Mabel: Extending Human Interaction and Robot Rescue Designs

Thomas Kollar, Jonathan Schmid, Eric Meisner, Micha Elsner, Diana Calarese, Chikita Purav, Chris Brown
Jenine Turner, Dasun Peramunage, Gautam Altekar, and Victoria Sweetser
University of Rochester
Computer Science Department
PO Box 270226
Rochester, NY 14627
brown@cs.rochester.edu, tk010j@mail.rochester.edu

Abstract

Mabel (the Mobile Table) is a robotic system that can perform waypoint navigation, speech generation, speech recognition, natural language understanding, face finding, face following, nametag reading, and localization. Mabel can interact intelligently to give information about the conference to patrons. Major additions to this year's design are Monte Carlo Localization, Filter-Cascade techniques for vision applications, and an improved robot search and rescue system using a 3D OpenGL mapping system. Mabel was the winner of the 2003 robot host event and tied for third place in the robot search and rescue event at IJCAI 2003 in Acapulco, Mexico.

Introduction

The American Association for Artificial Intelligence (AAAI) holds robotic competitions each year at its annual conference. At the 2003 conference, there were the robot host, robot search and rescue, and robot challenge events. The robot host event involved giving people information about the conference schedule over a large area. The robot search and rescue event involved a robot entering a mock disaster scene, locating mock human victims, mapping the victims' location, and returning safely out of the scene without causing damage to the arena. Finally, the robot challenge event required a robot to be dropped off at the conference center, navigate its way to the registration desk, register for the conference, navigate to the auditorium, and give a talk on itself.

Mabel, a robotic system developed mainly by undergraduates at the University of Rochester, competed in the robot host and robot search and rescue events. A picture of Mabel as used in the robot host event can be seen in Figure 1. Mabel serves information to conference patrons by using waypoint navigation, Monte Carlo Localization, speech generation, speech recognition, natural language understanding, face finding, face following, and nametag reading. Moreover, with a change of body, Mabel can also find victims in a mock disaster scene.

Mabel competed at AAAI 2002, and many of the same philosophies carried over to IJCAI 2003. The overall design philosophy for Mabel was to integrate human interaction with robotic control. We achieved this goal by using an interactive speech system, a pan/tilt/zoom camera that actively follows faces, and another pan/tilt/zoom camera that reads patrons' nametags. In this paper, we present the changes that have occurred since AAAI 2002 (Schmid, Kollar, et. al., 24-31). Moreover, we will discuss the methods that we used to integrate the hardware and software into a usable system. We will again discuss what worked and what we can improve upon in the future.

Figure 1: Information Serving Configuration for Mabel

Hardware and Software

The devices available for our use included: an ActivMedia Pioneer 2 AT mobile robot, an EVI-D30 Sony Pan Tilt Zoom camera, a Canon VC-C4 Pan Tilt Zoom camera, a directional microphone, a ViewSonic AirPanel, and a 1.3 GHz Pentium IV IBM laptop. A custom body was built on top of the robot, as can be seen from Figure 1. An extra sonar ring was added just below the keyboard to give extra range readings.

The effectors this year included speaking, nametag reading (and speaking), a pan/tilt/zoom camera, and the wheel

motors on the robot. The sensors of the robot include sonar range detectors, wheel counters, two pan/tilt/zoom cameras, and a microphone. The pan/tilt/zoom camera under the monitor reads nametags while the camera above the monitor searches for and follows faces. An upgraded facefinding algorithm allows the robot to search for people while moving, since it no longer uses motion.

The libraries and software packages that were used include: the Intel Image Processing Library, the Intel OpenCV library, CMU's Sphinx speech recognition system, OpenGL 3D modeling library, Microsoft text to speech SDK, Microsoft DirectPlay, Microsoft DirectX, and ActivMedia's ARIA robot control language (IBM Viavoice, web) (Intel OpenCV, web) (ARIA, web). Much of this project was programmed in the Python programming language. Any code that was done in C++ was interfaced into Python using the Simple Wrapper and Interface Generator (SWIG, web).

Thus, two different systems were built using these tools. One is an information serving robot and the other is a rescue robot. We will discuss what these systems do at a high level now.

Information Serving

A general overview of the architecture for the information serving robot can be seen in Figure 2. One can see that there are multiple layers in this system: the sensor level, the interpretation level, and the decision level. The sensors include sonars, cameras, keyboard, the dead reckoning position and a microphone. The drivers for using these sensors, as well as a library for writing behaviors, were provided with the ActivMedia robot (ARIA, web).

Moreover, we produced an a priori map of the competition areas using a tape measure. On this map we drew a graph of places that the robot could go from any given place. This map was not only used to visually direct people from one place to another at the conference center, but it was also used for localization purposes.

