A New Way to Interact With Robots

A thesis submitted in partial fulfillment of the requirements for the degree of Master of Science in Computer Science and Engineering

By
Adeline Duong

Dr. Monica Nicolescu/Thesis Advisor

May, 2016
THE GRADUATE SCHOOL

We recommend that the thesis prepared under our supervision by

ADELINE DUONG

Entitled

A New Way To Interact With Robots

be accepted in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE

Monica Nicolescu, Ph.D., Advisor

Mircea Nicolescu, Ph.D., Committee Member

Yantao Shen, Ph.D., Graduate School Representative

David W. Zeh, Ph.D., Dean, Graduate School

May, 2016
A New Way to Interact With Robots

Abstract

This thesis presents a new and portable robot control architecture for a multimodal user interface for interactive conversations with a robot. Multimodal user interfaces allow the robot to interact with more than one person within a group. Previous research produced architectures that allowed the robot to respond to different people in the conversation and determine which person is talking. However, there has been no architecture that can keep track of a multi-person conversation in the same way that a human can, by establishing and maintaining an identity for each person involved in the conversation. This thesis presents an architecture that attempts to mimic the way humans interact and view a conversation, allowing the robot to carry different conversations with multiple people.

The proposed architecture brings the following main contributions. First, it allows for online learning of people’s faces, which can later be used for appropriately tracking the conversations. Second, it tracks the faces that are recognized in the field of vision to determine who is in the conversation and who has left the conversation. Third, this architecture allows for the robot to respond differently to various people who are in a different stage in the conversation. The robot can give an appropriate response to different people by keeping track of each person and where they are in the conversation. Fourth, in addition to using vision and facial recognition to track and identify users, the architecture will also incorporate sound localization
to determine who has spoken. The architecture will also incorporate speech recognition to determine what was spoken.

To achieve these goals, the architecture uses finite state machines (FSM) to track interactions between the robot and the person. For each person who joins the conversation, an instance of the FSM is created for that person. The FSM records each person’s interaction as well as the robot’s response to the user’s interaction. FSMs also facilitate correct responses from the robot.

While existing implementations have been done on specialized robots in the past, this thesis presents a portable architecture that utilizes robots that are already on the market and libraries that are readily available. The advantage of such an implementation is the potential for further development, to an ‘off the shelf’ implementation that can be adapted for use on different robots. While the implementation in this thesis has not been tested on different robots, it has been successfully emulated on a laptop as well as the NAO robotic platform.
Table of Contents

Abstract ................................................................................................................................. i

List of Tables ........................................................................................................................ iv

List of Figures ........................................................................................................................ iv

Chapter 1: Introduction ........................................................................................................... 1
  1.1: Motivation ..................................................................................................................... 1
  1.2: Contribution .................................................................................................................. 1
  1.3: Approach ....................................................................................................................... 3

Chapter 2: Related Work ........................................................................................................ 6

Chapter 3: Methodology ......................................................................................................... 8
  3.1: Hardware ...................................................................................................................... 8
  3.2: Software ...................................................................................................................... 12
    3.2.1: NAOqi ..................................................................................................................... 13
    3.2.2: OpenCV .................................................................................................................. 15
  3.3: Design .......................................................................................................................... 16
    3.3.1: The Perceptual Capabilities: Face detection, training, recognition, and tracking ...... 17
    3.3.2: The Control Architecture: State Machine and Conversation Tracking ................. 18

Chapter 4: Face Detection and Recognition ......................................................................... 20
  4.1: Face Detection .............................................................................................................. 20
  4.2: Face Recognition .......................................................................................................... 21
  4.3: Implementation ............................................................................................................ 22

Chapter 5: Control Architecture for Human-Robot Interaction ............................................. 27
  5.1: Class objects and structure ......................................................................................... 27
  5.2: The Finite State Machine ............................................................................................ 28
  5.3: Design and Functionality ............................................................................................ 31

Chapter 6: Experimental Results ......................................................................................... 34
  6.1 Face Recognition Results ............................................................................................. 34
  6.2. Human-Robot Interaction Results .............................................................................. 37

Chapter 7: Conclusion and Future Work ............................................................................. 44

References ............................................................................................................................... 46
List of Tables

Table 1: Comparison of NAO and laptop’s processors. ................................................................. 12
Table 2: Version A script for user and robot interaction. ............................................................. 29
Table 3: Version B script for user and robot interaction. ............................................................. 29
Table 4: List of scenarios that were tested. ..................................................................................... 37

List of Figures

Figure 1: The NAO humanoid robot is small enough sit on a table and large enough to contain all the necessary functions to interact with multiple users................................................................. 9
Figure 2: Location of the NAO’s speakers [3]............................................................................... 9
Figure 3: Diagram of the location of microphones on the NAO head [3].................................. 10
Figure 4: The NAO’s vision range of the two cameras as seen from the side of the robot [3].... 11
Figure 5: The NAO's vision range of the two cameras as seen from the top of the robot [3]...... 11
Figure 6: How to threshold one pixel against its neighbors [5]. ............................................... 22
Figure 7: Flow chart diagram representing the overview of the face detection and facial recognition portion of the architecture. ...................................................................................... 23
Figure 8: Flow chart diagram of Stage 1 of the face detection and facial recognition............ 24
Figure 9: Flow chart diagram of Stage 2 of the face detection, recognition, and training........ 26
Figure 10: Flow chart diagram of version A FSM. ...................................................................... 30
Figure 11: Flow chart diagram of version B FSM. ...................................................................... 31
Figure 12: Flow chart of the control architecture for the multi-threaded robot conversation with various people. ......................................................................................................................... 32
Figure 13: New face that has not been trained. There is a red circle around the face. .......... 35
Figure 14: Face that has been trained. There is a green circle around the face..................... 35
Figure 15: The program has trained one face and detects a new face. The trained face has a green circle and the new face has a red circle. The program is only training on the new face............ 36
Figure 16: The program has detected and successfully trained two faces. Since the two faces are trained, the program will no longer train the recognizer................................................................. 36
Figure 17: Version B state machine test run on the prototype architecture during the computer simulation. ........................................................................................................................................ 38
Figure 18: Version A of the FSM; one person interacting with the robot. ............................. 39
Figure 19: Version A of the FSM; two people have consecutively joined the conversation and interacted with the robot. One participant completes interaction before the next participant joins the conversation. ................................................................................................................................. 40
Figure 20: Version A; two people joining the conversation at different points. ..................... 41
Figure 21: Version B; two people joining the conversation at different times....................... 42
Chapter 1: Introduction

