Interactive Directed Exploration for Mobile Robots

by

Pantelis Elinas

BSc (Space and Communication Sciences) York University, 1999

A THESIS SUBMITTED IN PARTIAL FULFILLMENT OF
THE REQUIREMENTS FOR THE DEGREE OF
Master of Science

in
THE FACULTY OF GRADUATE STUDIES
(Department of Computer Science)

We accept this thesis as conforming
to the required standard

The University of British Columbia
May 2002
© Pantelis Elinas, 2002
In presenting this thesis in partial fulfilment of the requirements for an advanced degree at the University of British Columbia, I agree that the Library shall make it freely available for reference and study. I further agree that permission for extensive copying of this thesis for scholarly purposes may be granted by the head of my department or by his or her representatives. It is understood that copying or publication of this thesis for financial gain shall not be allowed without my written permission.

Department of Computer Science

The University of British Columbia
Vancouver, Canada

Date April 23, 2002
Abstract

A mobile robot is often required to function intelligently within a dynamic environment with only partial knowledge of that environment. The robot is usually equipped with an exploration module that allows it to increase its knowledge of its surroundings by exploring unknown regions but it is often difficult for the robot to decide which are the most interesting places to visit first. In our experience the robot spends too much of its time exploring uninteresting regions. Existing robot control systems allow a user to intervene using a GUI program accepting input through a mouse and keyboard. In this thesis we improve on the method of human-robot interaction by building a system for guiding the robot during the exploration phase, using the more natural means of speech and vision.

To build a robust system we chose to implement a behavior-based control architecture. The entire system consists of a large number of different behaviors (software modules) running on a number of different computers connected over a LAN. There are distinct behaviors for speech recognition, user finding, speech synthesis, sound localization, safe navigation, map building and high level reasoning. Using this interface, a person can get the robot’s attention by calling its name. He can then direct the robot to a location to explore using a combination of verbal commands and hand gestures. We will show how the developed system will provide us with a platform for future research in human-robot interaction and behavior-based robot controllers.
Contents

Abstract ii

Contents iii

List of Tables v

List of Figures vi

Acknowledgements viii

Dedication ix

1 Introduction 1

1.1 Motivation ........................................ 2

1.2 Objectives ......................................... 3

1.3 Thesis Contributions .............................. 5

1.4 Thesis Outline ..................................... 7

2 Previous Work 8

2.1 Robot Control Architectures ...................... 8

2.2 Finding People in Images ......................... 13

2.3 Sound Localization ............................... 16
List of Tables

3.1 The transition function $\delta$ for the Hors D'oeuvres serving robot . . . 28

5.1 The transition function $\delta$ for the Sound Localization Behavior . . . 41

5.2 Correspondence of angles to time-delay units for two microphones
separated by 0.25m. ................................................. 45

5.3 The measured localization error ........................................... 49

5.4 The transition function $\delta$ for the User Finding behavior .............. 51

5.5 The transition function $\delta$ for the User Tracking behavior .......... 66

5.6 The transition function $\delta$ for the Navigation behavior ............. 80

5.7 The transition function $\delta$ for the Supervising behavior .......... 86
List of Figures

1.1 Example of a person visually guiding a robot ........................................... 2
2.1 The Sense-Map-Plan-Act architecture ..................................................... 9
2.2 The Subsumption architecture ................................................................. 10
2.3 The RGB and HSV color spaces .............................................................. 14
2.4 Occupancy grid map of an indoor environment ........................................ 19
2.5 The Triclops trinocular stereo system ...................................................... 20
2.6 Sample stereo image ................................................................................. 21
2.7 Left: Raw image of face and Right: Edges detected .................................. 24
3.1 The SR diagram for a simple generic behavior [Ark98] ............................. 27
3.2 The FSA diagram for an Hors D'oeuvres serving robot ............................. 28
4.1 Eric the robot ............................................................................................. 31
4.2 Microphone hardware setup ...................................................................... 33
5.1 The Speech Recognition behavior Stimulus-Response diagram .............. 35
5.2 The structure of a simple rule in the Java Speech Grammar Specification 36
5.3 The Sound Localization behavior Stimulus-Response diagram ............... 40
5.4 The Sound Localization behavior FSA diagram ....................................... 41
Acknowledgements

First, I want to express my most sincere thanks to my supervisor Dr. James J. Little for his valuable guidance and discussions as this thesis grew from an abstract idea to a useful system.

Second, credit is due to my parents who encouraged me to enter graduate school and pursue an academic career.

I want to also thank several more people in the robotics group. Jesse Hoey for helping me establish a theoretical basis for parts of my work. Don Murray for helping me understand the low level code that controls the robot and its sensors and for keeping all the hardware working as the deadlines approached fast. Luc Dierckx for keeping the robot’s Operating System up to date and for installing all new software I needed for my work. Bruce Dow for helping with the installation of new sensors on-board the robot and for keeping the robot’s hardware operational at all times. Dr. David Lowe for being the second reader for this thesis. The people of Wavemakers Inc. for supplying some of the code that I used for sound localization.

Finally, I want to thank all of the faculty and students in the Laboratory for Computational Intelligence for making the last couple of years a most rewarding experience both educationally and personally.

PANTELIS ELINAS

The University of British Columbia
May 2002
“Each problem that I solved became a rule which served afterwards to solve other problems.”

- Rene Descartes (1596-1650), "Discours de la Methode"
Chapter 1

Introduction

In the following chapters, we will describe the development of a behavior-based robot control architecture with application to user directed exploration for our mobile robots. A behavior-based architecture [Ark98] is based on building stand-alone software modules, called behaviors, that perform a specific task while having very carefully specified input(s) and output(s). The input(s) can be sensor data, i.e., an image, sonar data or a disparity map, and the output(s) is a string of bytes that has a unique interpretation for each behavior. A robot can achieve a high level goal, i.e., delivering mail to a specific location in a building, by dynamically selecting the correct subset, or the entire set, of behaviors to use. The selection of behaviors is either facilitated by another behavior, the supervisor, or algorithms are introduced that allow each behavior to autonomously decide when it needs to be activated.

In what follows, we will present a system that allows a person to interact with the robot in a natural way using vision and speech (see Figure 1.1). This thesis is not about developing a new robot-control architecture but rather using an existing architecture to solve a particular problem, while at the same time providing us with a platform for future research. With this interface, a person using speech and gestures
Figure 1.1: Example of a person visually guiding a robot. The person is pointing to an interesting location or object in the room.

can direct a mobile robot to a specific location to explore as the robot navigates to different locations collecting information about its surrounding environment.

1.1 Motivation

We want to create robots that perform robustly over long periods of time and can be easily used by the average person. People are accustomed to communicating with others using speech (audio signal) and gestures (visual signal.) There is a vast amount of literature on the difficult task of improving Human-Computer Interaction. The mouse and keyboard as input devices to GUI (Graphical User Interface) programs are adequate for use by well trained computer professionals and the younger generations that have been exposed to computer technology since their early child-
hood. Unfortunately, for older, disabled or non-technical people, computers are still a source of a lot of confusion. It is often too costly to train someone to properly use computer software, especially given the difficulty of technical people to communicating with those not familiar with technical terminology.

Another advantage is that using speech and vision as an interface to a robot allows a person to use the robot without him being tethered to a workstation. The computer and robot could be in different rooms limiting the person’s ability to monitor the exact state of the robot making it easier to give it incorrect commands.

Human-robot interaction (or HRI for short) is a form of HCI in which the body of the computer is represented as a mobile machine that sometimes looks like a human being (called an android.) One wishes to be able to communicate a task to a robot by using speech and hand/arm gestures.

1.2 Objectives

Building a robotic system capable of fluent and transparent communication with humans is a huge task requiring an interdisciplinary team of scientists and thousands of man hours, as previous attempts by teams in many research labs have shown. Our goal is to build a first system that can become the building blocks of larger and more complete systems in the future. This makes the choice of a behavior-based control architecture obvious as the latter has been proven to satisfy the requirements of extendability and robustness [Ark98].

So, one of the main goals is to produce or better extend the existing set of available robot behaviors. A system build around a behavior-based control architecture can be extended by developing and adding to it even more behaviors later on.
To test the behaviors developed, we had to choose a specific application domain. We chose to develop a human-robot interface based on vision and speech that allows a person to guide a mobile robot during an exploration phase. During exploration, a robot is placed in an unknown environment and it is asked to build a map of it by selecting the regions to visit based on some given constraints. Such constraints include using the least amount of energy and exploring the most interesting regions first. Using our interface, one can call on the robot's attention by calling its name and direct it to explore a specific area using speech and hand gestures. The robot can enter a brief dialog with the person to verify that all it has understood is correct, therefore, making the system not only useful to the average person but also fun to use. An example of the verbal interaction is (Eric is the name of the robot):

Person: (loudly) Eric!
Eric : (stops motion and turns towards the person)
Person: Find me!
Eric : I can see you. You are 4 meters away and 1.6 meters high.
Is this correct?
Person: Yes!
Eric : Confirmed!
Person: (Points with finger) Go there!
Eric : You are pointing 2 meters ahead and 4 meters to my left.
Do you want me to go there?
Person: Yes!
Eric : I am on my way! (Eric moves to that location and enters exploration mode)

The dialog includes a number of confirmation steps to verify that information
the robot has computed is valid. Safety of the robot and the people around it is of the utmost importance which makes the latter necessary. The language is still simple and intuitive enough that the average person can learn and later remember with ease as opposed to learning how to use a more complicated GUI program. One will immediately notice that some of the language the robot uses has too much detail such as giving the location in meters. This language is rather awkward, not the way people talk among themselves. Later we hope to improve on the language used so that the robot can describe the location using topological information of the objects in the room. This requires the ability for the robot to perform object recognition and localization, two areas that we did not have time to explore as part of this thesis.

1.3 Thesis Contributions

The main contribution of this thesis is that for the first time we can interact with our robot without having to explicitly make use of a workstation. For the first time, the robots can hear, see and speak all at the same time, in real-time. We have also developed a system that runs on more than two computers, harnessing some of the computing power available and overcoming the limitations of the robot's onboard computer. Also, we now have added three more additional sensors onboard the robot, one vision and two audio. Finally, we have increased the number of available behaviors and enhanced some of the existing ones providing us with more building blocks for future research.

Over the years a number of behaviors were developed for our robots and they have been used to construct other systems such as Spinoza [MJ97], a robot that was capable of tracking a colored ball and following it around. Some of these behaviors
are:

- a mapper (a process that uses stereo information to build an occupancy grid map of obstacles in the environment,) and

- a command executer (a process that receives high level motion commands and interfaces with the robot's hardware to execute them.)

We have added a number of new behaviors including:

- a speech recognizer (a process that performs speech recognition based on a command-and-control grammar)

- a user finder (a process that can find a person in the robot's camera field of view)

- a sound source direction localizer (a process that can compute the direction of the source of sound incoming to the robot)

- a hand gesture recognizer (a process that can compute where a user is pointing on the floor)

- a path planner and follower with dynamic obstacle collision avoidance

- a high level Artificial Intelligence supervisor for high level reasoning and control of the available set of behaviors listed.

Each of the newly developed behaviors takes advantage of all the available sensory information collected with the on-board sensors, to compute its results reliably and in real-time. Satisfying these two constraints is not an easy task but our system does a good job of using straightforward algorithms that take little time to compute and yet all together construct a robust system.
1.4 Thesis Outline

Chapter 2 presents an overview of related work in robot control architectures, finding and tracking people in images, skin color segmentation, sound localization and collision free robot navigation in dynamic environments.

Chapter 3 describes the notation that we use, to describe a behavior's input(s), output(s) and internal algorithms.

Chapter 4 describes the hardware we used.

Chapter 5 presents how each of the newly developed behaviors work. We will use the notation as presented in Chapter 3.

Chapter 6 presents the conclusions of our experiments and we present the necessary additions to the system for further improvement.
Chapter 2

Previous Work

In the following sections we describe previous work in building intelligent robotic systems. We start by describing the most commonly used robot control architectures: purely reactive, purely deliberative or hybrids of these two. The control architecture provides the backbone for the entire system. We focus on the behavior-based approaches that have been proven to be the more robust. We then continue with a description of other researcher's solutions to the specific problems that each behavior in our system tries to solve. We review work in the areas of finding and tracking faces/heads, sound localization, human-robot interaction and obstacle avoidance for navigation in dynamic environments.

2.1 Robot Control Architectures

There are three major robot control architectures: reactive, deliberative and hybrids of these two [Ark98, Alb92, Fer94, Tso97].

Early robotic systems were designed to be purely deliberative as a plan had to be generated for each action performed [Mor77, Nil69]. In such a system, during
Figure 2.1: The Sense-Map-Plan-Act architecture [Ark98]

Each cycle, sensor data are first collected and used to build a map of the environment; these are the Sense and Map states, respectively. Then, using the map built previously, a plan of what action to take next is devised; this is the Plan state. Finally, the plan is executed by mapping the plan to robot motion; this is the Act state. Figure 2.1 shows a diagram of the Sense-Map-Plan-Act architecture [Ark98].

