analysis only, but are also useful for real-time feedback to the user. For example, the pickup phrase can contain a predicate that highlights all pickable objects after receiving the word “pickup” from audio channel. This happens during the speech by the user and helps the user to develop a better interaction with the system.

Figure 2.4 shows system’s output during a parse. In order to fit the output in the paper, it has been edited. At lines 2 and 3, SpeechRegistrar reports the new input: “put”. Central parser creates two hypothesis based on it (lines 6,7) and reports their respective semantic representation (lines 8,9). Please note that at this point the semantic representation only includes empty Prolog variables. After arrival of word “it”, both hypothesis stand and since the word has no semantic code assigned to it nothing changes (lines 11-18). At line 20, a mouse input arrives and it carries the selected object (object119) with it. Now the central parser can parser the “singleObjectSelect” node, which belongs to mouse modality and this leads to a more meaningful semantic representation for that (lines 20-28). It should be noted that in this case, the parser did not wait for the word “here”, because it is defined as an optional phrase in grammar. Please also note that the other hypothesis is not removed from the memory, but is assigned a lower score at this point which is not shown here. After arrival of the word “here”, the parser fires a recognition event and reports its semantic meaning.

As a final note on system outputs we should point out that the changes that are made in variable names during different steps of parse are due to variable copying that was described before.
Figure 2.4: output of the system during a parse. The input consisted of the sentence “put it here” and a click. Please note how the semantic variables change and how fusion happens during the incremental parse.
Chapter 3

Results

In this part four sets of experiments will be presented. These experiments were designed to test the framework in different scenarios. The first experiment is based on our multimodal testbed which is a simple desktop application that includes a multimodal interface for user interaction. Three experiments will follow it and they use the framework to program and control an industrial robot.

Our setup consists of an ABB IRB140 industrial robot with a gripper and a camera attached to the 6th joint, therefore the user can view the world through the robot’s eyes. The images from the camera are overlaid with the transparent virtual objects creating an augmented reality (AR) user interface. Using the mouse, the user can select objects and give voice commands to pick them up and to put them in a location. The semantically analyzed voice and mouse commands are converted to simple Prolog commands such as pickup (ObjectName), which are later expanded to approach, and grasping patterns and finally converted to RAPID language (programming language for ABB robots), and sent to the robot controller for execution.

Figure 3.1: Screenshots from the AR UI are presented here. (a) The blue objects are highlighted after user says “pickup a blue object”. (b) While the robot is holding a green object, the user starts saying “put”, and all empty locations are highlighted.
To assist the user, we highlight possible objects that are applicable to the given command. For example in the pickup case, when the user says the words “pickup” all pickable objects are highlighted. As the user continues to describe which object he wants the robot to pickup, irrelevant highlights are removed. In figure 3.1 samples of this highlighting can be seen. The incremental nature of our system can be used to generate early feedback and assistance to the user even before the voice commands end. It is also useful in conditions when user commands are ambiguous; the feedback can help the user to recognize the ambiguity before finishing the sentence.

Figure 3.2: Setup for experiments 1 and 3.

The subjects have been asked to pick and place wooden blocks according to the tasks given to them (fig. 3.2). The palette has been designed so that it has 10 drop-locations. Nine of them are organized as a 3x3 matrix, whereas the tenth drop-location is places further away from the others towards right. In each location the robot can place a wooden block. The wooden blocks themselves have a drop-location over their top so that the blocks can be stacked on top of each other (this feature is blocked for
one of the tests). The experiments are carried out with four subjects. Two of them are expert robot programmers, and two are engineers not having previous experience in robot programming. Preceding each experiment, the subjects received oral instructions on what they will perform; similar to how it would be in an industrial environment. The goal was also to keep the instructions to a minimum, and let the subject discover the system by testing it. Oral instructions were, thus, limited to 5 minutes for each subject.