The goal of humanoid robotics research is to create a robot that can interact with humans in a very human-like way. One of the challenges in human robot interaction is conversation. Previous research has developed ways for a robot to interact with humans in a one-on-one fashion. There have been robot architectures that can interact with one person or interact with multiple people as a group. The architecture presented in this thesis is a new way for robots to interact with people. The architecture allows for the robot to keep track of different people’s conversations and interact with people individually through the use of FSMs.

1.1: Motivation

An important aspect of designing social robotics is developing behaviors that allow a robot to be more sociable and act in human-like ways to be more engaging and relatable to humans. One way to create a more natural human-robot interaction is to design the robot to mimic the way humans handle conversations. The robot should be able to track and be involved in conversations just as people are able to, with all the elements that come with conversation: acknowledging new people, keeping track of who is in the conversation, know who is speaking at the moment, and keeping track of who left the conversation. The robot needs to remember what each person said and be able to respond correctly. In this way, the robot can act more naturally and respond differently to various people.

1.2: Contribution

This thesis presents a new, portable architecture that is capable of on-line face training and face tracking for modeling multiple participant conversations. This architecture is capable of
keeping track of past interactions between people and the robot and people who leave and join the conversation.

The architecture brings the following main contributions:

1) **On-line training and facial recognition** – The architecture is capable of on-line face detection, training and facial recognition. Using this architecture, the robot is able to recognize new and old faces and decide whether new faces need to be trained.

2) **Face and user tracking during conversation** – The architecture allows the robot to remember and keep track of faces that appear in the robot’s field of vision during the conversation. The robot is able to keep track of when and who joined the conversation, who is participating in the conversation, and when and who left the conversation. Each face is tracked individually so the robot holds an overarching view of how participants are joining, participating, and leaving the conversation.

3) **Multi-threaded conversation** – This architecture is capable of tracking conversation individually with multiple people. The multi-threaded approach to tracking individual conversation allows the robot to respond differently to various people who are at different stages in the conversation. For example, the robot can be greeting a new individual who has joined the conversation while, in the next phrase, respond to a question from an existing participant.

4) **Sound localization** – Not only does the architecture use vision and facial recognition to track people in the conversation, the architecture also uses sound localization to determine who has spoken. Together with vision and sound the robot can accurately determine who is speaking and at what time they are speaking.
5) **Portable implementation** – Past research papers have developed implementations that were run on specially designed robots. These robots were specifically designed for the architecture and conversational modeling. This thesis presents an architecture that is designed and built with existing libraries and implemented on the NAO humanoid robot. This gives the architecture the potential to be developed into a portable ‘off the shelf’ implementation.

1.3: **Approach**

A specific architecture is designed to efficiently store interactions between multiple people and the robot in the conversation. This architecture is combined with a supporting program in a test platform for demonstration of the architecture.

The architecture will model each participant as state machines. Using face detection and sound localization, the robot can distinguish who is talking and who has joined or left the conversation. The current state of the conversation can be viewed as the most recent node in the list of each participant. This is a simple, easy, and intuitive approach to modeling conversation. The purpose of this architecture is to model the conversation as closely as possible to a script-based model, similar to how humans intuitively multitask in keeping up to date with all the happenings in a conversation.

The architecture has been programmed in Python 3.4. This is a high level dynamic language that does not need to compile in order to run. As a result, it is ideal for prototyping and development. Face detection and recognition is achieved through the OpenCV image processing
libraries. Python application program interfaces (APIs) are used to access the functions in the OpenCV libraries. The architecture is implemented and tested on the NAO humanoid robot.

The approach for each element of the architecture is as follows:

1) **On-line training and facial recognition** – Recognition is achieved through the OpenCV libraries, which are capable of accessing and editing video frames, running face detection, training the facial recognition module, and running facial recognition.

2) **Face and user tracking during conversation** – Each face is tracked individually. An instance of a “Face class” is defined for each new face that appears in the robot’s view. This class, along with the user class, is used track individual faces and record when participants join, participate, and leave the conversation.

3) **Multi-threaded conversation** – Each user is assigned an instance of the “user class”. This class keeps track of when the user says something, the responses of the robot, and when the user joins and leaves the conversation. This class also holds the FSM that controls the interactions of the robot.

4) **Sound localization** – This is done through the Python APIs available in the NAOqi libraries. The sound localization process accesses the robot’s microphones and determines the angle from where the sound is heard.

5) **Portable implementation** – The architecture has been implemented in Python 3.4 and was run on a Ubuntu 14.01 operating system (OS) on a laptop. The laptop remotely accesses and controls a NAO humanoid robot.

The remainder of the thesis is structured as follows:
- Chapter 2 presents related work and past research in the field of conversational modeling in robots.

- Chapter 3 details the hardware and software methodologies for designing and implementation of the architecture.