One of the major drawbacks of the SMPA architecture is that a good model of the environment must be known at any one time in order for a good plan to be derived. A good plan is one that will solve the task handed to the robot without danger to people, or to the robot itself. While in the SMPA cycle, the robot is incapable of sensing changes to the environment that have happened since it exited from the Sense state. As a result the chances of it selecting a bad plan increase as it enters a non-static and highly dynamic environment. On the other hand, the SMPA architecture maps nicely to functional programming practices that are well known. Therefore, the software development effort of driving a robot using the SMPA architecture, can be significantly minimized.
In the middle 80s a reactive control architecture that is based on principles that are the extreme opposite of the principles of deliberative control was introduced [Bro86]. Robot control is achieved using a number of different modules that run in parallel as opposed to in sequence as in SMPA. The new architecture is called Subsumption and it falls under the more general label of behavior-based robot control. Figure 2.2 shows a diagram of a general Subsumption architecture. Each module is referred to as a behavior. Intelligence is achieved through the interaction of the robot with the environment and it is not a property of either. No explicit model of the world is necessary which is what makes Subsumption so different from SMPA.

Each behavior is a pair of stimulus (or stimuli) and response(s). Control is achieved by selecting the subset of behaviors necessary to complete a task. Reactive architectures are based on the two principles of Situadetness and Embodiment. The former requires that the robot exists within a real environment and the latter requires that the robot has a physical body as opposed to being just a computer simulation.

One of the drawbacks of Subsumption is that it is very difficult to assign
a high level task to it [HP91]. The selection of which behavior(s) are supposed to be active at any one time is either facilitated through a supervising module or algorithms are introduced that allow the behaviors to resolve conflicts among the group. In general conflict resolution is priority based and competitive [Fer94]. The most important behaviors i.e., those that should be activated when a highly dynamic event occurs that puts the robot in danger, are usually assigned the highest priority and are hardwired to the robot’s actuators. One advantage is that, through modular design, Subsumption enables researchers to build robots that can easily be extended simply by adding new behaviors. Additionally, each behavior can be tested and perfected on its own before add to the set of behaviors, hence adding to the robustness of the whole system. Finally, since there is no need for a detail model of the world, Subsumption tends to be more robust than SMPA in highly dynamic environments [Bro86, Ark98].

The drawbacks and successes of the deliberative and reactive control architectures have led to control systems that are combinations of both. Hybrid control systems attempt to combine the benefits of both Subsumption and SMPA while minimizing their drawbacks. An example of such an architecture is Blackboard [HR85] introduced in the mid-80s.

The Blackboard architecture is named on the the metaphor of a number of experts that gather around a blackboard to solve a problem. Such a system is organized as independent modules all writing their output(s) to a common place (the blackboard) in shared system memory. There are two different types of modules. One set is purely reactive and it is set up to process sensor data and store it to the shared memory. The second set of modules are a set of experts that operated on the information stored in the Blackboard and they are purely deliberative. If
an expert realizes that it can contribute to solving the problem at hand, it asks for
the attention of a supervising module. The supervisor decides which of the experts
can contribute the most to solving the problem and gives it permission to run.
Since some of the modules are purely reactive and others are purely deliberative,
Blackboard is a hybrid architecture.

The S* proposal [Tso97] is also an attempt to a hybrid system. S* is an
extension to the subsumption framework. The system is build with a number of
behaviors operating on an SMPA-W (Sense-Map-Plan-Act-World) cycle. Unlike
the original Subsumption, S* includes a model of the world. The world consists not
only of the physical environment but also of all the internal state of the robot and
any of the outputs of the running behaviors. The application domain for S* is that
of Active Vision but the architecture applies to robot control as well. It also provides
mathematical methodology for proving whether a set of behaviors can solve the task
at hand within time and computational constraints that might be enforced with a
task specification. Unfortunately, no real system has been implemented using S* so
we don’t have any real data to support any claims that it can perform better than
any other approach.

The control architecture that we will be presenting belongs to the hybrid
category. It consists of a set of purely reactive behaviors and a single deliberative
behavior that supervises and controls them. The behaviors share data in two ways.
Data are either shared similarly to the Blackboard architecture using shared system
memory or through the supervising behavior by sending messages using system
sockets. This allows to distinguish between low level robot control and high level
goal-oriented planning and it makes the entire system easily extendable.
2.2 Finding People in Images

The problem of finding people’s faces in images has received a lot of attention by the machine vision community for many reasons. HCI can be greatly improved when a person’s face can be observed over time and emotions are derived from its expressions; thus allowing computer professionals to provide a more personalized experience to computer users. Face recognition has also been used widely in user identification for security and surveillance applications [Ess99, BA00].

Unfortunately, finding faces in images is not an easy task and many different algorithms of varying success have been proposed. Generally, the method used depends on the input data available. Input data can be gray scale images, color images, 3D shape information, all of the above or any other combination of them.

When the input data is just gray scale image information one of the oldest methods employed for face finding is correlation template matching of image regions against a database of known face images [JKS95, BP93]. This method yields acceptable results for a small database of people. On the other hand, it tends to fail as the number of templates increases because of the increased probability that two people look similar. Another advantage of this method is that it also allows the identification of a person since each template corresponds to a distinct individual. Template matching is usually augmented with other techniques that look for specific facial features such as eyes, mouth and nose, that satisfy certain obvious geometric constraints such as that the two eyes and mouth form a triangle on a person’s face [BP92].

When color information is available then most methods start by selecting candidate face regions in the image driven by skin color segmentation, [HAMJ]. Most images obtained with digital cameras store color information using the Red-
Figure 2.3: (a) The RGB and (b) the HSV color space [FvDFH96]
Green-Blue (RGB) color space shown in part (a) of Figure 2.3; using RGB notation, each color at a pixel is represented as a combination of the three colors red, green and blue. Not all colors can be represented using RGB values but the scheme is widely used as it can represent enough colors to please the human eye. Most commonly, color image processing is performed in the Hue-Saturation-Value (HSV) domain, shown in part (b) of Figure 2.3. HSV is often referred to as the chromaticity domain from the Greek word “chroma” that stands for color; the name is used because the HSV color space nicely groups the same colors into the same family even if they differ in intensity. So, if we are searching a set of images taken at varying amounts of illumination for the same light source, for the red hue, the H value should be almost constant and the difference in intensity will show in the V parameter. Images can be transformed from RGB to HSV and vice versa using a simple algorithm. For a detailed description of these two and other color spaces the reader is directed to read [FvDFH96].

Skin color has been proven to form a tight cluster in the chromaticity domain, [TA99, TMS98]. Unfortunately, skin surfaces are highly specular and most segmentation algorithms are very sensitive to changes in illumination. The result is the detection of a lot of false positives, pixels that are selected as skin even though they are not. Color constancy algorithms [FBM98] attempt to map the color values of an image to a space other than RGB or HSV that is invariant to illumination. Unfortunately the results presented in [FBM98] show that there is no guarantee that applying a color constancy algorithm will in any case give significant improvements to the performance of any color-based algorithm. In general, the system needs to be re-calibrated for changes in illumination.

Finally, if 3D image information is available then it can be used for detecting
faces as shown in [EHL+02]. In general, stereo vision systems designed to run in real-time onboard a robot are not accurate enough to be used as the only cue for face detection.

The best performing algorithms are usually those that try to use more than one of the above techniques together [SP96].

2.3 Sound Localization

Driven by our observations of human auditory perception and its uses, we realize that a complete interface for HRI requires the capability of sound localization. Sound localization is concerned with the problem of determining the 3D location of sound sources. In humans, studies show that there are three major cues that contribute to sound localization, [RM99]. In frequencies below 1KHz, Interaural Time Difference is the significant cue. Interaural refers to the difference in the sound signal arriving to the two ears. In frequencies above 4KHz Interaural Intensity Difference is most important. Lastly, spectral cues are known to augment the latter two. Spectral changes in the incoming sound waves are the result of the structure of the outer ear, head and torso.

Sound localization for computers is usually performed using an array of two or more microphones in a carefully set configuration. In the simplest case, we can use two microphones for determining the direction of a sound source on a plane to within an acceptable accuracy. We need at least three microphones in order to determine the direction to the sound source in 3D; we need more than three in order to improve the accuracy. If we also want to extract the exact 3D position of the sound source then we need at least four microphones.

Sound localization is often used to steer a camera system towards a speaker
in a conference room. Some researchers have used a large number of microphones separated by large distances, over half a meter, [RRD+96], with good results in a reverberant environment. They focused on extracting the 3D position of the speaker by computing the Interaural Time Difference of the signal arriving to each microphone. The problem with this approach is that the large separation distance among the microphones and the large number of microphones needed makes it unapplicable to use on a mobile robot. Another approach,[DPM96, ZC93], trains a Neural Network with inputs of Interaural Time Difference and Spectral cues to determine the sound source, an approach that most closely mimics the human auditory system.

One of the major difficulties of using such a system on a mobile robot is the large number of microphones and separation distance among them necessary for accurate results. As we mentioned earlier, we need at least three microphones to determine the 3D direction to the sound source. In [RM99] a system of two microphones is presented that is capable of extracting this 3D direction vector using two non-stationary microphones. The system simulates the existence of more microphones by making a decision as to how to move the microphone array to capture the incoming sound signal from different locations as if more microphones existed. The results they achieve are encouraging considering the low cost of the system. Unfortunately, this setup cannot work with impulsive sounds that do not exist long enough for the array to move and listen again.

We will present a sound localization system that uses three microphones to get a rough estimate of the 3D direction vector to a sound source. In our case, one of the main challenges to accurate localization is the amount of noise introduced by the robot hardware. So, we present simple techniques for filtering the noise and performing selective localization, i.e., localizing only on signals that are less likely
to be noise coming from the robot.

### 2.4 Obstacle Avoidance

Mobile robots are required to navigate safely in dynamic environments. Safe navigation requires that the robot move to a goal location without causing damage to itself or other objects in the environment including furniture, people and other robots.

If an exact map of the environment is available then safe navigation is simple and it reduces to finding an unobstructed path from the robot’s current location to the goal location. Unfortunately, such a precise map that is valid even over short periods of time is next to impossible to obtain. The problem of mapping is coupled with the problem of accurate robot localization. It is easy to create maps if the robot knows its exact location at all times so that new sensor data can be fused into the map correctly. To know the exact location of the robot, an accurate map is needed. Hence, making the problem of mapping and localization that of the chicken and the egg. As a result, only a partial map is usually available. Such a map is often stored as a 2D occupancy grid, [ME85, Elf89], of free, occupied and unknown space. Each cell in the grid represents an NxM area in the environment. An example of such a map can be seen in Figure 2.4.

In addition, dynamic environments change continuously and so any map constructed, no matter how accurately, quickly becomes obsolete and insufficient for safe navigation. Finally, once again as a result of the dynamic nature of the environment, obstacle detection and avoidance must be performed in real-time requiring the use of a sensor that can collect data and use them to build models of the environment in real-time. Such a sensor must return the distance to nearby objects that are potential hazards for the robot.
There are a number of such distance sensors. One kind uses sound to detect distance to objects. An ultrasonic (above 20000 Hz) sound is first fired, from the sensor, into the environment. Then the sensor listens for the same sound returning which would be the case if it bounces of an object nearby. Since the speed of sound in the air is approximately well known, the time it takes for the sound wave to return can be used to compute the distance to the object. A robot is usually equipped with an array of sonar sensors that provide 360 degrees of coverage. Sonar sensors are not perfect and their accuracy depends on the material of the object's surface, the distance of the obstacle and the surface properties of other nearby objects such as the floor and walls.

Sonar is highly inaccurate for material that absorb the transmitted sound waves. Such objects are seldom detected by sonar although algorithms have been designed to partly handle this problem [Kon97]. Also, if the object's surface is angled away from the sensor then the measurement will be highly inaccurate as the
sound waves bounce off the object away from the sensor. As a result the object is not detected. In general sonar sensors are the cheapest to buy and if their error is modelled appropriately and if some assumptions about the nature of the material properties of objects in the environment hold, they can be used adequately for navigation.

Similar to sonar are laser and infrared sensors that operate using light waves at different regions of the light spectrum, but unfortunately suffer from similar drawbacks as sonar.