At the next level, the interpretation level, the programs would interpret the sensors to provide high-level assessments about the environment and intentions of the person with whom the robot is interacting. For example, the vision component would tell whether or not there was a face in the scene (and its location) and it could also determine what the name of the patron was (based on their nametag). The intentions of the patron could also be determined from dialog. For example, the robot could leave a conversation if they said "goodbye".

Thus, at the top level is the control aspect (which is basically a finite state machine). If the robot gets input that a person is present in front of the robot then it starts dialog with them. In our system, a person can be present if they speak, type input, or if their face is being tracked by the camera and they are close enough to the robot. If a person is not detected by the robot then it will wander around to various points on our map of the competition area (and would localize at the same time), all the time searching for people. Should it find a person, then it will continue driving toward them until it either finds them, or it will go back to wandering points on the map.

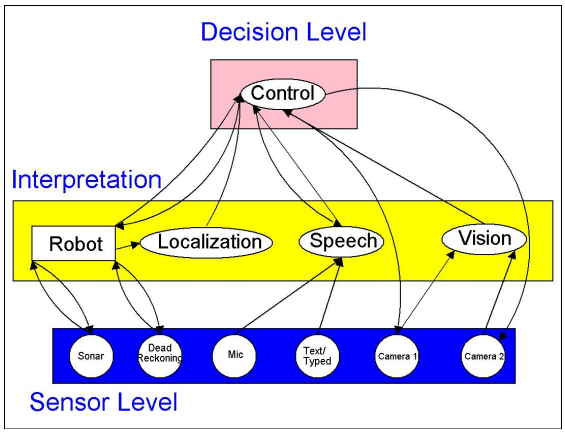

Figure 2: Mabel System Structure consists of three layers: sensory input and output, interpretation of sensor data and decisions about what to do given that data.

Whenever the robot was wandering it would use Monte Carlo Localization to determine its location on the map. This technique uses filtered sonar readings and the internal position of the robot to correct the error that occurs in robot movement. This technique will be discussed in depth later in this paper.

When a person was interacting with the robot, many things would happen. Immediately, the robot would start trying to find their nametag. Once it had attained that information it would use the information when speaking or displaying other information on the screen. At the same time, the robot would continue to track the patron's face with the higher camera. Anytime that someone spoke to it, it would parse the words using Sphinx speech recognition and it would then use a Bayes net to decide what their intentions were when talking to it. Given this information, and a query that was done on a database of events, a response about an event, speaker, time that an event occurred, time of day, or date would be spoken and printed to the screen. Moreover, if any events matched the query, then they would appear on the screen.

Robot Search And Rescue

Using the robot search and rescue system, one can teleoperate the ActivMedia Pioneer 2AT robot to locate and map victims in a mock disaster scene and provide a map of victim locations to mock rescuers. The robot and its sensors are the only things that the teleoperator can use to navigate the scene. Points are awarded in the competition for successfully locating victims, and creating useful maps of the disaster area. Points are deducted for making contact with victims and for interacting with the environment in a dangerous manner, i.e. causing secondary collapses of structures.

The hardware required for this included an ActivMedia Pioneer 2AT mobile robot, a small body, a flashlight, a laptop, a webcam and an ethernet connection to the base station. One can see the design in Figure 4. The ethernet cord produced a better connection than wireless ethernet could have

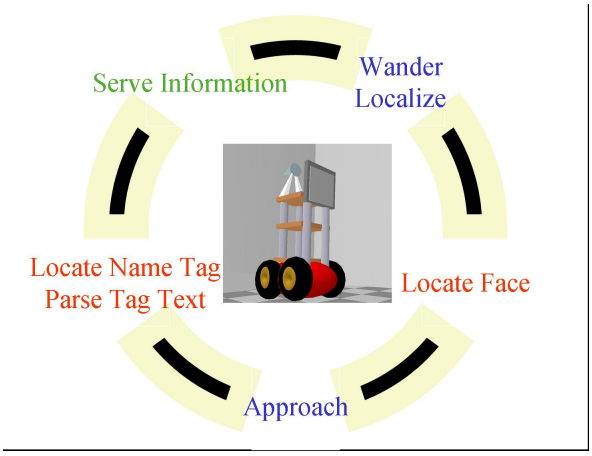

Figure 3: Sample interaction for the information server.

Figure 4: Robot Search And Rescue Configuration for Mabel

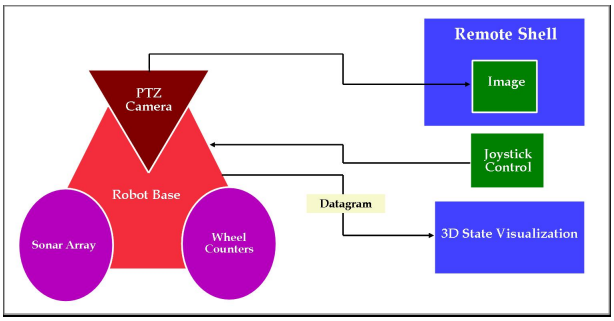

Figure 5: Overview of the Robot Search And Rescue system.