- Chapter 4 describes the vision processing portion of the design including sections in face detection and face recognition and training.

- Chapter 5 describes the control architecture that is responsible for conversational modeling and controlling responses of the robot.

- Chapter 6 shows the results of the tests performed with a prototype implementation the laptop and with the implementation on the NAO robot.

- Chapter 7 discusses future works and conclusions.
Chapter 2: Related Work

In response to the increasing demand for robots to interact naturally with humans, significant research has been done in the field of human-robot interaction. With respect to human robot conversations, there are several major topics or components to develop a more lifelike robot that can carry a meaningful conversation: 1) modeling an individual conversation, and 2) modeling multi-user conversation. There have been several research papers that have presented architectures and designs for conversation modeling.

Kai-Yuh Hsiao et. al published one of the first papers recognizing the importance of modeling the conversation element of the robot’s environment [1]. The paper presented a perceptually-coupled physical simulator so that the robot would contain an internal model of the world around it. The robot could keep track of objects that are currently in its field of view, as well as store a history of the objects that were once visible, but that are now no longer present. This allows the robot to have a larger field of view than what is seen through the camera lens of the robot. This internal visual model of the robot’s surroundings was used for “language grounding”. It allowed the robot to visualize the left side of the robot versus the left side of the user (who is across from the robot). The robot could change points of view from its own point of view to the user’s point of view. The extended visual capabilities of the robot were used to allow the robot to obtain a more cognitive understanding of the conversation.

Since then, there have been several developments on how to effectively model a conversation. Maren Bennewits et al. describes an architecture that is capable of monitoring a conversation involving multiple participants [2]. The robot maintains information about users
even though they may be temporarily out of the field of view of the robot. The robot uses a method to localize the speaker so that even though the robot may not see the speaker in its current field of view, it can still interact and acknowledge the speaker. This creates a more efficient and lifelike interaction. The robot was custom designed to allow facial expressions. The face of the robot can express: joy, surprise, disgust, anger, sadness, and fear. The underlying structure of the architecture for the function of the robot is an FSM. The robot’s task was to emulate a tour guide and speak to people who attract its attention and would like to know more about certain exhibits. States in the FSM represented different modes for the robot. Examples of the states are: small talk, explain exhibit, explain in more detail, and select exhibit to explain. The architecture modeled single threaded conversation in a one on one fashion: the robot vs. people in the crowd. The robot did not track individual conversation with each person in the crowd. The speaker localization program was developed in-house and commercial software was used for speech recognition.

While the above methods provide significant capabilities for dialog and human-robot interaction, they only address some of the capabilities that would be required. This thesis presents an architecture that can: perform online face tracking and recognition, track faces during conversation, model multi-threaded conversation, perform sound localization, and be portable to different platforms.
Chapter 3: Methodology

The architecture will be implemented within the hardware on the NAO humanoid robot. Due to the slow processing of the robot, a laptop running the Ubuntu Linux operation systems is used to remotely control the robot and process inputs. The laptop connects to the robot via IP address through an internet connection. The Python 3.4 language is used to implement the architecture. Supplemental libraries of Python APIs were used for additional functionality, control and processing. Open CV python libraries were used for vision processing, such as, face detection and facial recognition and training. The NAOqi libraries were used for robot control.

3.1: Hardware

The project was programed using Python 3.4 code in a LINUX Ubuntu environment. A humanoid robot was required to run and test the code. As a result, a small humanoid robot called the NAO was selected to run and test this project. As seen in Figure 1, the NAO is small enough to sit on a table top, but large enough to contain all the necessary functions for interacting with multiple users.
The NAO humanoid robot is small enough to sit on a table and large enough to contain all the necessary functions to interact with multiple users.

The NAO contains two loudspeakers, with a single speaker each on the right and left side of the head. Figure 2 shows the locations of the speakers on the NAO.

Figure 2: Location of the NAO’s speakers [3].
There are 4 microphones for the NAO to receive sound input. They are located in the front and in the back, on both left and right sides. Figure 3 shows the location of the four microphones on the NAO. With a combination of these four microphones, any sound can be triangulated to give the direction of the sound in the azimuth and elevation angles.

![Diagram of microphones on NAO head](image)

**Figure 3: Diagram of the location of microphones on the NAO head [3].**

The NAO also contains two video cameras. Figure 4 and Figure 5 show the locations and the range of vision for each of the cameras. Even between the two cameras, there are some blind spots, but this is inconsequential because the robot will be stationary on the table and the users are assumed to be somewhat slow to move. The users can leave and enter the robot’s field of vision, but they are assumed to do so at a slower speed.
While the robot has full range of motion of its head, arms, and legs, for simplicity the robot will not move. The NAO also contains various programmable light emitting diodes (LEDs) that are located on the head, eyes, chest, and feet. There are also force sensors in the feet of the NAO for balance, sonar sensors, joint position sensors, and contact and tile sensors [3].

There is a simple switch ON/OFF sensor on each foot. This can be used as a bumper sensor for when the robot is walking. There is a button on the chest of the NAO. When pressed, it will
speak, giving the IP address of the NAO and the state of the motors and battery. On the chest of the NAO, there are also two sonar sensors. There are also various internal sensors in the joints for actuation. There are force sensitive sensors in the feet, which can be used for programming the robot to walk. The eyes of the NAO are actually two infra-red sensors, one in the left eye and one in the right eye.

The NAO has an onboard processor, which is relatively slow for real time processing. The solution to this problem is to perform the processing off board, on a laptop. A Toshiba laptop was used to perform the processing and remotely control the robot. The laptop connects to the robot through the internet. Table 1 shows a comparison of the NAO’s onboard processor with the laptop’s processor. It is clear that the laptop will offer better processing speeds.