Another distance sensor is inspired by the human visual system, [MJ97, TSM+97], and it computes distance to objects visually. Such a sensor uses two or more cameras to capture images simultaneously from slightly different viewpoints. It then uses these images to produce stereo maps by matching the same pixels in the images, [Fua93]. A minimum of two cameras is necessary for stereo computation but for robustness more than two cameras are most often used. The Triclops system by Point Grey Research is one such stereo system, shown in figure 2.5. It uses three cameras, and so it is called a trinocular stereo system. The third camera is used to resolve ambiguities in images that a two camera system cannot handle. Triclops
Figure 2.6: (a) Gray scale image from the Right camera of the Triclops shown in Figure 2.5 and (b) the stereo image corresponding to (a) where darker pixels indicate objects closer to the camera. An intensity values of 0 and 255, i.e., black and white, denote pixels where stereo computation was invalid either because the object is too far away from the sensor for stereo computation or because the object has insufficient texture for pixel matching.

provides stereo maps in real-time which makes it useful for robot navigation. Part (a) of Figure 2.6 shows a gray scale image, from one of the three cameras, while part (b) of the same figure shows the computed stereo map. One of the problems with this type of distance sensor is that it requires a relatively large amount of computation that is often hard to find onboard the limited resources of a mobile robot. Luckily, the Triclops system paired with a new generation CPU is capable of generating stereo maps in over 10fps without exhausting all the onboard computer's resources. Finally, just like all other sensors, visual sensors do not always provide reliable data as, for example, it can easily fail if there is very little variety in the texture of the visible object's surface, i.e., a wall that is uniformly colored.

Needless to say, in a dynamic environment such as a laboratory inhabited by people, no matter the knowledge of a map, unknown obstacles can occur at any time, i.e., a person walking in front of the robot or a chair that has been moved...
since the map was built. We need to use one of the above sensors (the visual sensor is our preferred one) to detect these objects and avoid them. Most approaches try to navigate around such dynamically detected objects but we will show how we first try to clear the path by acting on the environment before reacting to it. So, if the former fails then we react by navigating around the obstacle if a clear path exists.

2.5 Face Tracking

Tracking in machine vision is the problem of identifying the same set of image features in a temporal sequence of images. Tracking has received wide attention, over the years, by the computer vision community and many different algorithms of varied success have been proposed.

Face tracking is the specific problem of tracking one or more person's faces over time in a sequence of images. One can apply general probabilistic tracking algorithms like EM [Bil97] and Mean Shift [CRM00, CM98] or algorithms that attempt to track the distinct features of a face such as the skin color and geometric shape. The former techniques are the most complicated to understand since they are solutions to the general tracking problem and we will not discuss them any further. These techniques can be split into a number of different categories. For the descriptions that follow, we are interested in the method of tracking the face and head assuming that their location in the first image is known perhaps after applying a face finding algorithm as described in an earlier section. Also, many tracking algorithms preprocess the images to remove the background in an attempt to reduce the amount of noise, i.e., uninteresting to the tracking problem regions.

First, some researchers have investigated the option of tracking faces by searching for skin colored regions in color images [FT97, SI96]. Such techniques
often fail because of the color constancy problem discussed earlier. However, we
will see later that skin color data, when available, can become useful information to
improving the performance of other methods.

A second approach to face tracking is one based on face template matching,
[HB96]. Template matching techniques can be successful if the person presents a
view that matches the available template. They are not usually robust to rotation
of the face out of the image plane, i.e., the track is lost if the person presents a side
view. The track can also be lost if the scale of the face in the images changes because
for example the person moved closer or further away from the camera. Finally, there
is no such thing as a generic face template that could fit all people which means
that a template for every person that needs to be tracked must be made available,
increasing the resource requirements for this algorithm and diminishing its generality
and usefulness.

Another method tries to use depth and motion information to track heads,
[NTH94, MG96]. It makes sense that a person would move smoothly in 3D space
which makes tracking using depth information a obvious choice. Motion tracking
analyzes successive images and extracts motion at pixel accuracy of the objects
visible; that is the results is an image that assigns to each pixel a vector that shows
the direction of motion of each pixel in image space. Intuitively, all pixels that
correspond to a face will move in the same direction smoothly. The downside to
using motion for tracking is that it is very computationally expensive to compute
it robustly, especially for fast moving objects, hence making it a bad choice for a
real-time system.

A third approach to face tracking attempts to track the unique contours
traced by a face in an edge map, [BH94, RMG98]. Edge images are generated by
computing the intensity-based directional derivatives at each pixel location [TP86, Can86, MH80]. An example of the edges detected given the gray image of a face is shown in figure 2.7. Usually the images are convolved with a Gaussian mask to remove Gaussian noise. Contour based trackers are very sensitive to background clutter as many unwanted edges can be introduced when not desirable. Unfortunately, there is no edge detector that can be configured to perform ideally at all times [JKS95].

At this point one would expect that it might be possible to salvage the best qualities of the methods described above to build a better face/head tracker. A very simple method that is a hybrid of contour and color-based tracking and performs robustly in real-time is presented in [Bir97, Bir98]. This method starts by fitting an ellipse to the last know location and size of the head. The choice of an ellipse is obvious considering the elliptical nature of a person's head. Then, it computes two terms, over the ellipse, that characterize a head. The first term is based on the color of the person's head, not only the skin color of the face but also including the color of the eyes, mouth and hair. As a result, the algorithm remains robust when the side or back of the head is the part visible to the camera. The second term is a
measure of the derivatives of the outline of the head in the edge map. Together the two terms complement each other and provide robust tracking even under temporary occlusion. At each step the algorithm searches a small neighborhood around the last known head location for the head's new location. It can remain successful even if the person is moving closer or further away from the camera by additionally searching over a small range of possible ellipses both larger (i.e., the person has moved closer) and smaller (i.e., the person has moved further away) than the last known size.

We present the use of a head tracker similar to the one described last. The only exception is that we will use stereo information to track the face in 3D space to avoid superfluous searching over different ellipse sizes and hence reducing the amount of computation necessary.
Chapter 3

Notation for Behavior

Description

In the following chapters, we will need to describe the functionality and organization of a number of different behaviors. For readability, we will use a uniform notation for behavior presentation, as described in [Ark98]. We will employ the use of Stimulus-Response (SR) and Finite-State-Acceptor (FSA) diagrams to describe each behavior. Next we describe each of the two methods.

Stimulus-response diagrams are simple and intuitive. We have already described how a behavior has specific input(s) and output(s). An SR diagram is simply a way to graphically present the general structure of a behavior without going into the details of how the behavior operates on the input(s) to produce the output(s). Figure 3.1 shows the SR diagram of a generic behavior. Finite state acceptor diagrams are used to describe both how an individual behavior works and how a set of behaviors interact with each other. Usually, an individual behavior is a simple SMPA and the FSA describing it is straightforward to construct. More formally,
an FSA is specified by a quadruple \((Q, \delta, q_0, F)\) where \(Q\) represents the set of valid states; \(\delta\) is a transition function mapping the transition from one state to another or the same; \(q_0\) is the starting state and \(F\) is the set of final, accepting states. FSAs are usually represented in graphical form accompanied by a table that describes the function \(\delta\).

In the following chapters, we use FSAs to describe the behaviors and their relationships. So, in order to make the notation better understood, we will provide a simple example of an FSA describing the operation of a set of behaviors designed for enabling a robot to serve Hors D'oeuvres to a large crowd, during a reception.

There are a number of behaviors that must be invoked to complete the complex task of serving Hors D'oeuvres to a large group of people. The most important behavior for such a task is the navigator that allows the robot to avoid crashing into people or furniture. The navigator is only one of several behaviors necessary. Additionally, we need a behavior for detecting when the serving tray is empty; one for the robot finding its way back to the serving table to refill its tray, if the latter is empty; and finally, one for speech recognition, and one for speech synthesis for interacting with the people. In general the system is in one of five states:

- a “starting” state when the robot is at the serving table refilling its tray and deciding on a location where there are people to go to and serve
- a “moving” state when the robot is trying to get to that location
Table 3.1: The transition function $\delta$ for the Hors D’oeuvres serving robot

<table>
<thead>
<tr>
<th>$\delta$</th>
<th>q</th>
<th>input</th>
<th>$\delta(q, input)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>start</td>
<td>has-food-in-tray and has-target-location</td>
<td>to-serve-location</td>
<td></td>
</tr>
<tr>
<td>start</td>
<td>no-more-food</td>
<td>Done</td>
<td></td>
</tr>
<tr>
<td>to-serve-location</td>
<td>at-target-location</td>
<td>serving</td>
<td></td>
</tr>
<tr>
<td>to-serve-location</td>
<td>no-food-in-tray</td>
<td>go-to-base</td>
<td></td>
</tr>
<tr>
<td>to-serve-location</td>
<td>not-at-target-location and has-food</td>
<td>to-serve-location</td>
<td></td>
</tr>
<tr>
<td>serving</td>
<td>has-food-in-tray and has-target-location</td>
<td>to-serve-location</td>
<td></td>
</tr>
<tr>
<td>serving</td>
<td>no-food-in-tray</td>
<td>to-serve-location</td>
<td></td>
</tr>
<tr>
<td>go-to-base</td>
<td>reached-base</td>
<td>go-to-base</td>
<td></td>
</tr>
<tr>
<td>go-to-base</td>
<td>not-at-base</td>
<td>go-to-base</td>
<td></td>
</tr>
</tbody>
</table>

Figure 3.2: The FSA diagram for an Hors D’oeuvres serving robot

- a “going back to base” state when the tray is empty and the robot must return to the serving table to get some more food
- a “serving” state when the robot has reached the target location and offers Hors D’oeuvres to the people standing there and finally
- a “finished serving” state when the robot is at the serving table and there is no more food to serve.

If $S$ denotes the FSA then we can write $S=(Q, \delta, q_0, F)$, where: $Q = \{ \text{Start,}$
to-serve-location, to-base, serving, done } $q_0 = \{ \text{Start} \}$ $F = \{ \text{Done} \}$ and $\delta$ is given in Table 3. Additionally, Figure 3.2 shows a diagram of the FSA with all the states in $Q$ and the transitions between states.
Chapter 4

Hardware Setup

The work presented in this thesis is centered around building a useful system for robotics. Hence the research platform we use to carry the experiments is a mobile robot equipped with a number of audio and visual sensors.

The robot is an RWI B-14 mobile robot, named Eric. The robot can be seen in Figure 4.1. It has an onboard computer equipped with a 700MHz Intel PIII CPU and 256 MB of RAM. The computer runs the Linux operating system. It has 16 sonar, infrared and bump sensors built-in. We have not made use of any of these in our work. Ours is a non-holonomic robot, i.e., it can move instantly forward and backwards but not sideways without turning the wheels first. Such restrictions on the robot's velocity dictate, for example, the navigation algorithm we must use.

Eric sees using two different camera systems. One is the Triclops trinocular stereo vision system developed by Point Grey Research and the other is a Sony EVI-G20 color camera with pan-tilt and zoom capabilities.

Triclops can be seen in Figure 2.5. It has three cameras positioned on the vertices of a right-triangle. The cameras are precisely calibrated. Images are captured by all three cameras simultaneously and passed to vendor supplied software.
Figure 4.1: Eric the robot
that computes the pixel correspondences among them producing stereo maps. An example of one of the input images and the resulting stereo map was shown in Figure 2.6. Stereo maps are computed at a rate of 5 to 15 times a second depending on the input image size and the computational load on the host computer.

Eric sees in color using a second camera mounted below the Triclops. It is a Sony color web-cam with pan-tilt and zoom capabilities. These can be controlled through remote control or a serial connection to a computer. In this work we have not made use of its pan-tilt and zoom capabilities. The color images are aligned with the gray and stereo images of the Triclops. The color camera has a smaller field of view than the Triclops camera. As a result, we must compute the correspondence between the color and stereo images during a calibration stage. Both images, gray and color, are $320 \times 240$ pixels large. Because the color camera has a smaller field of view, i.e., it sees less of a scene than the Triclops, we must scale down the color images by a factor of 0.7 in both the horizontal and vertical directions. We then manually point the color camera such that the middle pixel of both the color and gray images correspond to the same point in the scene while both cameras are kept aligned vertically and horizontally. We use color images for skin color detection as described later in Chapter 5. We opted for manual calibration because this setup is temporary as future plans include upgrading to a color Digiclops trinocular system; Digiclops is the newest upgrade to Triclops for real-time stereo computation also developed by Point Grey Research.

We have also added three microphones on the top of the robot as audio sensors. They can be seen in part (a) of Figure 4.2. The bottom two microphones are used for sound localization as explained later in Section 5.2. The third microphone is used for capturing human speech used for speech recognition. All three
Figure 4.2: (a) Setup of three microphones. Two are used for sound localization and one for capturing human speech for speech recognition and (b) the wireless microphone receivers.

are omnidirectional microphones with build-in noise cancellation. These are wireless microphones each transmitting in one of three distinct radio frequencies. The signals from the bottom two are sent to a workstation and input to a single SoundBlaster sound card where they are processed for sound source localization. The signal from the other microphone is send to another workstation for speech recognition. The wireless receivers can be seen in part (b) of Figure 4.2.
Chapter 5

Description of Behaviors

In the following sections, we will give a description of each of the behaviors in our system. Each behavior is a stand-alone program. The inputs and outputs of each behavior are described along with the specific algorithm used to process the input(s) in order to produce the output(s). In describing each behavior, we will make use of the notation presented in Chapter 3. Notice that the order in which the behaviors are discussed is more or less random and has nothing to do with the relative significance of the behavior.