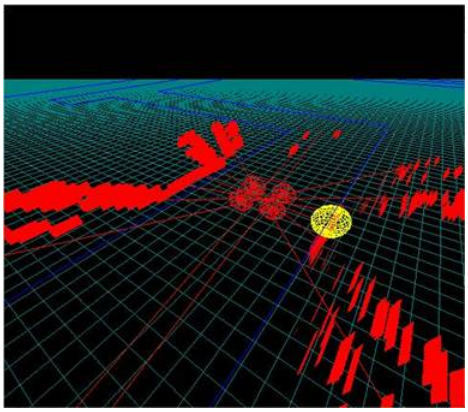

Figure 6: Real-time 3D mapping aids the teleoperator in robot search and rescue. The red lines are sonar readings and the yellow sphere is a mapped victim.

provided. The video was cleaner, the connection was faster, and there was always a reliable connection to the robot.

Our system consists of three core components: the control, mapping, and joystick applications. An overview of this system can be seen in Figure 5. The control application, which interfaces directly to the robots actuators and sensors, is responsible for obstacle avoidance, linear and angular velocity control and communication of sensor information back to the user workstation. This program was running on the robot and was at the core of the system. The teleoperator would never have to touch this program.

The mapping application is one of the teleoperator's tools for retrieving information from the disaster scene. It provides the user with a live interactive 3D rendering of the robot environment by mapping the raw sonar readings and the internal robot position $< x, y, \theta >$ to a 3D world. This rendering was done using OpenGL. One can see an example of the mapping interface in Figure 6. The teleoperator was also able to load maps of the environment or save maps for printing.

The teleoperator would use a joystick to control the robot's movement, where the buttons would plot victims' locations or change whether the robot should be in an autonomous state or in a teleoperated state. Input from the joystick was obtained from Microsoft's DirectPlay library. This input would then be transmitted over the network to the control application, for use in controlling the movement of the robot.

Moreover, we had many applications that needed to be run simultaneously. The python programming language allowed us to start and stop all of these with the click of a button, thus streamlining the start procedure and reducing startup time. Moreover, python helped to make the system robust to network failures or other catastrophes where we might lose a connection with the robot.

Monte Carlo Localization

Localization is the problem of determining the position of the robot given its sensor readings (vision, sonar, and dead reckoning). It is implemented here using a technique called Monte Carlo Localization. In this project a standard Monte Carlo Localization algorithm with distance filtering was implemented as per (Thrun et. al., 99-141). As in that work, we are only concerned with the readings from the robot's wheel encoders and range devices.

Background for other Techniques There are various forms of localization that vary in the ambition of the problems that they attack. Local techniques aim to correct drift from the inaccuracy of dead reckoning (the most well known of these are Kalman Filters). Global techniques aim to locate the robot even when catastrophic things happen. For example there are two problems usually solved by the global techniques: the wake-up robot problem and the kidnapped robot problem (Fox et. al, 391-427)(Thrun et. al., 99-141). The wake-up robot problem occurs when the robot is given no initial information about its position and is expected to localize itself in some environment. The kidnapped robot problem occurs when the robot is carried to another location during its operation and is expected to recover from this strange happenstance where the dead reckoning does not match up with its range data. Thus, global techniques must be more powerful than local ones.

Localization is often considered to be one of the most fundamental problems of mobile robotics (Fox et. al, 391-427)(Thrun et. al., 99-141), since without it a robot will have an inaccurate internal representation of its current position. This can lead to some disastrous consequences, especially if there are stairs, ledges, or other obstacles that cannot be readily seen by the range devices and which could have been avoided if the perceived and actual positions coincided. Thus, maybe the robot will run off a cliff, fall into a hole, or run into something that it could have otherwise avoided. This is not to say that an internal map is needed for localization from the start. (Montemerlo et. al., 593-598) gives a nice solution to the simultaneous mapping and localization problem (often abbreviated SLAM).

There are a variety of ways to localize a robot. Some of these include: Kalman Filtering localization, Markov localization and Monte Carlo Localization. Moreover, we will go in depth into the Monte Carlo localization, since this is the technique used in this project.

Kalman Filter localization The Kalman filter cannot solve any of the global localization problems. However, it does a good job of tracking the location of a robot, given an original known location. Kalman filtering presumes that inaccuracies in position over time can be modeled by a unimodal Gaussian distribution (one that has only one peak). Thus, the Kalman filter only has one hypothesis as to the location of the robot. Thereby, this version of the Kalman Filter is inadequate as a solution to the global localization problem (Fox et. al, 391-427).