3.1 Multimodal Interface Test Application

The Multimodal Interface Test Application is a part of COLD IDE (see fig. 2.2). It includes a 2D screen containing several rectangles of different colors. Users can manipulate these rectangles (i.e. move them to a new location) based on the COLD program defined in the IDE. Of course this simple testbed cannot be used to test complex COLD programs, but it can be used to test simple COLD functions and predicates during the development and before deploying them on other targets such as robots. In order for the test application to work properly, developers will have to implement their own EventRegistrars in C#.

For this test we designed a simple COLD grammar to move the rectangles around. The interaction involves two modalities: speech and mouse click. Users are allowed to give three simple commands: pick, put and move. They can refer to objects either by their color or by selecting them through the mouse input. They can also select a location by clicking on it or by vocally defining the location (i.e. “lower”, “left of this”, etc.).

Figure 3.3 shows the results when the user says “move this here” and simultaneously clicks on one of the rectangles and clicks on a location for the final position of the rectangle. Please note that the optional definition of some parts of the grammar will allow the user to use any incomplete sentences like “move”, “move this” or even “move here” while the information from other modality is capable of filling the semantic outcome. It is also possible to use only verbal commands i.e. “move the red rectangle to the left of blue rectangle”.

![Figure 3.3: Multimodal test box. Before (left) and after (right) performing the move command.](image-url)
3.2 Block Stack Up

As an introductory warm up task, the subjects were asked to stack the wooden blocks to make a stack. In this experiment the subject would choose a drop-location, and move wooden blocks placed in other drop-locations to the target drop-location. Note that the target drop-location does not have to be empty prior to the experiment, i.e. it may be occupied by one or more wooden blocks as in figure 3.4. The purpose of this task is to make the users familiar with the commands and the operation logic of the system. Once the users have gained experience, they easily moved the wooden blocks around, and build stacks with them.

After roughly 5 minutes of testing the subjects felt that they can interact with the robot easily. In this initial experiment there were no differences between the performance of experienced robot programmers, and other subjects. Thus, all four subjects find the interaction the robot straightforward using the instructions.

![Figure 3.4: Experiment 2, where the users were asked to build stacks of wooden blocks.](image1.png)

3.3 Kitting

For this test we used a kitting theme including a palette with empty places. Red, blue and green colored objects in 3 different stacks are placed on the workbench. The task is to create a robot program for rearranging the colored objects on the palette. All the users start from the same setup of the blocks.

The Prolog statements generated at the end of the semantic analysis can be saved to a file, and recalled later on to recreate the robot program. Figure 3.5 shows the test setup for this experiment.
3.4 Block Sort

In this experiment the subjects perform a more complex task. They were asked to sort the numbered wooden blocks in sequential order starting from WoodenBlock1 at the top-left to WoodenBlock9 at bottom-right as shown in figure 3.5. Prior the experiment wooden blocks are placed randomly on top or each other, or on empty drop-locations. This task is more challenging than other experiments, since it requires a temporary storage area. In addition, the problem solving skills of the subjects plays an important role, since different strategies could be utilized for solving the problem. The simple and naive way would be to swap the blocks using the empty space on the wooden board one by one. However, this requires many code execution operations. A more efficient way to solve the problem would be to stack the wooden blocks in an ascending order on the empty space on the board while freeing up the target locations and then move them into the desired drop-locations. Using such a method it is possible to give multiple pick ad place commands in one execution batch. Which method to be followed depends on the experience and intuitive of the operator.

This simple task demonstrated how subjects just immediately after the start of the experiment constructed a mental model of the problem and solved it. Obviously, based on background the solutions differed considerably. One thing was common to all subjects though. They all could solve the problem as they planned initially, and without feeling
that they are limited by the system.

Figure 3.6: Setup for experiment 4, where the users were asked to sort the numbered wooden blocks in an ascending order.
Chapter 4

Discussion and Future Work

In this chapter we discuss our results and plans for further development of the system. It should be noted that this project has been a part of a bigger project within school of Innovation, Design and Technology at MDU and with this regards it does not represent a complete finished product.