<table>
<thead>
<tr>
<th></th>
<th>NAO Processor</th>
<th>Laptop Processor</th>
</tr>
</thead>
<tbody>
<tr>
<td>Speed</td>
<td>1.6 GHz</td>
<td>2.50 GHz</td>
</tr>
<tr>
<td>RAM</td>
<td>1 GB</td>
<td>6 GB</td>
</tr>
</tbody>
</table>

**3.2: Software**

The program was written in Python 3.4. The Python language was chosen because of its fast and dynamic testing process. Python is an interpreted language. This means that the code is not compiled into an executable file. The code is compiled and executed at each line during run-time. Although Python has slow processing times because of this structure, the structure allows for quick development, prototyping, and testing. Other libraries were used to provide more specialized functionality such as robot control and in vision capabilities for face detection,
recognition, and training. These additional libraries contained available and well developed Python functions and APIs. Therefore, Python was a good language of choice.

Two other libraries were used in conjunction with the Python language to develop this project:

1) NAOqi
2) OpenCV

3.2.1: NAOqi

This is a specialized library of drivers to control the robot programmatically. The Python API for the NAO was well developed and well documented. Functionality of the library included APIs to control the robot: control of speakers, microphones, tactile sensors, cameras, LEDs, and motors in the legs, arms, fingers, and neck. There were other APIs that perform other functions such as: image capture, video streaming, sound localization, speech recognition, and an autonomous life module.

Below is a short summary of the NAO Python APIs that were used in this project:

1) **ALMemory**
   
The `getData` function in this module allows access to all sensor values and data that the robot collects and measures such as: position of joints, sound, and word recognition.

2) **ALTextToSpeech**
   
Functions in this module allow the robot to speak. The input is a string, and the output is either a .wav file of the robot’s enunciation of the phrase or the robot’s direct enunciation of the phrase using its speakers.
3) **AL Sensors**

This module allows subscription to the memory locations where the sensor data is stored. It also allows for setting sensor precision and sampling period.

4) **AL Video Device**

This module allows access to the video cameras that are on the robot. The module contains functions to start and stop the camera, as well as start and stop access to the video stream from the camera. The module also allows video recording, although this functionality is not necessary and not used in this implementation.

5) **AL LEDs**

Functions in this module allow control over LEDs in the eyes of the robot, the chest button and on the sides of the head by the loudspeakers. Specialized functions can change the color of the LEDs surrounding the eyes.

6) **AL Speech Recognition**

Functions in this module allow for the training of the speech recognition module on the NAO robot. A function is responsible for uploading a list of vocab to be recognized. Functions are also available for starting and stopping the speech recognition engine.

7) **AL Sound Localization**

Functions in this module all for sound localization. When a sound is detected the robot stores the azimuth, elevation, and level of energy of the sound in a memory location. This can be accessed by using the AL Memory API.
3.2.2: OpenCV

This is a library of functions for image processing. Face detection and recognition was used for identifying an individual in the conversation. Although there were some python APIs in the NAO robot language structure that are capable of performing face detection and facial recognition, the OpenCV library was used to perform these functions. The OpenCV library was chosen because it is specialized in face detection and facial recognition. This allows for more efficient and accurate processing. The two major functions used from the OpenCV library are:

1) `cv2.faceCasecade.detectMultiScale()`

   This function passes in tuning parameters and a black and white image frame. The function will return a list of coordinates where a face is detected. If there are no faces detected, the function will return a null.

2) `cv2.recognizer.train()`

   This function passes in a list of frames and a list of labels for the frames. Frames are associated with the corresponding labels in order. For example, the first frame in the list of frames is associated with the first label in list of labels. The function will train the recognizer based on the list of faces and the list of labels. The list of faces and labels will be associated with each other, so the correct labels must be put in the same order as the faces.

3) `cv2.recognizer.predict()`

   This function uses the previously defined and trained recognizer to predict the label for a given face. The output is a label and the percentage of confidence for that predicted label.
Other supporting functions that were used allowed access to the video stream on the webcam for testing purposes, drawing circles and rectangles to mark a recognized face and a face that needs training, displaying frames from the video stream, setting up the classifier for face detection, and setting up the filter for the face recognition.

3.3: Design

The complete project can be broken down into two main parts:

1. The perceptual capabilities: face detection, training, recognition, and tracking.
2. The control architecture: the FSM that tracks conversation and executes appropriate responses from the robot.

The perceptual capabilities consist of the components necessary for the architecture to function in a real scenario. This includes face detection in the current video frame and facial recognition on detected faces. A combination of these two elements will allow the robot to determine if a detected face is a new speaker or a returning speaker. Visual processing will allow the robot to determine who is participating in the conversation. A combination of visual processing and sound localization will be used to determine who is speaking. Speech recognition will be used to determine what was said and it will determine transitions between FSMs.

The architecture will be implemented using a FSM. Transitions between states are contingent on what the user says. For easy testing, the user and the robot will follow a script for the interaction. The architecture will include storage of past interactions. Each user that begins to interact with the robot will be assigned a model. This model will store everything the user has said and everything the robot has said to the user.
3.3.1: The Perceptual Capabilities: Face detection, training, recognition, and tracking

These capabilities were implemented using Python API functions from the OpenCV library. There are two stages to this process. In the first stage (face detection and recognition), the program waits for a face (or multiple faces) to appear in the frame and then processes and trains the faces. If no face is present, the program will wait until a face appears before further processing. If a face or multiple faces appear, this stage processes the frames assuming that there are no previous data. For each face detected, an instance of the face class is created and this instance is assigned attributes and an ID number. The program then trains on these faces and their corresponding labels. This stage initializes and sets up the face training for stage 2.