However, the Kalman Filter can be extended to have multiple hypotheses. These represent beliefs using mixtures of Gaussians, instead of a single Gaussian, thereby allowing the robot to pursue multiple hypotheses (Thrun et. al., 99-141). (Jensfelt and Kristensen, 2001) even uses topological world models and landmarks, along with a multi-hypothesis Kalman filter to localize the robot. The latter does not take raw sensor input, but relies on higher level features to localize the robot. It would nice if the localization could be a black box to an end user. This would not be the case with the method described by (Jensfelt and Kristensen, 2001).

Figure 7: The MCL(X,a,o) from (Thrun et. al., 99-141), a is any odometry reading, w_i are weights for x, and o are the range sensor readings

$X' = \oslash$
for $i = 0$ to m do
  generate random x from X according to $w_1, ..., w_m$
  generate random $x' \sim p(x'|a, x)$
  $w' = p(o|x')$
  add($< x', w' >$ to X'
end for
normalize the importance factors w' in X'
return X'

Markov Localization The mathematical framework for Markov Localization is based around the Markov assumption, which in this case will state that the robot's location is the only state in the environment that affects sensor readings (Fox et. al, 391-427). Of course, in a dynamic environment, this assumption is invalidated. The way that most researchers get around it is by using filters on the sensor readings as in (Fox et. al, 391-427).

In Markov Localization there is no single hypothesis as to the current location of the robot, but a probability distribution over the state of all such hypotheses. A large state space can be used to store and retrieve the belief of the robot being at a given location. The probability distribution is stored in a very large three dimensional grid with axes x,y, and $theta$. For example, if we have a $100x50$ meter space with a 3 degree accuracy of theta and a 5cm accuracy of position in x and y , then we have a storage requirement of $6 * 10^9$ cells. In the naive implementation the robot would have to update all of these cells each time it received new information from the environment. For a large state space this takes too much time. Thus, specialized techniques have to be incorporated into these algorithms so that all of the cells do not have to be updated after every sensor measurement (Fox et. al, 391-427). There are also other methods of storing this state space. One of the methods uses oct-trees to resize the cells dynamically, thereby reducing their number(Dellaert et. al., 1999).

Either way, Markov localization provides a way to have multiple beliefs about the location of the robot. Moreover, it can solve the global localization problem. After applying some filters to the range sensor data, we can perform Markov Localization in a dynamic environment. The filters discussed in (Fox et. al, 391-427) include a distance filter, where range sensor readings from unmodeled obstacles are disregarded, and an entropy filter, where the robot only takes sensor readings that make it more sure of its position.

The Monte Carlo Localization Algorithm

MCL is a relatively new method that provides a solution to the global localization problem and was implemented for the Mabel system. We will discuss the algorithm in depth here. We implemented the algorithm in Figure 7 with distance filtering as per (Thrun et. al., 99-141). The major benefits of using MCL over the other approaches are the following: it can globally localize the robot, it greatly reduces the mem-

ory requirements in comparison to Markov Localization, it is more accurate than Markov Localization, and it is implemented easily (Fox et. al., 1999).

For MCL, we divide the new sensor data into two groups: a new odometry reading and a new range sensor reading. $S = s_i | i = 1, ...N$ is a set of N weighted and random samples over a space. For example, in our implementation of Monte Carlo Localization this distribution would initially be uniformly distributed. Each sample is a two-tuple with a pose and a probability of that pose. Thus, a sample is:$<< x, y, \theta >, p >$. Moreover, we assume $\sum p_i = 1$.

Each time the robot gets a new odometry reading a, MCL generates N new random samples that approximate the robot's new position. Let s_i be a given sample from S and l' denote its position. According to (Fox et. al., 1999), "Each sample is generated by *randomly* drawing a sample from the previously computed sample set [S], with likelihood determined by their p-values [probability]." Thus, the value for the new sample's l can be generated by sampling according to $P(l|l', a)$.

In our implementation this probability ($P(l|l', a)$), often called the motion model, is computed by taking the ideal movement of any sample, and sampling a new position with a probability coming from a Gaussian distribution in the rho and $theta$ components of this movement. The standard deviation and the mean of this distribution were computed from experimentation.

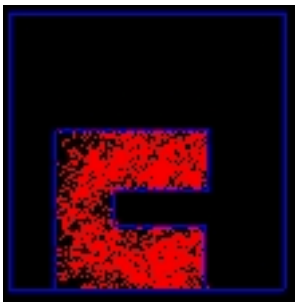

Figure 8: Uniform distribution over the space.

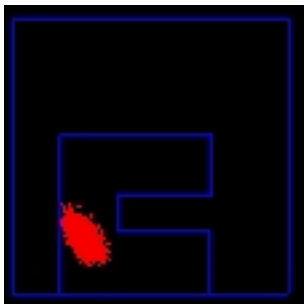

Figure 9: Convergence of MCL on the location of the robot.