Here we will describe six behaviors, those that we developed. There is a behavior for each one of speech recognition, sound localization, user finding, head and gesture tracking, navigation and high level artificial intelligence. The latter is responsible for coordinating all other behaviors. Behaviors that we use but do not discuss here (since they already existed as part of the existing robot software,) are exploration and sensor data collection.
5.1 Speech Recognition

The Speech Recognition Behavior (SRB) is responsible for the very specific task of interpreting spoken speech into sentences. The language of communication for our system is English.

The input to this behavior is sound data. The output is an integer corresponding to a unique sentence out of a set of possible 42 different sentences. It should be immediately noted that this module is not concerned with Natural Language Understanding, i.e., the process of interpreting the spoken utterance to extract its meaning [BG99, BJ99, GNS+01]. Figure 5.1 shows the Stimulus-Response diagram for this behavior. We do not give the FSA diagram for this behavior because it trivially consists of a single state.

The SRB runs continuously, receiving sound data from a remote microphone attached to the robot. Sound data are processed using IBM's ViaVoice speech recognition engine. ViaVoice is an off-the-shelf speech recognition package. The speech engine is easily programmable through Sun's Speech for Java Advanced Programming Interface (SAPI). ViaVoice can be programmed using both a continuous or a command-and-control (also known as a rule) grammar. A continuous speech recognition grammar has a large vocabulary (ultimately equal to the size of the English language) and rules that capture the possible forms of English sentences. Theoretically, a user can speak any sentence and the recognition engine can recognize it. In reality, the large amount of rules that must be checked along with the large vocabu-
a "token" is any word that can be spoken

<table>
<thead>
<tr>
<th>can be spoken 1 or more times</th>
<th>can be spoken 0 or more times</th>
</tr>
</thead>
</table>

\[
<\text{Rule Name}> = <\text{Another Rule}>+ | \text{token}* ;
\]

always ends with ;

LHS always separates LHS and RHS

Figure 5.2: The structure of a simple rule in the Java Speech Grammar Specification

lary with many words that sound similar along with the distinct accent of different people makes recognition very inaccurate. ViaVoice makes an effort to improve correctness by considering the context of a spoken word in order to disambiguate among those words that sound similar but are spelled differently. Additionally, in order to decrease the number of recognition errors, a speech recognition system is trained to the vocabulary and accent of each user that uses it.

Some applications use a very limited vocabulary and sentence structure in order to significantly reduce the number of recognition errors. In such cases, a limited rule grammar is used. In a rule grammar, the application programmer specifies both the allowed vocabulary and sentences that can be spoken. The Java SAPI specifies the syntax for writing such a grammar. A software program that wants to use such a grammar first connects to the speech recognition engine (ViaVoice in our case) and then it loads the grammar and instructs the engine to use it as a guide for recognition. A carefully designed rule grammar can be made to be quite powerful in terms of what can be spoken while at the same time be very efficient computationally. Speech recognition is also well known to be very sensitive to environment noise especially when the number of rules and vocabulary grow in size. As
a result, we chose to build the SRB using a rule grammar tailored to the domain of our application.

We will briefly describe the notation used for writing a rule grammar although the interested user is referred to read the actual specification as given in [SunM]. The notation is based on the Extended Backus-Naur Notation, a mathematical way to describe a language.

Each line in the grammar represents one rule. Figure 5.2 shows the general structure of a simple rule. A rule consists of a name followed by its structure. It is structured as a series of tokens that can be spoken in the order specified. A token can be a reference to another rule or a word in the available vocabulary. The order in which the words are specified matters and the precedence is from left to right. Using the less-than, <, and greater-than, >, symbols, one can refer to another rule i.e., <my other rule's name>. Tokens can be grouped together using parentheses and alternatives can be specified using the | symbol. Optional tokens are specified using square brackets, i.e., []. Usually, one starts specifying the grammar by writing the words that can be spoken. Then, one writes the rules that specify the permitted sequence that words can be spoken. In this notation, the text in curly braces, i.e., {}, is the text that is returned by the speech recognition engine when a sentence that fits the rule is spoken. This returned string is used by the program to distinguish the sentence spoken. When a predetermined sequence of strings, corresponding to a valid sentence in the grammar, has been returned, it is mapped to a unique integer in the range of 0 to 41. This unique integer is the output of this behavior.

Using the syntax described in the Java SAPI documentation the rule grammar that we use in the SRB is given below:
grammar robot_control;

<robot_name>  = eric ;
<polite>  = please | kindly | thanks | thank you ;
<greeting>  = hello | hi | good morning | hey ;
<mode>  = diagnostic {diagnostic} | interactive {interactive};
<accept>  = yes | correct | ok | agreed | continue | cool |
confirmed | go for it | sure ;
<cancel>  = no | forget it | cancel | negative ;
<verb>  = move {move} | come {come} | (go | go to) {go} |
return {return} ;
<proposition>  = to | from | of | at ;
<location>  = closer {closer} | further {further} | target {target} |
base {base} | there {there} | away {away} | home {base};
<number>  = one {one} | two {two} | three {three} | four {four} |
five {five} | six {six} | seven {seven} |
eight {eight} | nine {nine} | ten {ten} ;
<status_token>  = voltage {voltage} | location {location} | orientation |
{orientation} | sound direction {sound_direction} ;
public <start>  = [ <greeting> ] [ <robot_name> | look [at me] ] {start};
public <demo_on>  = (demo on) {start_demo} ;
public <demo_off>  = (finished demo ) {stop_demo} ;
public <mode_change>  = (change | go) to <mode> mode ;
public <mode_current>  = What is the current mode {mode_check} ;
public <status_report>  = [<polite>] what is {status} [your | the] |
[current] <status_token> ;
public <command_move>  = <verb> [<proposition>] <location> [<number>];
public <command_move_simple>  = [move] ( forward {forward} | ( backward | 
back ) {back} ) ;
public <command_find_face>  = ( (Find | locate | can you see ) me ) |
{find} ;
public <command_track_face>  = ( ( Track | follow ) [me] ) {track} ;
public <command_stop>  = ( stop | halt | enough ) {stop} [all | 
everything] ;
public <command_rotate_simple>  = [rotate] ( left {left} | right {right});
public <command_set_target>  = set {set} ((target <number>) |
base {base});
public <polite_question>  = How are you {polite_question} ;
public <accept_action>  = [<polite>] (<accept> {accept}) ;
public <cancel_action>  = [<polite>] (<cancel> {cancel}) ;
public <stop>  = Good bye {bye} | So long {bye} |
(Until next time | exit now){bye};
There are many different sentences that can be spoken given this grammar. It is constructed to allow the user to speak both a very technical set of sentences as well as more natural ones. Some examples of technical sentences are “what is your voltage,” “finished demo,” or “enter diagnostic mode.” This simple and direct language allows a user to request specific technical information from the robot to assess its current mechanical status or change its mode of operation for debugging. On the other hand, a user can speak more natural sentences such as, “find me, please” or “move closer.” Notice that to enrich the language we allow for the use of more decorative terms in the sentences such as greetings, i.e., “good morning” and polite expressions, i.e., “thank you” or “please”. This grammar is by no means complete in terms of allowing a user to say any sentence and it was never intended to be. It is sufficient to allow interaction to the level of directing the robot to explore a certain region while at the same time allowing some flexibility as to what it can be said to make the interaction more pleasant.

The core number of sentences that this grammar specifies are mapped to a unique integer between 0 and 41 inclusive. The first entry, 0, is used to designate a recognized sentenced that is not expected. This entry actually exists just to help verify that the grammar can only accept the set of sentences that it was designed for. Through careful design, we know that our grammar is correct, i.e., it accepts all the expected sentences, and complete, i.e., it only accepts the correct number of sentences and no more.

The output of this SRB, as noted earlier, is one of these 42 numbers. This output makes sense since no natural language understanding or argument generation is really performed. As a result, to actually output the entire spoken sentence is of little help. The downside to this approach is that extending the SRB is rather
difficult. To increase the number of valid sentences one must change both the grammar and the source code for the behavior in order to correctly map the new sentences to a unique integer. At the same time all the behaviors that make use of the SRB's output must also be changed to handle the new output.

5.2 Sound Localization

The Sound Localization Behavior (SLB) is responsible for computing the direction of incoming sounds with respect to the robot. It allows the robot to orient itself towards the user. Having an approximate idea of the user's position also guides the processing of sensor data by other behaviors such as User Finding and Tracking discussed later.

Figure 5.3 shows the SR-diagram for the SLB. The stimuli are sounds that originate in the robot's immediate environment. The output is a 3D vector that points in the direction of the sound source along with a percentage that signifies the confidence in the results of the localization. If the confidence is below a certain threshold the behavior gives no output. Figure 5.4 gives the FSA diagram for this behavior. More specifically we have $S=(Q,\delta,q_0,F)$ where $Q=\{\text{idle, localizing}\}, q_0=\{\text{idle}\}, F=\{\} $ and $\delta$ is given in Table 5.2. This behavior is simple. It keeps looping until a loud sound is heard. When this happens the behavior computes a direction to the sound source and it returns the results. If no loud sounds are heard, it remains in an idle state. It has no final state. We proceed to describe how we perform the
localization by comparing the signals received by two microphones mounted on the robot. We follow with the discussion of two important issues, the limiting factor to the accuracy of the localization and a model for its error. The hardware setup of the microphones was presented earlier in Chapter 4.

5.2.1 Localization Algorithm

We will briefly discuss the concept of sound localization using signal matching of two signals captured simultaneously by two different microphones. A more complete treatment of sound localization can be found in [RM99].

Figure 5.5 shows the geometry, for the 2D case, of the setup for two microphones, \( m_1 \) and \( m_2 \), separated by a distance \( b \) and a single sound source. Since the speed of sound is constant (or more precisely can be assumed constant as long as the
Figure 5.5: The 2D geometry of a two microphone setup for sound localization.

(5.1) The medium of propagation does not change composition) the time delay of the signal at the two microphones is related to the ratio of the distance of each microphone to the sound source by

\[ \text{TimeDelay} = n = \frac{r_2 - r_1}{c} \]

where \( r_1 \) and \( r_2 \) are as shown in Figure 5.5 and \( c \) is the speed of sound.

Equivalently the time delay between the two signals is related to the angle \( \alpha \). More specifically,

\[ \sin(\alpha) = \frac{cn}{fb} \] (5.2)

where \( c \) is the speed of sound, \( n \) is the time delay in samples, \( f \) is the sampling frequency and \( b \) is the distance between the microphones. Equation 5.2 assumes that the sound source is at an infinite distance with respect to the baseline, i.e., the direction of arrival of the sound is the same for both microphones, \( m_1 \) and \( m_2 \); this is called the far-field assumption. Given this assumption, we can rearrange Equation
5.1 to compute the distance $r$ by computing the time shift between the two signals from the two microphones.

In 3D, the baseline defined by the microphone pair has orientation $(\phi, \theta)$ in spherical coordinates and a time delay computed from the two signals corresponds to a cone of possible source directions as shown in Figure 5.6. If we have three pairs of microphones then we can intersect the cone defined by each pair for the same sound source and compute a unique direction. Of course the task of intersecting the cones is not a simple one considering the error in computing the cone for each baseline. This error is the result of environmental factors and errors in the signal matching process.

For our application, we only need an approximate direction to the sound source. We take advantage of the far-field assumption and our person model (see Figure 5.14) to perform robust localization using only two microphones.