In stage 2 (face recognition and tracking), the program will continuously grab frames for the video stream. If no faces are found, the program will grab the next frame. If a face or multiple faces are found, then the program will run facial recognition on the detected face. The predicted label of the face is recorded in the program’s memory. Then the program checks the history of the faces that the robot has seen. Each face is recorded in the robot along with its label and centroid. If the centroid of the detected face is near the centroid of another known face, it is assumed that the detected face is the same face as the one historically close to that position. In this way, identification of the face is done through facial recognition and verified through localization. This increases the robustness of the algorithm for identifying a detected face. For each face that is recognized, the centroid of the face is calculated and recorded according the appropriate label of the class. For each face that is not detected in a frame, a location of (0,0) is recorded and the number of idle frames for the face is increased by one. This effectively records how long a face has been active and how long a face as been inactive.
Stage 1 is critical for the setup of the program. This is because the robot should wait for faces to appear before recording data. During stage 2, if no faces appear, all known faces are marked as idle faces for that frame. Therefore, in stage 2, even for frames with no faces, data is recorded. There is no reason for the robot to record idle data initially. Another difference between Stage 1 and Stage 2 is that there is no facial recognition in Stage 1.

3.3.2: The Control Architecture: State Machine and Conversation Tracking

The control architecture is responsible for tracking the conversation, maintaining the FSM that controls the responses of the robot, and recording the actions of participants in the conversation. The control architecture is programmed in Python 3.4. Supplemental libraries and Python APIs were used to remotely access the NAO robot through wifi, access the video stream from the robot’s cameras, control the robot’s actions, and allow the robot to speak and listen to the participants.

After initializing the program, the control architecture waits and listens for any speakers. Face detection is also run on every frame. Once a word is spoken, the architecture will run a speech recognition module to determine what was spoken. Then the program will use sound localization to determine where the sound originated and assign an ID for the speaker based on the location. The output of the face tracking module is also used in this control architecture to identify faces. Then, according to what was spoken, the robot responds accordingly.

The state machine controls the reaction and responses of the robot. Interaction between the robot and the user facilitate the transition between states in the FSM. For simple testing purposes, the robot and the user will follow a scripted conversation. The FSM is based on this
scripted conversation. There were two versions of the script, version A and version B that were implemented in two separate FSMs.

A combination of existing hardware and software languages and libraries were used to develop this architecture. The architecture is implemented on the NAO robot, which contained all the necessary hardware for receiving user input and sending the robot’s output. The NAO robot included microphones for receiving audio input from the user and speakers for responding to the user. The Python language was used to implement the architecture due to its dynamic properties and ease of use for prototyping. Supplemental libraries were used to provide additional functionality such as vision processing and robotic control of the NAO.
Chapter 4: Face Detection and Recognition

OpenCV libraries were used to implement the vision processing part of the architecture. The first step in this vision processing is face detection. Frontal face detection is performed using Haar Cascades. The OpenCV libraries offered a function with several turning parameters for face detection. Facial recognition is done using the Local Binary Patterns Histogram (LBPH) classifier method in OpenCV.

4.1: Face Detection

Face detection is a combination of object detection with a specific selection of feature detection. OpenCV has a classifier for detecting faces in an efficient and robust manner. This classifier can be trained to detect other object such as trains, airplanes, and cars. OpenCV also provides a specific classifier for detecting faces: the Haar Cascade classifier.

This classifier for face detection follows methods for feature and object detection detailed in [4]. In this thesis, faces are recognized in stages. Regions of a picture are processed individually. If the algorithm does not find the features in the first stage in a region of the picture, that region is discarded and not processed any further. If the region does contain the features in stage one, that region is marked for further processing. If at any point a region fails to contain features in a stage, the region is discarded. This allows the algorithm to heavily process regions with a high probability of containing the object, while discarding background regions that do not contain any interesting information.
4.2: Face Recognition

Once face detection is successful, the next step is to identify and recognize the face. Facial recognition is something that is taken for granted in everyday life. Humans of all ages are able to distinguish between familiar and unfamiliar faces. It is hard to quantify what is recognized in a face that makes the face ‘familiar’ to a person. It could be place and size of features on a face, or it could the face holistically. But at any rate, humans are very accurate at recognizing faces. Face can be recognized through changes in hairstyle, changes in facial hair, changes in lighting and environment. Humans can recognize faces from left and right side profile views of the face.

OpenCV has a library that attempts to quantify and implement some of how face recognition works. There are three different available algorithms in OpenCV for doing this [5]:

1) Eigenfaces
2) Fisherfaces
3) Local Binary Patterns Histograms.

In this project, the Local Binary Patterns Histograms method was used to recognize faces.

While the Eigenfaces and Fisherfaces methods use a holistic approach on high dimension data, the LBPH method does not. The first two methods work well and result in high accuracy predictions when there are many images in the training set. The methods do not work as well with smaller training sets and are susceptible to variations in the environment such as lighting differences. This does not make for a very robust algorithm. The LBPH method uses low dimensionality data for quicker more accurate training [7] and will need about 8 images for a good recognition rate.
The LBPH method extracts local features of an object and retains them to be used for facial recognition. This way, the data extracted is in low dimension and easier to store and work with. However, solely extracting local features leaves the data susceptible to properties such as scaling, translation, or rotations. Therefore, the LBPH extracts features and stores them by comparing each pixel with its neighbor. By noting whether the surrounding pixels are higher or lower in value than the pixel, the algorithm can effectively use this relationship to mark and compare data between different images. This solves the problem of having different lighting affect the results of face recognition. An example of how to compare a pixel with its neighbors is shown in Figure 6. The pixel’s value is 5. Values are compared in a 3 by 3 frame around the pixel. Values that are larger than or equal to the pixel value is marked as a 1 and values that are smaller than the pixel value is marked as 0. In this way, a binary code is generated [5].