Moreover, for a new sensor reading s and a normalization constant α that enforces $\sum p_i = 1$, we re-weight the sample set S. We let $< l, p >$ be a sample and we recalculate p such that $p \leftarrow \alpha P(s|l)$. This can be done in $O(N)$ time according to (Fox et. al., 1999).

The probability $P(s|l)$ is often called the sensor model. In our implementation it is merely a Gaussian distribution over the ideal sensor reading of the robot given that sample's location. The $P(s_i|l)$ is measured by ray-tracing to get the ideal reading of the ith sensor and then sampling from the Gaussian using the difference of the ideal and measured readings. Moreover, we integrate these probabilities for each sonar by multiplying them together. This integrated probability gives us $P(s|l)$. The standard deviation and the mean of this Gaussian distribution were derived from experimentation.

There are some cases where we run into problems with Monte Carlo Localization. First, since MCL uses finite sample sets, it can (and sometimes does) happen that none of the samples are generated close to the robot position (Fox et. al., 1999). This causes the robot to lose its location, never to recover. There are some techniques to prevent this from happening. However, many of these methods are not necessarily mathematically sound (Thrun et. al., 99-141). The solutions usually involve introducing artificially high amounts of noise into the sensor measurements or by generating samples by the most recent sensor reading. We use a simple technique of generating random samples around a sample s_i using a progressively larger Gaussian distribution, should the hypothesized position move into an unmodeled space. We would also take some uniformly distributed random samples on each round of MCL.

There is also the problem of the dynamic environment. Regarding our assumption that the environment is static, (Thrun et. al., 99-141) says, "Clearly this conditional independence can be violated in the presence of people (which often block more than one sensor beam). In such cases it might be advisable to subsample the sensor readings and use a reduced set for localization (Fox et. al, 391-427)." Thus, he suggests that we subsample the sonar readings using a distance filter or an entropy filter to get the readings that correspond to the world, and not to the dynamic obstacles in the world. In this project, we used the distance filter to rule out sensor readings that come from unmodeled obstacles, as per (Fox et. al, 391-427).

Finally, there is the problem of the map that is used as a model of the world. In our system, we used a simple mapping system that only allowed lines. Moreover, we had no way of automatically generating these maps from real data. In other words, we used a tape measure to make the maps by hand. Now when one is using a simulator there are no problems and everything works as expected, since your model of the world matches exactly with the world used by the simulator. However, when one is using MCL with real world data, then the data will often not match the modeled one due to sometimes quite large errors in generating a map by hand. This is a problem that we didn't have time to solve, and thus our algorithm would have many problems when working in the real world. In the simulator, however, the convergence and tracking of MCL worked very well.

Speech

The communications system is responsible for interacting with patrons by finding out which information they want, retrieving it if possible and giving it to them. The various inputs and outputs can be seen in Figure 10. It supports three input modes; speech recognition, typed input and graphical input via a touch screen.

Interaction is turned on whenever the robot recognizes that someone is in front of it, when someone speaks to the robot loudly enough to trigger speech recognition, or when someone types or clicks in the GUI. The communication system never initiates utterances of its own; it responds to user's input by directly answering their questions, then waits for more input. Interaction turns off when the person stops using input functions and walks away.

The robot handles typed input by filtering out keywords relevant to the conference, then using a Bayesian tagger to find the type of information the user is asking for. This algorithm is essentially unaltered from (Schmid, Kollar, et. al., 24-31). The graphical interface transforms clicks in the window to tags and filtered output identical to the result of Bayesian filtering. The system then processes both typed text and clicks identically, as in (Deneke, 1997).

After tagging, the robot immediately handles some conversational pleasantries with randomly chosen rote responses. This allows it to deal with 'Hello', 'Thank you' and other statements that are not requests for information. Real questions are turned into SQL queries that retrieve information from a database. The database was hand-constructed from the conference schedule, and includes the same information as the schedule.

The robot uses graphical display, text display and text-to-speech as output modes. Graphical output is an unranked table containing the information the user requested. The user can click on any item in the table for more information.

Text display uses a natural language generation algorithm. This algorithm first constructs a core sentence of the form:

$< speaker >< verb >< event >$

It guesses the verb by examining the event, and if there is no speaker, it uses an expletive construction such as 'there was' to mimic the same basic form. Then it recursively adds prepositional phrases until it has used up the remaining information. If there is too much information to fit on the screen, it omits some and adds a phrase containing 'more' or 'others' to the end of the sentence. Text to speech speaks the same output as is displayed graphically; it uses Microsoft Speech SDK.