Earlier in Chapter 4, we presented the hardware setup of two omnidirectional microphones mounted on-top of the robot. The two microphones define a single baseline and for any incoming sound we can compute the cone of directions as mentioned above. We capture the sound signal from both microphones simultaneously and then we perform signal matching to recover the time shift between the two. The baseline is set to 20cm. Assuming that we can resolve the time-delay between the two signals only in integer units we can compute the smallest discernible angle for our setup by Equation 5.3 (presented in [RM99]).

$$\sin(\alpha_{min}) = \sin^{-1}\left(\frac{c}{fb}\right) = \sin^{-1}\left(\frac{342m/s}{22050Hz \times 0.20m}\right) \approx 4.5°$$

Assuming a sound source at an angle $\alpha = 90°$ we can compute the range of possible
Figure 5.6: The 3D configuration of the sound localization problem. Shown are the angle of incidence, $\alpha$, the two microphones, $m_1$ and $m_2$, the baseline vector, $b$, the direction to the sound source vector, $s$ and the orientation vector of the baseline given in cylindrical coordinates by $\phi$ and $\theta$. Also shown is the cone traced by all possible sound source directions that would produce the same time delay as the direction $s$. [RM99]
Time Delay in integer units in degrees

<table>
<thead>
<tr>
<th>Time Delay in integer units</th>
<th>α in degrees</th>
<th>Time Delay in integer units</th>
<th>α in degrees</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0.0</td>
<td>7</td>
<td>32.9</td>
</tr>
<tr>
<td>1</td>
<td>4.5</td>
<td>8</td>
<td>38.3</td>
</tr>
<tr>
<td>2</td>
<td>8.9</td>
<td>9</td>
<td>44.3</td>
</tr>
<tr>
<td>3</td>
<td>13.4</td>
<td>10</td>
<td>50.9</td>
</tr>
<tr>
<td>4</td>
<td>18.1</td>
<td>11</td>
<td>58.5</td>
</tr>
<tr>
<td>5</td>
<td>22.8</td>
<td>12</td>
<td>68.5</td>
</tr>
<tr>
<td>6</td>
<td>27.7</td>
<td>13</td>
<td>90.0</td>
</tr>
</tbody>
</table>

Table 5.2: Correspondence of angles to time-delay units for two microphones separated by 0.25m.

time-delays by Equation 5.4 (also from [RM99]).

\[
\pm n = \pm \frac{f b}{c} \pm \frac{22050 \text{Hz} \times 0.20\text{m}}{342\text{m/s}} \approx 13
\]  

(5.4)

That is 27 distinct time-delay units in the range $[-13, 13]$. One can use interpolation to achieve sub-sample resolution. We can easily compute the corresponding angle for each number of time delay units for our setup and these are shown in Table 5.2.1.

To compute the time-delay between the two signals we perform the following steps, [RM99]:

1. Capture signals from both microphones simultaneously.

2. Filter the signals by applying a band-pass filter in the range of 300-3000Hz wherein lie most of the frequencies of human voice.

3. Compare the power of the signal to a threshold value and only proceed if it exceeds this value. Otherwise return.

4. Smooth signals and compute the largest peaks.
5. Apply a Hough-transform algorithm to compute the correspondence of peaks between the two signals. Use a bin of 33 values, one for each discernible angle.

6. The bin with the largest accumulated value is the selected direction. The confidence of the localization is computed as the ratio of the power at the selected peak to the power of the signal.

It is important to note that we apply the above processes to localize on impulsive sounds. Since the sampling rate is 22050Hz, each pair of signals corresponds to roughly 1/3sec slice of the incoming sound. Localization as presented above is performed for each instance independently. Since people speak for longer than 1/3sec duration it makes sense that consecutive localization results should be smoothly related. So, we forward propagate the results of the localization at each instance not allowing sudden changes to the computed direction that could result by random noise such as someone closing a door.

5.2.2 Signal Noise and Error Modelling

The accuracy of the localization depends both on the size of the baseline, with a larger baseline yielding a larger number of discernible angles, and the error in the signal matching. We have set the baseline distance to 0.2m which gives a reasonable number of discernible angles. Signal matching becomes difficult because of noise in the input sound data. In this section we discuss how we deal with noise.

Noise in the input signal is a function of the environment of operation. Sound waves from the source will most likely bounce off objects before reaching the robot. The original signal and its echoes are received at the robot and skew the localization results. The problem of echoes is emphasized in environments with many specular, i.e., highly reflective, surfaces. We want a robot that can operate in any environ-
ment. We have two mechanisms for minimizing the localization error induced by the specularity of surfaces in the environment.

For one, we only localize on sounds that have power above a certain threshold. If a reflective surface perfectly reflects an incoming signal then no loss of signal power would occur. In real environments this is not the case. Every time that the signal bounces off a surface its power is reduced by a certain percentage that depends on the reflective properties of the surface and the spectrum of the signal. We do realize that a simple check on the power will not eliminate all echoes but it will definitely reduce the effects of those that have either been reflected by more than one object or that have bounced off less reflective surfaces.

Secondly, our simple assumption that the localization results for consecutive signals of 1/3sec duration, as mentioned above, should vary smoothly aids in reducing the error that would result from echoes of the original signal.

We have also found that a major source of noise in the incoming signal is the robot itself. There are many mechanical components in the robot that range from motors to cooling fans for the onboard electronics. In particular, one of the cooling fans for the onboard computer that is situated closer to one of the microphones tends to bias the localization results. We filter the captured signal to remove the frequencies that are mostly affected by this fan. We already filter the incoming sound signal by applying a bandpass filter in the range 300Hz to 3000Hz where most of human speech lies. Noise coming from the fan is white noise with components all over the spectrum. We have not been able to isolate and remove this signal. We measured the power of this signal and set our power threshold mentioned above to a value twice as large. Hence we only localize on only strong incoming sounds that drown the noise signal.
Figure 5.7: The measured sound localization error. The solid line shows the theoretically discernible angle. Triangles show the median of 15 samples taken at each location and squares show the average of the same 15 samples. The error bars show the standard deviation for the median values.
Table 5.3: The measured localization error. Where N/A is shown the angle gave extremely erroneous results and are not shown here.

We measured the real error in the localization for a number of different angles as shown in Figure 5.7. This is the error in computing the angle $\alpha$. The error can be computed directly from Equation 5.2 from the estimated values of time-delay. Using the rules of error propagation, the error in the angle $\alpha$, denoted by $\delta\alpha$, is given by Equation 5.5.

$$\delta\alpha = d\frac{d}{dn} \left( \sin^{-1} \left( \frac{cn}{bf} \right) \right) = \left( \frac{\frac{cn}{bf}}{\sqrt{1 - \left( \frac{cn}{bf} \right)^2}} \right)$$  \hspace{1cm} (5.5)

Figure 5.8 shows a plot of the error function for the angle $\alpha$. We can see that for the larger values of the time-delay occurring for larger values of the angle $\alpha$ the error grows to infinity. This explains why the measured error shown in Figure 5.7 is very accurate until about $n = 6$ and it grows quickly thereafter.

<table>
<thead>
<tr>
<th>Time Delay in integer units</th>
<th>Real Angle</th>
<th>Measured Angle</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Average</td>
</tr>
<tr>
<td>13</td>
<td>90.0</td>
<td>42.18</td>
</tr>
<tr>
<td>12</td>
<td>85.5</td>
<td>N/A</td>
</tr>
<tr>
<td>11</td>
<td>81.0</td>
<td>N/A</td>
</tr>
<tr>
<td>10</td>
<td>76.5</td>
<td>-2.11</td>
</tr>
<tr>
<td>9</td>
<td>71.9</td>
<td>N/A</td>
</tr>
<tr>
<td>8</td>
<td>67.2</td>
<td>34.89</td>
</tr>
<tr>
<td>7</td>
<td>62.3</td>
<td>90.00</td>
</tr>
<tr>
<td>6</td>
<td>57.1</td>
<td>54.82</td>
</tr>
<tr>
<td>5</td>
<td>51.7</td>
<td>37.81</td>
</tr>
<tr>
<td>4</td>
<td>45.7</td>
<td>46.24</td>
</tr>
<tr>
<td>3</td>
<td>39.1</td>
<td>37.96</td>
</tr>
<tr>
<td>2</td>
<td>31.4</td>
<td>28.88</td>
</tr>
<tr>
<td>1</td>
<td>21.5</td>
<td>22.45</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>-3.64</td>
</tr>
</tbody>
</table>
5.3 User Finding

For building a natural human-robot interface, it is necessary that the robot knows where the user is during the interaction. Such knowledge can be used to decide on the emotion, context and social rules of conversation. It is also necessary for the robot to be able to analyze the user’s gestures given its camera’s limited field of view. The robot must be able to identify a person out of all objects in a scene. There is a behavior designed to accomplish this task, the User Finding Behavior (or UFB.)

Once again, we first present the SR diagram for this behavior in Figure 5.9. The behavior remains dormant until it is asked to search for the user. So, the only stimulus is the simple command FIND_FACE. The behavior searches for the user’s face in images captured with the onboard cameras. This explains the choice of name for the command. The output of this behavior is simply the 3D location of the user’s
face. If the user is not found then the location is set to all 0's to indicate failure.

The FSA diagram for the UFB is presented in Figure 5.10. So, $S=(Q, \delta, q_0, F)$
where: $Q = \{ \text{idle, Searching for user} \}$ $q_0 = \{ \text{idle} \}$ $F = \{ \}$ and $\delta$ is given in table 5.3. This behavior also does not have a final state. It remains in the “idle” state until asked to find the user. It temporarily switches to the “Search for user” state and when it is done searching it returns back to the “idle” state.

The UFB searches images obtained using the onboard cameras (both the color SONY and gray scale Triclops) in order to find the user. Images are first obtained and then processed in the sequence shown in Figure 5.11. There are three types of input images as already mentioned. We use RGB color images obtained

<table>
<thead>
<tr>
<th>$\delta$</th>
<th>$q$</th>
<th>$\text{input}$</th>
<th>$\delta(q, \text{input})$</th>
</tr>
</thead>
<tbody>
<tr>
<td>idle</td>
<td>no-stimulus</td>
<td>idle</td>
<td></td>
</tr>
<tr>
<td>idle</td>
<td>search-for-face</td>
<td>Searching for user</td>
<td></td>
</tr>
<tr>
<td>Searching for user</td>
<td>done-searching</td>
<td>idle</td>
<td></td>
</tr>
</tbody>
</table>

Table 5.4: The transition function $\delta$ for the User Finding behavior
Get Color, Grey and Stereo Images

Compute Real Depth Image

Do Skin Color Segmentation on Color Image

Apply Median Filter on Skin Binary Image

Compute Connected Components

Apply an Area Threshold Filter

Compute Component Centers

Compute the Components' Distance from the Camera

Compute the Likely Size of a Face at the Distance of Each Component

For each Component Compute Motion and Face Similarity Scores

Compute Total Probability of Being a Face for Each Component

Return the Most Likely Face Component

Figure 5.11: The processing steps performed on images for finding a person
from the onboard SONY camera; gray scale images obtained from the onboard Triclops camera and stereo images also obtained from the Triclops. We can obtain three gray scale images from the Triclops system since it actually has three cameras. However, we always use the image from the right camera which is the reference one, i.e., the stereo maps are aligned for this camera.

As mentioned earlier, this behavior searches for the user in the images by finding his/her face. There are several reasons for this. First, a face has distinct characteristics that make it stand out with respect to background objects. Such features are its color and geometric shape. Second, one can always expect to be able to see an almost frontal view of the person's face a result of the fact that when a person talks to another he/she almost always looks directly at him/her. This is a fundamental aspect of interpersonal communication. Finally, the texture of a face as seen in a 2D image, i.e., the combination of the face, hair, eyes, mouth and ears, is similar for all people as opposed to using information about one's clothing that for example can change not only from person to person but also for the same person seen at different times.

The first step to finding the user is to search for skin color in the color images. The RGB color image is first transformed to the HSV (Hue-Saturation-Value) color space where the color of human skin forms a small tight cluster. The values of all three channels, H, S and V, range from 0.0 to 1.0. We then threshold the HSV image in value so that too bright or too dark pixels are discarded. Pixels that are too bright are either the result of the imaging process, i.e., the dynamic range of the camera is not large enough to capture pixels that are brighter than a certain value, or are the result of specularities on the surfaces of objects. Human skin in particular tends to be very specular. We have tuned our system to work in environments illuminated
by ambient artificial light that adds in the reduction of specularities. We set the valid range for Value to be a constant in the range of 0.2 to 0.5 (the Value channel takes values between 0.0 and 1.0), i.e., not too dark and not too bright.

Next we threshold the image in the Hue and Saturation channels. Pixels that are within a given range of values for each channel are selected as the most likely skin pixels. The difficult part here is to actually determine the appropriate range of valid pixel values for each of the channels. We employ a simple method that works well for our application domain. It only requires that we provide the system with an average of 40 to 50 sample skin pixels at the current level of illumination. To acquire the data, a person positions himself in front of the camera while another manually selects skin pixels in the color image. The mean and standard deviation of these pixels is then computed for each of the Hue and Saturation channels and the valid range of skin pixel values is set as the mean plus/minus two times the standard deviation. Figure 5.12 part (a) shows the hue of 20 sample pixels for 8 different people. The people have been selected to have varying skin hue. Part (b) of Figure 5.12 gives the median and standard deviation for the pixels corresponding to each person. We can see that the Hue values for all the subjects cluster around 30 with standard deviation of plus/minus 10. We observe similar results for the Saturation channel shown in parts (a) and (b) of Figure 5.13.