![Thresholding example](image)

**Figure 6: How to threshold one pixel against its neighbors [5].**

### 4.3: Implementation

There are two stages that incorporate both face detection and facial recognition. To summarize, in the visual processing part of the architecture, the robot first initializes the face detection and facial recognition module. Then, the program proceeds to Stage 1 where it first waits for a face and loads initial data with initial assumptions. Stage 1 only processes one frame: the first frame that a face is detected. After Stage 1, the program trains the face
recognizer module on the faces detected in Stage 1. Proceeding to Stage 2, the architecture starts to continuously analyze, detect, identify, and train on the various faces that appear and disappear from the robot’s field of vision. Figure 7 shows the overall flowchart for the vision portion of the architecture.

Figure 7: Flow chart diagram representing the overview of the face detection and facial recognition portion of the architecture.

First, the program initializes and creates an instance of the face detection module and the facial recognition module. Then, the program proceeds to Stage 1 where it waits for one frame with a face or multiple faces. Then the program goes through the list of faces and decides if these are new faces or previously seen faces based on localization of the face. Each new face is assigned an instance of the Face class and a unique label. After the program is done processing all the faces in that single frame, the program runs training for the face recognizer on the list of faces and associated labels. Figure 8 shows a flowchart of Stage 1.
Figure 8: Flow chart diagram of Stage 1 of the face detection and facial recognition.
After the system is done training, the program proceeds to Stage 2 where it actively detects faces in each frame and predicts a label for the face. To determine the label of a given face more robustly, localization of the centroid of the face is used in conjunction with the facial recognition prediction to determine the label. The idea behind this is that a face cannot disappear immediately in the next frame and a face cannot be on one side of the frame in frame 1 and be far on the other side of the frame in frame 2, due to the fact that people cannot move that quickly. For any given face, the program checks a list of all active faces to see if the centroid of the given face is within the vicinity of a known face. If this is true, it is assumed that the given face is the known face and the given face is assigned the ID of the known face.

A face can disappear from the robot’s field of view. When this happens, an idle frame is recorded for that face. If there are 5 idle frames, the program marks this instance of a face as inactive. When searching through known faces for localization, the program will skip over inactive faces.

After a face is detected and identified with the correct label the program decides if there are enough training sets for faces of that label. If there are not enough training images, the program will append the face and the label to the training data and train the recognizer. If there are enough training images, the program will draw a green circle around the trained face and no data will be added to the training set and the recognizer will not be trained. Figure 9 shows the flow chart diagram for Stage 2.
The vision processing consisted of face detection and facial recognition and training. Both vision processing methods used an implementation from the OpenCV libraries. Face detection is done through Haar cascade classifiers and facial recognition is implemented using the LBPH method. The complete implementation of the vision processing module for this thesis consists of stage 1 and stage 2. In stage 1, the program assumes that all faces are new and trains the recognizer at the end of processing the first frame. Then the program enters stage two where it continuously detects, recognizes and trains faces that are in the robot’s field of vision.
Chapter 5: Control Architecture for Human-Robot Interaction

The control architecture is responsible for keeping track of the conversations between the robot and multiple participants. It tracks each individual’s interactions with the robot in a FSM. An instance of the FSM is created for each participant. Inputs from the participants facilitate transitions between states in FSM. The state of the user controls the response of the robot. This chapter presents the class objects and structure for the architecture to model conversation, the FSM that was used to facilitate and control conversation, and the overall architecture and modeling.

5.1: Class objects and structure

The architecture is designed to model a conversation with multiple people as efficiently and as simply as possible. The robot’s responses are also recorded. A class object: user, is created for each person as they join the conversation.

The user class members are:

1. **ID**: assigned number to identify the user
2. **State**: holds the current state of the conversation
3. **History**: one dimensional array that holds a concatenated string containing the timestamp and words exchanged between the robot and the user.

The functionality of the user class are:

1. Update the state of the user based on received input from the user.
2. Generate a timestamp and records input from the conversation.
3. Print out the entire history of the user at any given time.
The class object represents a unique person in the conversation and it is assigned an ID number at the start. Then, as the user interacts with the robot, the robot responds back to the user. The user’s responses are recorded with a timestamp in the class object as the robot hears the response. The robot’s responses are also recorded in the class object with the appropriate timestamp. As a result, each user’s interactions with the robot are modeled like a script for a play, with the words exchanged and their timestamp. The robot’s response to each person is recorded in that person’s model.

The robot also has its own class. The robot contains the state machine for interacting with the users.

The robots class members are:

1. **ID**: used to identify the robot.

The capabilities of the robot class are:

1. Respond to the user based on the state of the user and update the user’s history to reflect the robot’s responses.

The robot class is very similar to the user class in terms of architecture, but the functions are different because they are operating on two different sides of the conversation.

**5.2: The Finite State Machine**

The robot and user interaction will follow a scripted interaction for simple testing of the functionality of the program. There were two versions of the script used: version A and version B. The script for version A is shown in table 2 and the FSM that was developed from this script...
is presented in figure 10. The script for version B is slightly longer. It is shown in table 3. The FSM that was developed from this script is presented figure 11.

Table 2: Version A script for user and robot interaction.