Because of the technical problems inherent when using speech recognition in noisy settings with unfamiliar speakers, speech recognition was handled separately. It uses CMU Sphinx as in (Schmid, Kollar, et. al., 24-31). An example conversation can be seen in Figure 11.

Vision

Mabel the mobile infoserver's vision system focused on finding people in the environment, and reading their nametags (see Figures 12 and 13). Both techniques made use of a general filter cascade architecture that searched for

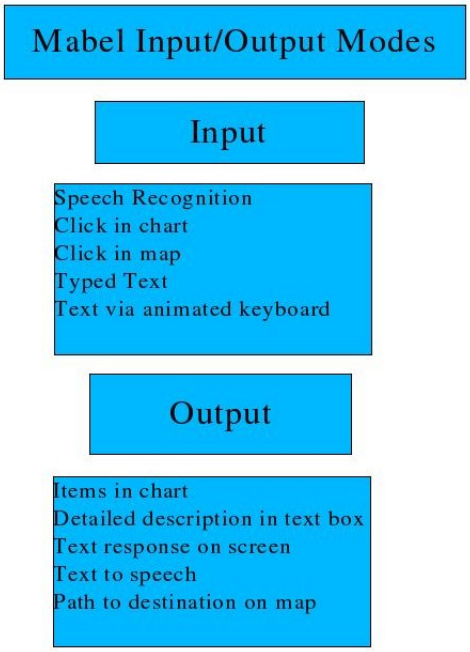

Figure 10: The inputs and outputs for the speech system.

a pattern throughout the image using a constant size detection window. The filter cascade detects a single rectangular bounding box around the largest target pattern in the source image, which can then be tracked over multiple frames with an Alpha-Beta filter to smooth over any noisy or incorrect detections. For a detailed description of the Alpha-Beta filter see the next section.

A filter cascade is a search technique that focuses its searching effort on small areas of the image that show initial promise for matching the target pattern. These small areas of the image are called sub-windows and the initial promise of a sub-window is determined by the success of the first classifier. If a sub-window of the image passes the initial classifier, then additional classifiers are applied to it. We apply a cascade of these filter-classifiers to each sub window, testing for the presence of several simple rectangular features. See (Viola and Jones, 2001) for a detailed description of this process. While most filter cascade architectures to date have utilized only features within grayscale images, we apply color models to create additional feature channels as shown in the two central images of Figure 14.

While searching through a feature channel image, the sub-window often detects a spatial pattern in several adjacent locations. Also, the target pattern might appear in more than one location in the image. Jones and Viola perform a disjoint-sets algorithm to combine locally adjacent detections and select the largest target region. Our filter cascade algorithm instead fills in each detected rectangle with 1s in an initially empty binary image. This binary image shares the same dimensions as the signal images and the starting color image. A contour finding algorithm is applied to the

Hello, Mabel.
MABEL: Hello.

Where is Henry?
MABEL: There are two events that will occur given by Henry.
Please look below for more information.

(Chart display shows : Cholula 7, Cholula 3)
<click on Cholula 7 in chart>

MABEL: (Chart display shows data on the event in Cholula 7;
map display shows route from robot location to
Cholula 7)

What is the keynote address?
MABEL: Keynote address–Computer Vision–AI or Non-AI Problem is
held in the Juan Ruiz Alarcon Theater of the Acapulco convention
center at 9 AM on August 12.

(Chart display shows details of the event.)

Will you marry me?
MABEL: Can you try to be a bit clearer?

Bye.
MABEL: Have a nice day, bye.

Figure 11: Sample conversation.

binary image. The largest contour from this detection image becomes the filter cascade target. Adjacent detections from the source image overlap when filling the binary image, and thus form only a single contour. See the bottom of Figure 14. Spurious detections from other similar objects tend to create smaller contours and are thus often ignored.

The person-finding algorithm used both a skin colored channel and an intensity channel (see the right of Figure 14) for locating faces in the detection window. To generate the binary skin channel (where 1s represent skin pixels), we test for the presence of each pixel from the image in a binary Hue/Saturation model (see the left of Figure 14).

We first generate this model from images of human skin tones captured in the environment by using a series of pictures that were taken of various individuals. The skin regions of each individual were then isolated. To make the model robust to all individuals, a sample of different skin pigmentations was carefully selected. From the training set of skin images, the value of the color in each pixel was calculated using the HSV scale and plotted on a Hue/Saturation graph. We save this as a bitmap so that we can fill in missing skin tone regions using a standard image-editing program. This improves the robustness of the skin detection.