The next step is to eliminate any false positives that have been detected during the skin color segmentation step. False positives are regions that have been marked as skin even though they are not, i.e., a brown box in the background could possibly fall in the same range as human skin and so it could falsely be included in the segmented image. We perform connected component analysis on the binary image resulting from the last step in order to eliminate the false positives.
Figure 5.12: Sample Hue data for 8 different people of varying skin color (a) sample Hue data for 20 sample skin pixels, (b) the mean Hue value for the data shown in (a)
Figure 5.13: Sample Saturation data for 8 different people of varying skin color (a) sample Saturation data for 20 sample skin pixels, (b) the mean Saturation value for the data shown in (a)
Before computing the connected components however, we apply a median filter to the binary image. Applying a median filter implies that we count the set pixels in the local neighborhood of each pixel in the image; if the sum of these pixels is larger than the sum of pixels that are not set, then the center pixel is set and vice versa. Convolving a binary image with a median filter has the effect of growing the set regions. Additionally, it helps eliminate any Gaussian noise in the image. The face region is expected to have gaps in the interior, the result of facial features such as the eyes and mouth that are not of skin color and are excluded during the segmentation step. These are the gaps that we want to fill in as they contribute to the area of the component that becomes important in a later step as we explain further on. Through experimentation, we concluded that the best results are obtained when using a 7 × 7 size filter. The results are better in terms of eliminating noise and growing the valid region as well as in terms of computational efficiency.

The next step is to actually compute the connected components in the image. We scan the image and we mark all neighboring pixels as belonging to the same region. We look in the 4 neighborhood of each pixel; the 4 neighbors of a pixel are the pixels, north, south, east and west from it. We use a well known recursive algorithm as presented in [JKS95] to compute the connected components. Once all the components are identified, we apply a simple area threshold algorithm to eliminate the smallest of them. We expect that the person will be within a few meters from the robot and so his/her face could be no smaller than a certain size. From our experiments we set the size to 49 pixels, that is, a 7 × 7 pixel region. So, any component with an area smaller than 49 pixels is deleted as background noise. This is one of the two reasons that makes filling in the areas of the eyes and the
mouth necessary. Because of the wide field of view of the camera and the fact that
the person must be at least 3.0 meters from the camera in order for his face to be
visible and the stereo to be accurate, the face covers only a small area in the image.
If we do not fill these gaps in the skin segmented image then we run the danger of
eliminating the real face component as noise.

Once the less likely components have been eliminated using the procedure
outlined above, we need to select one of the remaining components as the user. We
determine a score, i.e., a value between 0.0 and 1.0, where a large value denotes a
higher chance that it is a face. This number is computed as the linear combination
of 3 variables (also referred to as scores.) The values of these variables are also
in the range of 0.0 and 1.0, inclusive. The first score measures the similarity of
each component with respect to the area it covers in the image given the expected
size of a person’s head at that distance. The second measures the similarity of
the component to a face template that was acquired offline and finally the third
penalizes components that are at a distance that the user is less likely to be at given
the parameters of the hardware. The result of the linear combination of these three
variables is multiplied by a Gaussian whose $\sigma$ depends on the results of the Sound
Localization behavior.

More formally, we evaluate the following Equation, 5.6, for each component,
$c_i$:

$$p(c_i) = G_\sigma(i) \times (w_1 \times \frac{\text{size}_{\text{expected}}}{\text{size}_{\text{actual}}} + w_2 \times \text{Score}_{\text{Template}} + w_3 \times D_z(i)) \quad (5.6)$$

where $w_i$ is the weight of the $i-th$ attribute, $G_\sigma(i)$ is the Gaussian determined by
the localization error, $\frac{\text{size}_{\text{expected}}}{\text{size}_{\text{actual}}}$ is the size attribute, $\text{Score}_{\text{Template}}$ is the matching
cost of the component gray scale image and a template of the user’s face and finally
$D_z(i)$ is a penalty term for components that are too close or too far from the camera.
Figure 5.14: A simple model capturing important information about a person including a person’s average height, head size, arm and shoulder length.

Each attribute is further explained in the following paragraphs.

To compute the face size attribute, we first determine the distance of each component from the camera. The distance is computed as the mean of the distance of all the pixels belonging to the component. Then we compute the most likely size of a person’s face at that distance. We measured the average size of a person’s head to be approximately 11x15cm; this is shown in figure 5.14 that presents our model of a person including information about a person’s average height, head size and arm length. Then the expected size of the person’s head at distance $z$, given the 110 degree horizontal field of you of the Triclops camera, is given by Equation 5.7:

$$\text{face}_{\text{expected, width}} = \text{face}_{\text{template, width}} \times \left( \frac{2 \times \text{Image}_\text{width} \times \text{face}_\text{width}}{\text{component}_\text{distance} \times \tan(\text{fov}_x)} \right)^{-1} \quad (5.7)$$

where $\text{face}_{\text{template, width}}$ is the width of the face template image from the training set; $\text{Image}_\text{width}$ is the width of the input Triclops gray scale image that in our
experiments is 320 pixels; *face\_width* is the measured width of the average person face to be 11cm as already mentioned; *component\_distance* is the distance of the component from the camera and *fov\_x* is the horizontal field of view of the Triclops camera. We compute the expected height of the face at the component’s distance as given by Equation 5.8 below:

\[
\text{face\_expected\_height} = \text{face\_expected\_height} \times \frac{15}{11}
\] (5.8)

where the ratio \(\frac{14}{11}\) is derived from the average face size of 11x14cm. Because *face\_expected\_width* and *face\_expected\_height* are computed in integer number of pixels, lastly we must take the ceiling of the computed values in Equations 5.7 and 5.8.

Because we want to measure the relative size of the actual and expected areas of the connected component and because we require that the ratio of areas is always in the range of 0.0 and 1.0, we need to modify the first part of Equation 5.6 to be

\[
\frac{\text{Size\_actual}}{\text{Size\_expected}}\]

if the expected size is larger than the actual component size. So, a 20 percent difference in size is returned whether \(\text{Size\_actual} = 100\) and \(\text{Size\_expected} = 120\) or vice versa.

The second component attribute is the *Score\_Template*. We compute it as the normalized cross correlation score of the component’s gray scale image values and an image template of the user’s face. The cross correlation is given by Equation 5.9 below:

\[
\text{Score\_Template} = \frac{\sum_{k=1}^{m} \sum_{l=1}^{n} g[k, l] f[k, l]}{\sqrt{\sum_{k=1}^{m} \sum_{l=1}^{n} f^2[k, l]}}
\] (5.9)

where \(g\) denotes the \(m \times n\) face template and \(f\) denotes the \(m \times n\) face region in the gray scale image. An example of a face template is shown in Figure 5.15. It can be shown that the cross-correlation score is maximized for \(g = c \times f\) where \(c\) is an integer.
greater than 0. The results are correct only for frontal views. The cross-correlation score does not perform well for partial views of the template. Additionally, we require one template for each user that we want to identify. Currently the system is only setup to work with one person at a time.

Alternatively, one can compute the cross-correlation score using the binary skin images instead of the gray scale images. This way the template matching is no longer user specific. We did experiments using binary templates and we found little difference in performance in terms of only a single user interacting with the robot.

In general, we have found that the template score by itself is not reliable enough for face detection. This is expected considering that the area that the user’s face occupies in the image is most often very small for an accurate match. This is the result of both the large field of view of the Triclops’s cameras and the average distance of 3.0 meters that the user must be away from the robot in order to be seen by the color camera as the latter has a rather small field of view.

There is one last term in Equation 5.6, the distance penalty. Each component is penalized depending on its distance from the camera. We know that stereo data
is inaccurate for object closer than 0.5 meters and further away than 5.0 meters. Also, since the average height of a person is somewhere between 1.4 to 1.8 meters and because of the 80 degree field of view of the color camera, we are most likely to see the face of a person that is at least 3.0 meters away from the robot. In approximation, the same dimensions apply for a person sitting on a chair. Additionally, we expect that there are no obstacles between the robot and the user which means that we want to favor a person that is closest to the robot. We define the penalty term as

\[
D_z(i) = \begin{cases} 
0 & \text{if } z < 3.0 \text{ and } z > 5.0 \\
\alpha \times z & \text{otherwise}
\end{cases}
\]  

(5.10)

where \(\alpha\) is set to 0.5. So the components are penalized linearly the further away from the camera that they are.

The values of the three attributes discussed above are linearly combined to compute the total component face probability. The weights \(w_1\), \(w_2\) and \(w_3\) where experimentally determined to equal

\[
w_1 = 0.3 
\]  

(5.11)

\[
w_2 = 0.4 
\]  

(5.12)

\[
w_3 = 0.3 
\]  

(5.13)

This probability is lastly multiplied by a Gaussian with \(\sigma\) that depends on the sound localization error. Multiplying the resulting score with the Gaussian will favor components that are closer to the center of the image if the sound localization error is small. If the error is large then we want to give components that are further away from the image a better chance of being selected. This makes sense since if the sound localization error is small then the robot will rotate and position the user
in the middle of the cameras' field of view and so it is very likely that one of the components near the center is the user. Alternatively, if the error is large then there is a good chance that the person might not be centered in the image and we do not want to ignore any components that are not near the center.

Figure 5.16 shows the result of each of the main steps in locating the user in the input images. In this example we see how the input images are processed to select the candidate face regions followed by the result of the connected component analysis to select one of these as the user's face. We see how our approach correctly selects the face component even though there are two other components, those corresponding to the user's hands, that are very similar in size and shape to a face.

5.4 User Head/Gesture Tracking

Once the user's location is found, we can avoid the difficult and inaccurate process of relocating him every time that we need information about his state by keeping the location current over time. Once the user is located, we track his position in 3D space forward in time. The User Head/Gesture Tracking Behavior (or UTB) is responsible for this task. It has the responsibility of a second task, that of determining where the user is pointing on the floor.

We present the SR diagram for this behavior in Figure 5.17. There are two major stimuli, each one triggering one of the two tasks of head and gesture tracking. The FACE_LOCATION stimulus initializes the head tracking subtask. When one sends this message, it must be accompanied by the last known \((x, y)\) location of the user's head in the 2D image plane; in our system, this location is the output of the user finding behavior described earlier. The second stimulus, GET_DIRECTION, is valid only if received after a FACE_LOCATION message and
Figure 5.16: Example of finding the user in an image, (a) the original gray scale image, (b) the original color image, (c) the result of the skin color segmentation after applying the median filter and with the color image data registered with the gray scale image data, (d) the connected components remaining after detection and applying the area threshold filter, (e) the Gaussian penalty function and (f) the component selected as the user's face shown with a rectangle at the expected face size superimposed on the color image; the square dots mark the centers of all of the connected components processed.
a valid track exists. The UTB responds with an output message only as a response to a GET_DIRECITION stimulus. The output is one of the four messages, POINTING_RIGHT, POINTING_LEFT, NO_DIRECTION or LR_DIRECTION, indicating where the user is pointing or an error condition. The first two messages are accompanied by the \((x, y)\) coordinates on the floor plane where the user is directing the robot to go. The second last message, NO_DIRECTION, indicates that the gesture tracking subtask has failed to produce a valid direction. The last of the four messages, LR_DIRECTION, indicates an ambiguous direction, i.e., that both the left and right arms have been detected and the algorithm cannot distinguish which one the user is pointing with.

Once again, all four output messages are returned by the gesture tracking subtask. The head tracking subtask does not return any messages other than keeping the current location of the user valid. If the head tracking subtask loses the track then the behavior simply returns to an idle state.

We present the FSA for this behavior in Figure 5.18. More precisely, we have \(S=(Q, \delta, q_0, F)\) where: \(Q = \{ \text{idle, tracking, compute direction}\}\) \(q_0 = \{ \text{idle}\}\) \(F = \{ \}\) and \(\delta\) is given in Table 5.4. The behavior remains in the "idle" state until it receives the FACE_LOCATION stimulus at which point it enters the face tracking state. As long as the behavior tracks the user's location, it remains at this state. If while at this state, a GET_DIRECITION stimulus is received then it moves to the third state "compute direction." It returns to the "tracking" state once the
computation finishes. If the track is lost then it returns back to the "idle" state. If the GET_DIRECTION state is received while in the "idle" state, it responds with a NO_DIRECTION message.

The following two sections provide the details of the algorithms used for head and gesture tracking.

<table>
<thead>
<tr>
<th>δ</th>
<th>q</th>
<th>input</th>
<th>δ(q, input)</th>
</tr>
</thead>
<tbody>
<tr>
<td>idle</td>
<td>face-location</td>
<td>has-track</td>
<td>tracking</td>
</tr>
<tr>
<td>tracking</td>
<td>tracking</td>
<td>get-direction</td>
<td>compute direction</td>
</tr>
<tr>
<td>compute direction</td>
<td>has-track</td>
<td>lost-track</td>
<td>tracking</td>
</tr>
<tr>
<td>tracking</td>
<td>has-direction</td>
<td></td>
<td>idle</td>
</tr>
</tbody>
</table>

Table 5.5: The transition function δ for the User Tracking behavior
5.4.1 User Head Tracking

To keep the user's 3D position current, we chose to track the position of his head. The head is a good candidate for tracking since its shape remains constant as seen from all directions. The head tracker that we implemented is the one proposed in [Bir97] and it is contour-based. It uses a simple prediction method to track the ellipse that a person's head traces on the 2D image plane. We use the original algorithm with two proposed changes of using the stereo information, available from the Triclops sensor, and an a priori known model of a person to improve the algorithm's robustness in cluttered environments.