<table>
<thead>
<tr>
<th>State</th>
<th>Speaker</th>
<th>Line</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;----&quot;</td>
<td>robot detects user</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>Robot</td>
<td>Hello!</td>
</tr>
<tr>
<td></td>
<td>User</td>
<td>Hi!</td>
</tr>
<tr>
<td>1</td>
<td>Robot</td>
<td>Hello there, what is your name?</td>
</tr>
<tr>
<td></td>
<td>User</td>
<td>My name is ___. What is your name?</td>
</tr>
<tr>
<td>2</td>
<td>Robot</td>
<td>My name is NAO.</td>
</tr>
<tr>
<td></td>
<td>User</td>
<td>Bye.</td>
</tr>
<tr>
<td>3</td>
<td>Robot</td>
<td>Good bye.</td>
</tr>
<tr>
<td>4</td>
<td>&quot;----&quot;</td>
<td>User is deactivated</td>
</tr>
</tbody>
</table>

Table 3: Version B script for user and robot interaction.

<table>
<thead>
<tr>
<th>State</th>
<th>Speaker</th>
<th>Line</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;----&quot;</td>
<td>robot detects user</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>Robot</td>
<td>Hello!</td>
</tr>
<tr>
<td></td>
<td>User</td>
<td>Hi!</td>
</tr>
<tr>
<td>1</td>
<td>Robot</td>
<td>Hello there, what is your name?</td>
</tr>
<tr>
<td></td>
<td>User</td>
<td>My name is ___. What is your name?</td>
</tr>
<tr>
<td>2</td>
<td>Robot</td>
<td>My name is NAO.</td>
</tr>
<tr>
<td></td>
<td>User</td>
<td>How are you?</td>
</tr>
<tr>
<td>3</td>
<td>Robot</td>
<td>I am doing well.</td>
</tr>
<tr>
<td></td>
<td>User</td>
<td>Bye.</td>
</tr>
<tr>
<td>4</td>
<td>Robot</td>
<td>Good bye.</td>
</tr>
<tr>
<td>5</td>
<td>&quot;----&quot;</td>
<td>User is deactivated</td>
</tr>
</tbody>
</table>

The robot responds to the participant based on the state of the participant. Interaction and speech from the participant facilitates the transitions between states. If there is no response from the user, or if the user gives an unanticipated response, there will be no transitions in the state machine. The robot remains in the same state.
More interesting interaction between the user and the robot and more complex responses from the robot can be programmed into this architecture easily by replacing the FSM. The architecture will still retain its ability to use vision and sound localization to track multiple people and still be able to track multiple individual conversations with various people. Two versions of the script were implemented in two FSMs.

Figure 10: Flow chart diagram of version A FSM.
5.3: Design and Functionality

The architecture will initialize the robot, face detection modules, speech detection and speech recognition modules. Once this is initialized, the robot will begin to listen for participants joining the conversation. The architecture marks a participant as joining the conversation when the participants greet the robot with a “hello” or “hi”. Once the robot hears these words, it will use sound localization to determine which user has spoken. If the sound localization determines that there is no previous user from that location, the architecture will mark this participant as a new participant and create a new user for it. A new user is created with a default state and the robot will respond according to the state that the user is in.
The architecture continuously listens for input from participants. When it receives input, the architecture will first determine if it is a new participant. If it is a new participant, the architecture will create an instance of the FSM and assign it to the new participant and responds to the participant according to the state of the participant. If the architecture determines that the participant is returning participant, the architecture will determine who it is, transition the state machine accordingly, and respond to the participant based on the new state that that participant is in.

When a participant leaves the conversation, the architecture marks the participant as inactive. The architecture will no longer update the state machine for the participant and the robot will no longer respond to the participant. Figure 12 shows a flow chart diagram of the control architecture.

![Control Architecture](image)

*Figure 12: Flow chart of the control architecture for the multi-threaded robot conversation with various people.*
The control architecture is responsible for the multi-threaded modeling of conversation and controlling the response of the robot. A FSM is used to control the conversation and allow the robot to respond correctly. Responses from the user facilitate transitions between states. The state of the FSM dictates the robot’s response. Different conversations can be programmed by designing a different FSM. There were two different FSM implemented in this thesis: version A and version B. They each follow a different, though similar, script. Each person who joins the conversation is assigned an instance of the user class. The user class contains the state machine for the conversation. The class also logs input from the user and responses from the robot to the user with an associated timestamp.
Chapter 6: Experimental Results

This chapter presents the results of the face recognition module and the control architecture module. A prototype was first designed, implemented, and tested on the laptop first, using keyboard inputs. Next, the architecture was implemented and tested on the NAO robot.

6.1 Face Recognition Results

During stage 2 of the vision training, the program is continuously looking for faces and localizing the known faces. New faces are added to the training set and the facial recognition module is trained until there are enough samples for the new face. In this test, the number of samples per face is 10. Consequently after 10 samples of a given face are trained in the recognizer, the system will not train on that face anymore.

New faces that have not been completely trained have a red circle drawn around the face. Faces that have completed the training have a green circle drawn around the face. Figure 13 shows a new face that has not been trained – the circle around the face is red. Figure 14 shows several frames later when the face is trained. There is a green circle around the trained face. Figure 15 shows the introduction of a new face. The program recognizes the first face as a face that has already been trained, and the program also recognizes that there is a new face that has not been trained. The new face is marked in red and the trained face is marked in green. The program will only train on the new face. Figure 16 shows a frame where both faces have been recognized and trained successfully.
Figure 13: New face that has not been trained. There is a red circle around the face.

Figure 14: Face that has been trained. There is a green circle around the face.
Figure 15: The program has trained one face and detects a new face. The trained face has a green circle and the new face has a red circle. The program is only training on the new face.

Figure 16: The program has detected and successfully trained two faces. Since the two faces are trained, the program will no longer train the recognizer.
6.2. Human-Robot Interaction Results

A prototype of the architecture was designed, implemented, and tested using a laptop. Then the architecture was implemented and tested on the NAO robot. Then, the architecture was implemented on the NAO robot using the NAOqi libraries to control the robot. Both version A and version B state machines were implemented and tested on the NAO robot.