The first level of the filter cascade for faces drags a sub-window throughout the binary skin-tone map. Sub-windows are eliminated as possible faces if the sum of their pixels is not over 25% of the sub-window, a result that would suggest that there are insufficient skin pixels on the object to deem it a face. The second and third levels both operate on a grayscale intensity image. In the second level of the filter cascade, we look for the eyes–a trait that distinguishes faces

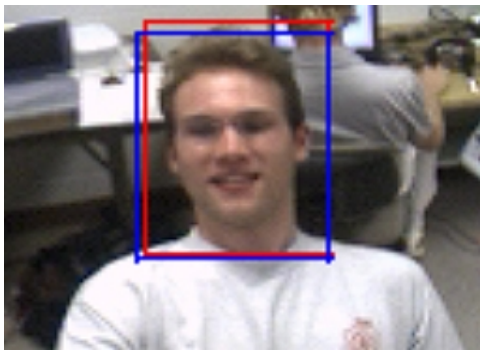

Figure 12: This demonstrates the finding of a face.

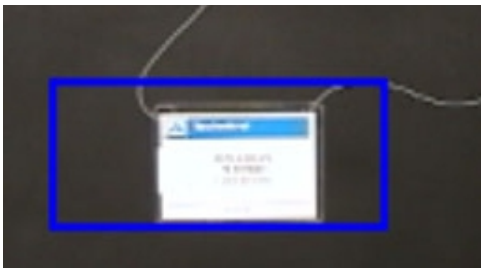

Figure 13: This demonstrates the finding of a nametag.

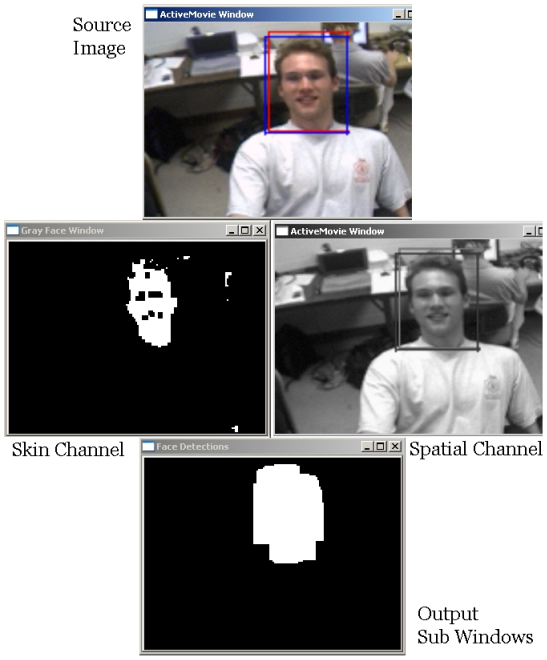

Figure 14: The process of finding a face.

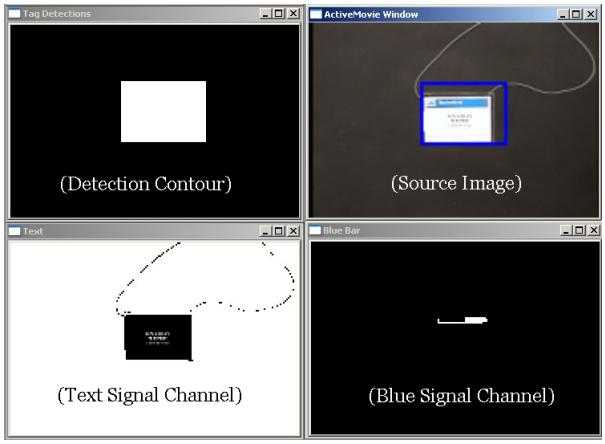

Figure 15: This demonstrates the process of finding the nametag.

from other objects in most cases. We look for the eyes by (1) summing the number of skin-tone pixels within a rectangle covering the forehead, (2) summing the number of skin-tone pixels within a rectangle covering the eyes, and (3) subtracting the result of the second step from the result of the first step. If the result of the subtraction is a large positive number (thereby suggesting the the rectangle convering the forehead and the eyes describe two dissimilar entities), then we gain confidence that the sub-window spans a face. If the sum tends to be near zero , then we lose confidence that the sub-window spans a face and we terminate the cascade. Note that the result from step (3) must constitute at least 8% of the sub-window to allow the cascade to continue. Intuitively, this requirement captures the idea that the forehead should consist of many skin tone pixels and the eye region should consist of no skin-tone pixels (thereby producing a high number in the subtraction of step 3). In the final level in the cascade, we compare a rectangle covering the area of the lips to the chin below in a similar manner as above. In this case, the chin region is subtracted from the darker lips region above it. We have empirically found that this technique works well.

The nametag reading process employs two different zoom levels using a Canon PTZ camera. The central control system activates the tag reader during the person approach phase. The tag is found at this outer zoom level using the filter cascade, and the camera is centered on its position. When the alpha-beta filtered tag location is centered in the image, the camera zooms in. When the zoom is completed, the image resolution is increased from the usual 180x120 to a full 760 x 480. Ten frames of the nametag are stored at high resolution and read by the Abby Fine Reader engine. The most frequent occurrence of the first two string tokens are assumed to be the person's first and last names.