This tracker has been shown to be robust to temporary occlusion. It is real-time. It is robust to out of plane rotations of the head, i.e., it keeps locked on the target even if the person shows the back of the head. It has also been shown to be partially resilient to background noise. We first explain the algorithm and then we propose two small additions that improve tracking in cluttered backgrounds.

The core of the algorithm uses a simple two dimensional model, namely an ellipse, to track the contour the head traces on the edge image. Intuitively, using an elliptical model for the projection of the head on the image plane makes sense. The size of the ellipse is a function of the distance of the head from the camera. It varies as the person moves closer or further away.

The contour traced by the head on the image plane is tracked by maximizing the normalized sum of the gradient magnitude around the perimeter of the estimated ellipse. We perform a local search of hypothesized head locations around the last known position of the head. The only assumption is that of constant velocity that leads to this simple prediction scheme. Let \( s_t = (x, y, \sigma) \) denote the ellipse of size \( \sigma \) and centered at \((x, y)\) on the image at time \( t \). The prediction of the new head...
location derives from the constant velocity constraint given the known location of
the head in the previous two frames and it is given in equation 5.14 below:

\[ \begin{align*}
x_t &= 2x_{t-1} - x_{t-2} \\
y_t &= 2y_{t-1} - y_{t-2} \\
\sigma_t &= \sigma_{t-1}
\end{align*} \tag{5.14} \]

where we predict the location at time \( t \) given the known locations at times \( t - 1 \) and
\( t - 2 \). We denote the set of possible ellipses as \( S \). Notice from equation 5.14 that
the algorithm does not explicitly search over the size of the ellipse.

For each predicted ellipse in \( S \), we compute the normalized sum of the gra­
dient magnitude around it as given in equation 5.15 below:

\[ s_t = \text{argmax} \left\{ \frac{1}{N_\sigma} \sum_{i=1}^{N_\sigma} |g_i| \right\} \tag{5.15} \]

where \( N_\sigma \) is the number of pixels on the perimeter of the ellipse and \( |g_i| \) is the
absolute value of the gradient at pixel \( i \). The choice of the edge detector is not
specified in the algorithm. We will discuss later our selection of edge detector after
we explain our improvements to the algorithm that allow us to choose almost any
kind of edge detection scheme.

So far we have described the original algorithm. We have implemented the
algorithm as mentioned with two additions. First, we process the input image data
to remove background noise as much as possible and second we predict the size of
the ellipse at each image location by using a priori knowledge of a person’s average
head size.

Birchfield has shown that the algorithm might lose the track in scenes with
cluttered background. We define the background as all the pixels in the image that
do not belong to the person that we are tracking. He makes no effort to remove
this noise, a task that in his case would have been difficult as the only available information to him are the 2D gray scale images. We, however, have access to depth maps to accompany the raw image data. We take advantage of this extra information. At each hypothesized head location, we extract the distance from the camera. We segment the image using this distance value ± a small number to compensate for the error in stereo. We use only 32 disparity values for stereo and because the head only occupies a small number of pixels in the image, we expect that all the pixels in the head will be at approximately the same depth. Our experiments have verified this assumption.

Figure 5.19 shows the depth segmented image for one of the hypotheses. We have set all the pixels that belong to the background to zero intensity. A fortunate side effect of the segmentation is that in the segmented image we always get the outline of the head and upper torso clearly defined. That is we can compute a strong elliptical edge at the location of the head, exactly the contour we wish to track with equation 5.15. This strong edge is clearly visible in the edge image shown in Figure 5.19. Secondly, we have improved the original algorithm by correctly predicting the correct size of the ellipse at each hypothesized head location. We can do this by computing the size of the head at the distance of the test location using the measured size of the average head and equations 5.8 and 5.7. From our model of a user, shown earlier in Figure 5.14, we take the average size of a person's head to be 15 × 11cm. We set σ to equal the calculated head's width. This prediction allows us to reduce the number of hypotheses that we need to search hence saving valuable CPU cycles.

We deferred our discussion of our choice of an edge detector until this point. The reason is that in most computer vision applications actually choosing the correct
Figure 5.19: Display of some of the data used in head tracking, (a) The intensity image depth segmented for one hypothesis and (b) the edges of the image shown in (a). One can clearly see the edge defining the location of the person's head.
edge detection scheme along with the best parameters for obtaining the best results is an extremely difficult task. Our background subtraction scheme allows us to use almost any kind of edge detector even the simplest one of taking the difference among neighboring pixels in the horizontal and vertical directions. The reason for this is because as we have already mentioned, removing the background pixels creates a very strong edge in the image where the head stands out with respect to the background in the stereo map. This edge is easily detected. However, since the stereo maps are not fine enough to retrieve the head-background edge perfectly, we use a more robust but well known edge detector, the Laplacian of Gaussian [Hil92, JKS95]. The LofG is robust as it filters the image of Gaussian noise and it is easy to implement as it only requires the convolution of the image with an $n \times n$ easily computable mask.

The algorithm proceeds as follows:

1. Get the gray and stereo images denoted $I_g$ and $I_s$ respectively. These are the input data along with the last known location of the user's head, $l_{head}$.

2. Compute the edges in $I_g$ by convolving with a LofG mask.

3. For each pixel, $i$, in the neighborhood of $l_{head}$, $i = 1...N$.

   (a) Compute the distance to the camera, $d_i$.

   (b) Segment $I_s$ using $d_i$ as a guide in order to remove the background.

   (c) Compute the expected size of the head at $d_i$.

   (d) Fit an ellipse and compute 5.15.

4. Select the location that maximizes 5.15 as the new head location and return.
We have found that we obtain the best tracking results when searching the $11 \times 11$ neighborhood around the last known position of the head. Also, we set the size of the convolution filter used for edge detection to $5 \times 5$ generated using $\sigma = 2$ pixels. Even though the tracker can run in real-time, in our system we limit it to running at approximately 5fps in order to reduce the computational load on the CPU. This constraint does cause it to lose the track if the person moves too fast. We will address this issue later when improvements to our control architecture will allow us to stream image data off-board and essentially dedicate a workstation entirely to this behavior.

5.4.2 User Gesture Tracking

The second task that this behavior is responsible for is what we refer to as user gesture tracking. To be more precise however, this behavior only extracts the 3D position of the user's hands and computes the direction that the user is pointing once when requested. The term tracking implies that the position of the hands is kept current over a number of frames forward in time. Our simple approach to gesture recognition is reliable enough that we can extract the user's gesture whenever needed and so it is as if we are actually tracking the hands. Later, we envision the addition of a more traditional tracking algorithm not too dissimilar to the head tracker that we presented in the previous section. These justify our use of the term gesture tracking for this task.

The user needs to be able to direct the robot to different locations in the room. The most natural way to do this is by pointing to that location. The task of extracting this information from image data is not an easy one. We have taken advantage of simple results such as depth and skin image segmentation along with
straightforward assumptions that derive from our application domain to efficiently
and robustly compute this direction.

The algorithm starts by utilizing the already known location of the user as
kept current by the head tracking task. Color, gray and stereo images are initially
captured and along with the last known image location of the user's head comprise
the input data. The output is the 2D location on the floor where the user is point-
ing or a flag indicating one of two error conditions, one designating an ambiguous
location and another indicating that no location was found. The major steps in the
algorithm are as follows:

1. Get the gray, color and stereo images denoted $I_g$, $I_c$ and $I_s$ respectively. These
   are the input data along with the last known location of the user's head,
   $l_{head}$.

2. Select the skin pixels in the neighborhood of $l_{head}$, and generate the set of skin
   pixels $N$.

3. Compute the average depth, $d_N$, of the pixels in $N$.

4. Segment the color and gray images using $d_N$ as a guide.

5. Select the skin pixels in the segmented image forming the set of foreground
   pixels $M$.

6. Compute the connected components with the pixels in $M$.

7. Select the component closest to the last known head location as the face.

8. Classify the remaining components as being to the left or right of the face
   component. Construct the sets $L$, $R$ respectively.
9. Remove the components from \( L, R \) that are too far or too close to the face component.

10. From the components remaining select the dominant direction.

11. Compute the point on the floor and return.

Of these steps 2, 3, 8, 9 and 10 need some further explanation while the rest are the same operations as presented in the section on head tracking.

In step 2 of the algorithm we select the skin colored pixels in an \( n \times n \) region around the last known head location in \( I_c \). We experimentally selected this to be \( 30 \times 30 \) pixels large. We need to perform this step in order to make sure that we have the most up to date location of the user. Because the head tracking task runs only 5 times a second, it is likely that the user has moved since the last time the location was updated. We need to know precisely how far away the user is in order to perform background subtraction in step 4. The time penalty to performing this step is minimal as we only search a small region in the image and the benefits are large as a mistake in the user's position will cause the entire algorithm to fail.

In step 3, we compute the distance of the user from the camera. Some of the pixels in set \( N \) formed in step 2 could belong to the background. These pixels are outliers and we need to compute \( d_N \) robustly with respect to those. For obvious reasons taking the median distance of all the pixels that has a break-down value of 50% is the most obvious choice as opposed to for example taking the average distance of all the pixels in \( N \) that is sensitive to outliers.

Step 8 is a simple classification of the skin components to either belonging to the left or right hand. We compare the relative direction of each component to the face component as returned in step 7, to either being to the left or right of it
in the image plane. This classification derives from a simple assumption that if the user wants to direct the robot to a point on his left, he would do so with his left hand and vice versa. This assumption limits the range of available gestures but it specifies a constrain that is not too awkward to people.

Similar to the simple assumption guiding us in step 8, we remove the most unlikely components from the sets $L$ and $R$ by assuming that the average length of a person's arm is somewhere in the range of 0.5 to 1.2 meters. This assumption is valid with respect to our model of a person's height and size of head presented earlier (see Figure 5.14). At this point we expect that each of the two sets will either contain one element each corresponding to the left and right hands of the user or be empty. If a set is empty this means that the skin segmentation performed in step 5 did not detect the hand or that the hand is out of the camera's view because for example the user has it in her pocket.

Finally, in step 10 if both the $L$ and $R$ sets are non-empty we have to decide which hand the user is using to direct the robot. Obviously if one of them is empty and the other is not then that is the selected hand component and we proceed to compute the point on the floor; in this case the behavior returns a POINTING_LEFT or POINTING_RIGHT message along with the location on the floor. If both sets are empty then an NO_DIRECTION message is returned and no further processing is performed. For the case that both sets contain an element we need to decide which is the dominant direction. For this purpose, we compute the two direction vectors defined by the two arms. We then compute the angles that each vector makes with two planes $p_1$ and $p_2$. The first plane, $p_1$, is defined by the three points of the head and two shoulders. The second plane, $p_2$ is perpendicular to $p_1$. We first compare the angles that the two hand vectors make with $p_1$. If their difference
is larger than a certain threshold then we select the component corresponding to
the vector selecting the larger angle as the dominant. If not, then we compare the
angles that the two vectors form with respect to $p_2$. If one angle is larger than the
other by a given threshold then we proceed to select the corresponding component.
If not then we return an ambiguous direction message, LR_DIRECTION.

Figure 5.20 shows the intermediate steps and final results of running the al­
gorithm. This one example demonstrates many of the elements in the algorithm.
First, one can see from observing the input image data in part (a) that the user
need not be standing at some unnatural position in order for this to work; sitting
comfortably on your workstation works just as well as standing up for example. Sec­
ond, in this case both hands are clearly visible both in the input color images and as
regions after skin color segmentation. The algorithm has successfully discriminated
between the two and has returned the dominant direction, user pointing with his
left arm. We also see how the previously presented model of a person is correctly
applied to identify the shoulder points with respect to the face component’s location
in the image plane.

5.5 Navigation

The Navigation behavior (NB or just Navigator) is responsible for the safely moving
the robot in a dynamic environment. The Navigator acts based on stimuli originating
in the surrounding physical environment or as commanded by another behavior.
Figure 5.21 shows the SR-diagram for this behavior.

The input for the NB is one of the four commands MOVE_FORWARD,
MOVE_BACK, ROTATE_LEFT or ROTATE_RIGHT. An extension to these al­
 lows one to send the distance to move or the angle to rotate (in either direction.)
Figure 5.20: Example of gesture tracking, (a) the original gray scale image, (b) the original color image, (c) the binary image after background subtraction using depth segmentation, (d) the selected foreground pixels in the gray scale image corresponding to the binary image in (c), (e) the skin pixels detected in the segmented color image using (c) as guide and (f) the final result showing the direction selected, yellow line, along with the locations of the face component, green dot, and two shoulder points, yellow dots.
Additionally, one can just submit a goal location for the robot to go to. In that case the Navigator attempts to move to that location avoiding any obstacles in a safe manner both for the robot and the people in its path.