4.1 Discussion

We have designed and implemented an incremental framework for development of multimodal interactions and tested it in several scenarios. The experiments can be discussed from two different viewpoints: first, the COLD framework and its use in development of multimodal interfaces and second, the multimodal interface itself and its usability in human-robot interaction domain.

4.1.1 COLD Framework

The four different test experiments, demonstrate how COLD can be used for development of multimodal interfaces. Implementation of those experiments required two pairs of parsers and input registrars for the two modalities involved. When these components are in place, COLD makes it easy to program different scenarios for the interaction as shown by different tests. With COLD framework, developers will focus on the grammar and semantic representation of the inputs. They can also plan robot’s feedback through COLD code and in the same grammar.

COLD is also an incremental system. Incremental processing of inputs through COLD allows for in-time interaction between the robot and its human colleague. This was presented in our third test (kitting scenario, fig. 3.5 and fig. 3.1), where the AR interface gave feedback to the user by making selections during user speech. Users did not have to finish the command and wait for the system response; as soon as they say “put”, the system highlights all available drop locations. This creates a more natural interaction
between human and the robot, which in turn leads to less ambiguity and more robustness in the system.

Inclusion of optional phrases in COLD grammar is another beneficial feature of the framework. With optional phrases developers can create more complex and natural grammars with less code. Optional phrases won’t affect the semantic results, but can help to improve system robustness, by considering different forms of a sentence.

It is also easy to integrate COLD within .NET projects, because it is developed as a .NET class library. This let’s the developer to implement the required UI in Visual Studio environment and use COLD as an input module. Development of unimodal parsers and input registrars for new modalities can also be done in the same project and COLD framework supports their development by offering base classes for them which can handle most of the required code.

4.1.2 Multimodal HRI through Augmented Reality

COLD has been designed and developed with an ultimate goal of developing a new intuitive programming language for industrial robots. Our tests not only plan to examine the COLD framework, but also examine usefulness of multimodal augmented reality system is robot programming domain.

The proposed system demonstrates an alternative method for interaction between industrial robots and humans. Using natural means of communication is definitely an interesting alternative to well-established robot programming methods. These methods require considerable larger amount of time, and perhaps more importantly, a programming expert. In the experiments it is demonstrated that efficient collaboration between the robot and its human peer can help to clarify vague situations easily. Multimodal language seems to allow the system to understand the user’s intentions in a faster and more robust way.

There is no doubt that additional tests have to be carried out to fully test the system. It is, however, evident that the users with robot programming background do not feel that they are restricted due to the interaction method. Users with no experience in robot programming, express that they are as fast at robot programming as their expert counterparts.

COLD language also makes it possible for the user to assist the robot. This means that some of the computational load is shifted to the user. One example is when the user pointing out regions in robot’s operational space through the clicking modality. There is, thus, no need for a lengthy description of the location. In addition the clicking modality also helps to reduce the search space of the object recognition module (not demonstrated here). Thus, with multimodal communication both the robot and the operator can help each other in the way they are best at. This opens new possibilities for human–robot interaction in industrial environments, and hence is an evidence for the usefulness of this paradigm shift.
4.2 Future Work

As described in [40], a complete system for managing human-robot interactions should also include a dialogue manager. A dialogue manager can help developers to plan for different scenarios during an interaction and improves the system’s robustness by guiding the robot through dialogues that might need several commands to be understood and executed. We have made plans for integration of a dialogue manager through COLD language and the work on it’s development is in progress now.

Another beneficial feature for the system would be addition of a context manager that can resolve references to previously mentioned objects and commands in the interaction. This can be easily achieved by treating the context as a virtual modality in the system. By virtual modality, we mean a modality which has no inputs, but has its own parser; in this sense the context manager can access hypothesis pool for unresolved references and can suggest solutions for resolving those issues with referring to previously acquired data through the interaction.
Bibliography