A speech recognition engine was used to allow the robot to determine what the user has said. The architecture logs the phrases that were detected through the speech recognition. Due to natural variations in accents and tonal differences between the users, the speech detection engine is not 100% accurate. The most difficult phrase to recognize is “my name is ____”. This is the phrase that is responsible for transitioning from state 1 to state 2. Therefore, to make this transition, the robot only needs to recognize any combination of “my name is ____”. As seen in some of the examples, the robot will sometimes log the phrase “my name is” or “name” and still be able to successfully transition to the next state, continuing the conversation.

Table 4 shows a list of the scenarios which were tested.

<table>
<thead>
<tr>
<th>Scenario</th>
<th>Implemented on</th>
<th>Version of the Script</th>
<th>Number of participants</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Prototype: computer</td>
<td>Version B</td>
<td>3</td>
<td>Users join the conversation at different times</td>
</tr>
<tr>
<td>2</td>
<td>NAO Robot</td>
<td>Version A</td>
<td>1</td>
<td>one user completes the conversation without interruptions</td>
</tr>
<tr>
<td>3</td>
<td>NAO Robot</td>
<td>Version A</td>
<td>2</td>
<td>one user completes the conversation without interruptions before the next user joins</td>
</tr>
<tr>
<td>4</td>
<td>NAO Robot</td>
<td>Version A</td>
<td>2</td>
<td>one user joins the conversation, second user joins mid-conversation</td>
</tr>
<tr>
<td>5</td>
<td>NAO Robot</td>
<td>Version B</td>
<td>2</td>
<td>one user joins the conversation, second user joins mid-conversation</td>
</tr>
</tbody>
</table>

Scenario 1
First, a prototype architecture following the version B state machine was developed and simulated on the computer. The user input was in the form of text input from the keyboard.

Figure 17 shows test run with three users, each joining the conversation at different times.

Figure 17: Version B state machine test run on the prototype architecture during the computer simulation.

Scenario 2

Figure 18 shows the output of the test run with version A of FSM implemented. This shows one user interacting with the robot. The output is a timestamped log of the lines spoken by the robot and lines spoken by the user.
Scenario 3

Next, another user was added to the conversation after the first user had finished the conversation. Figure 19 shows the result of this. The timestamps show that user 1 completes the conversation before user 2 begins the conversation.
Figure 19: Version A of the FSM; two people have consecutively joined the conversation and interacted with the robot. One participant completes interaction before the next participant joins the conversation.
Scenario 4

For scenario 4, another test was performed with two people interacting with the robot. This time, one person joins the conversation initially. The second person joins the conversation in the middle of the first person’s conversation. Then, both participants interact with the robot to complete their conversation. Figure 20 shows the results of this test. The timestamps show user 1 greeting the robot first. Then mid conversation, user 2 joins the conversation. In the middle of user 2’s conversation, user 1 leaves the conversation. Then user 2 leaves the conversation. The robot successfully interacts with each user and responds correctly.

Figure 20: Version A; two people joining the conversation at different points.

Scenario 5
Finally, in scenario 5, version B of the FSM was implemented and tested with two users. Figure 21 shows these results. User 2 greets the robot and proceeds to interact with the robot. Mid conversation, user 1 appears and begins to greet the robot. The robot switches attention and interacts with user 1 until user 1 leaves. Then the robot interacts with user 2 until user 2 leaves.

The results of these tests on both the prototype implementation and the robot implement show that the architecture can successfully carry on simultaneous conversations with more than one person. The architecture is capable of successfully tracking faces using the vision module and
can respond correctly to the dynamics of human conversation: participants joining and leave the conversations, interrupting each other.
Chapter 7: Conclusion and Future Work

An important aspect of designing robots for human interaction is the way robots handle conversation with humans. While there has been research in the past that deal with conversational robotics, they are limited in their implementations and designs. One paper presented an architecture that allowed the robot to communicate with one user. Another paper presented a design for the robot to interact with multiple people, but the robot did not track individual conversations; it would track either the robot talking or the people talking.

This thesis presents a new architecture that will track individual conversations and allow the robot to carry on simultaneous conversations with multiple people. The architecture tracks each person as they join the conversation and interact with the robot. Face detection and facial recognition and training allow the robot to recognize each person and determine if there is a new person who joined the conversation. Sound localization was utilized to determine where the voice originated from. Speech detection was used to determine what the user has spoken. Interactions are recorded in the robot’s memory with an associated timestamp. The architecture was implemented in Python 3.4 and the NAOqi and OpenCV libraries were used to provide additional functionality including robot control and vision processing. Due to the use of libraries and a simple implementation on an existing robot, the architecture has the potential to be developed into an ‘off the shelf’ software.

Results show that the architecture successfully utilized vision processing to recognize and train faces in the robot’s field of vision. The architecture can successfully handle one person
conversations, two different people’s consecutive conversations, and two different people’s conversations simultaneously.

Future extensions of this project include designing a more complex FSM for more interesting conversations and testing of the architecture on different robotic platforms. Although two simple versions of the FSM were designed and implemented to follow two different scripts, they were of the same theme. Further development of more complex FSMs could involve FSMs to allow the robot to carry on different topics of conversation with different people. Although the architecture was tested on two different platforms: the NAO robot and the laptop, the architecture could be modified to run on different robots with the same capabilities as the NAO. Noting the ease of conversion between several robotic platforms can give a measurement of the potential for the architecture to be developed into an ‘off the shelf’ implementation to be adapted on many different platforms.

Overall, the architecture brings a new method for increasing the human-like abilities of a robot. It is capable of multi-threaded conversation with different people and it can handle interruptions in its conversation.
References