The output of the NB is one of GOAL.REACHED, GOAL.UNREACHABLE or OBSTACLE. Intuitively, the first message, GOAL.REACHED, signifies that either the robot has rotated by the given angle, or it has translated by the given distance or finally that it has reached the given goal location. The meaning of it depends on the command sent initially, i.e., one of the move or rotate commands. The second message, GOAL.UNREACHABLE, designates that no path to the given goal location can be found and no motion of the robot has occurred or will occur. This message is only sent as a response to a GO-TO-GOAL stimulus. The last message, OBSTACLE, is sent when the navigator detects a dynamic obstacle in its path. Later, we will elaborate further on the behavior of the navigator during motion and how it handles dynamic obstacles.

Figure 5.22 gives the diagram for the navigation behavior. More specifically, we have $S=(Q, \delta, q_0, F)$ where: $Q = \{ \text{idle, rotating}_{\text{left, right}}, \text{translating}_{\text{forward, back}}, \text{path planning, to next waypoint, waiting on obstacle} \}$ $q_0= \{ \text{idle} \}$ $F= \{ \}$ and $\delta$ is given in table 5.5. There is no final state for this behavior. The navigator remains in the idle state if it is not executing a given command. Also, at any one time, receiving a “stop” command causes the navigator to move from the current state to the “idle” state. We are not showing this transition in figure
Figure 5.22: The Navigation behavior FSA diagram
When the navigator is first started, it loads an occupancy grid map of static obstacles in the environment. An example of such a map can be seen in figure 2.4. The navigator uses this map for initial path planning when at the "path planning" state. It stores a copy of the original map in memory at all times. Obstacles that are detected and are not on this map are considered dynamic; a dynamic obstacle is defined as one that exists only temporarily at a given location. An example of a dynamic obstacle is a person walking in front of the robot. In contrast a static obstacle is defined as one that persists at a given location over long periods of time. Examples of static obstacles are walls and furniture such as table and chairs. So, if the obstacle persists for a given number of seconds, the navigator updates the obstacle (occupancy grid) map at the location of the object and then replans a path around it. When the goal location is reached or if no path to that location exists, the navigator returns to the "idle" state and resets the obstacle map to the original

<table>
<thead>
<tr>
<th>$\delta$</th>
<th>$q$</th>
<th>input</th>
<th>$\delta(q, \text{input})$</th>
</tr>
</thead>
<tbody>
<tr>
<td>idle</td>
<td>rotate-(left, right)-to-angle</td>
<td>rotating-(left, right)</td>
<td></td>
</tr>
<tr>
<td>idle</td>
<td>translate-(forward, back)-distance</td>
<td>translating-(forward, back)</td>
<td></td>
</tr>
<tr>
<td>idle</td>
<td>go-to-goal-location</td>
<td>path planning</td>
<td></td>
</tr>
<tr>
<td>rotating-(left, right)</td>
<td>not-at-target-angle</td>
<td>rotating-(left, right)</td>
<td></td>
</tr>
<tr>
<td>rotating-(left, right)</td>
<td>at-target-angle</td>
<td>idle</td>
<td></td>
</tr>
<tr>
<td>translating-(forward, back)</td>
<td>not-at-target-distance</td>
<td>translating-(forward, back)</td>
<td></td>
</tr>
<tr>
<td>translating-(forward, back)</td>
<td>at-target-distance</td>
<td>idle</td>
<td></td>
</tr>
<tr>
<td>path planning</td>
<td>at-goal-location or goal-unreachable</td>
<td>idle</td>
<td></td>
</tr>
<tr>
<td>path planning</td>
<td>go-to-next-waypoint</td>
<td>to next waypoint</td>
<td></td>
</tr>
<tr>
<td>to next waypoint</td>
<td>at-waypoint</td>
<td>path planning</td>
<td></td>
</tr>
<tr>
<td>to next waypoint</td>
<td>obstacle</td>
<td>waiting on obstacle</td>
<td></td>
</tr>
<tr>
<td>waiting on obstacle</td>
<td>timer-expired and obstacle</td>
<td>path planning</td>
<td></td>
</tr>
<tr>
<td>waiting on obstacle</td>
<td>timer-not-expired and obstacle</td>
<td>waiting on obstacle</td>
<td></td>
</tr>
<tr>
<td>waiting on obstacle</td>
<td>no-obstacle</td>
<td>to-next-waypoint</td>
<td></td>
</tr>
</tbody>
</table>

Table 5.6: The transition function $\delta$ for the Navigation behavior

5.22 or table 5.5 to avoid clutter.
One. This way, the navigator can avoid dynamic obstacles.

One critical issue is to correctly and efficiently detect obstacles blocking the robot's path. Our robot uses vision as the only distance sensor. The stereo maps computed by the Triclops system are processed to create what are referred to as the radial maps. An example of a radial map is shown in Figure 5.23. Radial maps are constructed by selecting the pixel closest to the camera for each column of the stereo image [TSM+97]. In that example we can clearly see the obstacle that is in the middle of the image at distance of 1.5 meters. We can also see the objects on the left of the image at 2.5 meters from the camera. There are no objects to the right of the robot. The radial map is searched and if an object is closer than a given distance then it is designated as an obstacle. The minimum distance is experimentally chosen so that obstacles are detected early and robustly given the limitations of the Triclops. In our case, we use a minimum distance of 45cm as we know that our Triclops is inaccurate for smaller distances. Additionally, we have limited the translation speed of the robot to a small value in order to make obstacle detection more robust, since stereo maps are updated at a rate of about 5 times a second. The robot’s translation speed is currently fixed at 350mm/sec.

The above process of detecting and avoiding obstacles is only used when the robot is in translation and not when it is in rotation. The reason for this is very simple. There is not danger to the robot when in rotation since it is already occupying the space within which it is rotating. The only danger to the robot while rotating can arise from an object thrown directly at it. However, since our robot has no arm manipulators, it cannot respond to such an event. So, in hopes that one will not try to throw an object at it we do not perform obstacle detection and avoidance in any case but when the robot is in translation. We realize that when
Figure 5.23: (a) Raw image data and (b) the radial map corresponding to the scene shown in (a)
the robot is in translation both stationary and moving objects present a hazard to it and both are handled equally using the procedure outlined.

5.6 Supervisor

All the Behaviors presented thus far operate autonomously waiting for a particular stimulus or stimuli to produce output. For our robot to behave intelligently, all the behaviors must somehow work together. The last behavior that we will present here, the Supervising Behavior (or SB) is the connecting block for all these pieces.

The Supervisor continuously monitors the output(s) of each of the Behaviors and decides what to do next, towards satisfying the high level task of communicating with a person. The supervisor uses sockets to communicate with the middle level behaviors allowing us to dedicate a separate workstation to it. Currently, the supervisor operates as a very large finite state machine. Ultimately, one would want a Supervisor that can learn from experience both in solving already known tasks but also in learning new skills. For now, this behavior can go into exploration and interact with the user through speech and gestures as presented earlier to navigate around its environment.

We will finish this chapter by presenting the Supervisor using our normal notation of an FSA diagram. We do not present the SA diagram but we note that all the inputs to this behavior are outputs of the middle-level behaviors. Additionally, the output of the Supervisor is any input to the middle-level behaviors that is not sensor or robot status data. The Supervisor never has to directly handle raw sensor data. This makes sense since it is the highest level of the hierarchy where the incoming information is the result of processing sensor data at lower levels. Figure 5.24 shows the organization of the behaviors into three levels with the Supervisor
Figure 5.24: Behavior organization chart. The Supervisor communicates with the behaviors in the middle level using sockets while the middle and lower level behaviors communicate using a shared memory mechanism.
Figure 5.25: The FSA diagram for the supervising behavior. Dashed lines indicate exclusive state transitions i.e., only one can happen.

The Supervisor is a rather large state machine and here we only show the major states and transitions between them. Figure 5.25 shows the FSA diagram for the Supervisor and Table 5.6 shows the corresponding transition function. This behavior is in one of three major states: exploring, interacting with the user, or idling. The robot will most likely be at the “exploring” state as long as there is space to explore; it interacts with the user when a person gets the robot’s attention and tries to direct it to a new location to explore. If there is not more unknown space
to explore or onboard resources are low, i.e., low battery voltage, the robot moves to the "idling" state and waits for the proper signals to move to one of the other states. While the robot is interacting with the user, it can be in one of several states i.e., "rotating to user", "locating user", "tracking user", or "getting user direction". When at the "locating user" state, the Supervisor may move to either the "idle" or "exploring" state when a failure occurs. A failure in this case is the inability of the "user finding" behavior (see Section 5.3) to locate the user. The supervisor will report this to the user and he may ask it to try again in which case the robot remains in this state and the cycle repeats. When failure occurs the supervisor returns to the "idling" or "exploring" state depending on which state it originally transitioned into the "rotating to user" state.

There is plenty of room for improving the Supervisor. Currently all transition between states are fixed and never change. It is very difficult for the robot to learn a new skill even if the appropriate set of middle-level behaviors is available. Learning

Table 5.7: The transition function $\delta$ for the Supervising behavior
a new skill requires re-writing the Supervisor software and changing many of the states and transitions between states. Also, at this point there is no room for taking into account the uncertainty involved with any of the actions that the Supervisor takes. Future work will focus on developing a supervising behavior that is easier to extend and can learn through experience.
Chapter 6

Conclusion and Future Work

This thesis focused on developing a human-robot interface based on vision and speech designed to allow a person to direct a robot to specific locations in its surrounding environment. We implemented the system using a behavior-based architecture utilizing a set of behaviors that were previously developed in our lab and a set of newly developed behaviors tailored to this application. We added behaviors for locating and tracking users, gesture recognition, sound localization and high level artificial intelligence. We also improved upon the existing navigation behavior. The system developed runs in real-time and performs robustly in dynamic environment with noisy sensor data.

As a result of this work, we have added four new sensors to the robot, one visual (a color camera) and three audio (wireless microphones used for sound localization and speech recognition.) All the newly developed and improved behaviors are designed to run at a minimum computational cost and to take advantage of all the available sensor data including color and stereo images.

The resulting interface still constrains the user in terms of what he can say or where he can stand with respect to the robot. However, verbal commands have
been selected to be close to natural form so that a non-computer literate person can easily learn them. Additionally, most of the restrictions on the vision-based behaviors are the result of the limited field of view of the robot's on-board cameras and limited CPU power. We are looking forward to addressing hardware issues by replacing the robot's camera with a 2-camera stereo unit on a pan-tilt base. This will most closely mimic the mechanics of the human visual system allowing the robot to focus its attention to different parts of the world quickly without change in its position.

The behavior-based control architecture addresses the problem of limited on-board CPU power by off-loading some of the computational load to remote workstations connected on a Local Area Network. Currently, the on-board wireless network that allows the robot's computer to communicate with remote workstations is limited to a maximum throughput of 10 Mbps which often is a lot less depending on the quality of the radio signal. The radio signal is sensitive to noise originating from electronic equipment in the surrounding environment as well as the type of obstacles the signal has to travel through. The inherent restriction to the available bandwidth makes transmitting the images captured with the robot's cameras to remote workstations in real-time unfeasible. For this reason all the middle-level behaviors must execute on-board the robot. In the future, we must explore the option of compressing the images before transmission and upgrading to a faster wireless modem. Recently, we improved our software to transmit in real-time the 2D occupancy grid maps, the radial data and odometry information.

The behavior-based control architecture allowed the timely completion of this work. Each of the new behaviors was implemented and tested individually and later added to the whole system by making the supervising behavior aware of it. Adding
new behaviors or using the existing ones to build new systems is straightforward with the need only to change the Supervisor. Handling data from different type sensors was also facilitated by the use of this architecture making it easy to build a system with a varying number of data sources.

The ideas and some of the behaviors presented in this work were subsequently used to develop the control software for a robot taking part in the 2001 AAAI (American Association for Artificial Intelligence) international robotics competition [EHL+02]. The theme of the contest was that of developing a mobile robot capable of serving Hors D'oeuvres during receptions, navigating a dynamic environment, locating people and interacting with them. Our robot won first prize.

Currently, we are in the process of developing behaviors for topological mapping, face recognition and natural language understanding that will allows us to personalize the experience of interacting with the robot. In the future, we will focus on improving the supervising behavior to be easier to extend and even add mechanisms for modeling the uncertainty in the robot's actions and its knowledge of its surrounding environment. We also plan to experiment with algorithms within the Supervisor to allow online learning of new tasks and mechanisms for adding new behaviors to enable new skills while the robot is running.
Bibliography