The nametag filter cascade consists of four levels with each level paying close attention for the presence of a particular feature. The cascade's first level eliminates all filter sub-windows that lack a high percentage of white pixels. In most instances, the first level eliminates close to half the sub-

Figure 16: This shows how the nametag picture would have been rotated before processing.

windows in the image, thereby narrowing down our options significantly.

The second level checks for the presence of a colored bar at the top of the nametag (see the bottom right of Figure 15). We distinguish a bar from other objects through the use of a hue/saturation color model for the bar, which was created by sampling several images containing the bars. Once the second level is complete, the majority of the sub-windows are centered vertically on the tag, but remain uncentered in the horizontal directions (i.e., to the left and to the right).

The third and fourth levels attempt to horizontally center sub-windows on the tag. The third level begins the process by (1) summing the pixels in the sub-window containing the tag text, (2) summing the pixels in the sub-window to the left of the tag text, and (3) subtracting the result of step 2 from the result of step 1. If the result of the subtraction in step 3 is a high number, then we gain confidence that the sub-window is adequately centered from the left of the tag. If the result of the subtraction is a low number, then we lose confidence that the sub-window is adequately centered from the left. In the fourth level of the operation, we perform the same series of steps as done in the third level. This time, however, we consider the sub-window to the right. If the cascade passes the fourth level, then we can be reasonably confident that the sub-window is centered on the tag.

In order to ensure quality text reading, we implemented rotation invariance on the high resolution frames of the nametag. This is done by calculating the angle formed between the blue rectangular bar and the bottom of the image. The image is rotated to cancel the calculated angle (see Figure 16).

Alpha-Beta Filter

Let X be the object's position along the x screen dimension and Y be the object's position along the y screen dimension. Then let $\hat{X}$ and $\hat{Y}$ denote the object's velocity along the x and y screen dimensions, respectively. Then we can represent the entire state of an object being tracked with the vector $\mathbf{x} = [X, \hat{X}, Y, \hat{Y}]$.

The $\alpha - \beta$ filter for state prediction has the form

$$\hat{\mathbf{x}}(k+1|k+1) = \hat{\mathbf{x}}(k+1|k) + \begin{pmatrix} \alpha \\ \beta/\Delta t \end{pmatrix} D \qquad (1)$$

where $\hat{\mathbf{x}}(k+1|k+1)$ is an updated estimate of $\mathbf{x}$ given $\mathbf{z}$(k+1), the measurement of the object's tracking state at time k+1. We assume that $\mathbf{z}(k+1)$ consists of the two position components (X, Y) but not the velocity components

$(\hat{X}, \hat{Y})$. The state estimate (i.e., prediction of an object's position and velocity in the next frame) is the sum of the state $\hat{\mathbf{x}}(k+1|k)$ predicted from the last estimate and the weighted difference between the actual measurement and the predicted measurement (Brown, 1995). The difference between the actual measurement and the predicted measurement is know as the innovation and is denoted by

$$D = \mathbf{z}(k+1) - \hat{\mathbf{z}}(k+1|k) \tag{2}$$

The weight is a vector consisting of α, β and Δt components. The values for α and β can be derived from

$$\alpha = -\frac{\lambda^2 + 8\lambda - (\lambda+4)\sqrt{\lambda^2+8\lambda}}{8} \tag{3}$$

and

$$\beta = \frac{\lambda^2 + 4\lambda - \lambda\sqrt{\lambda^2+8\lambda}}{4} \tag{4}$$

where λ is the object's maneuvering index. For our tracking uses, we experimentally determined the optimal value of λ to be 0.2. Finally, we let $\Delta t = 1$ since the time difference between observations is 1 image frame.

Discussion

During the fall of 2002 and the spring and summer of 2003, there about six people working on the project full time. Thomas Kollar focused on localization, navigation, speech recognition, TTS, robot search and rescue, and global integration. Eric Meisner and Chikita Purav also worked on the MCL algorithm. Jon Schmid, Dasun Peramunage, and Gautam Altekar focused on the vision components of the robot. Micha Elsner, Diana Calarese, and Jenine Turner worked on the natural language understanding and GUI. Eric Meisner worked on mapping and robot search and rescue designs.

The whole system was developed in a Microsoft Windows lab and with a common CVS tree. CVS made integration fairly painless. Since programs in Python have fast development time, it proved to be a very useful tool this year. Python is a high level programming language and can interface to C++ or C code by using a program called SWIG. Any code that needed to be fast and less processor intensive, we wrote in C++. Then, using SWIG, we would use those functions in our python program. Thus, the vision code, robot control code, and the interface to the speech recognition and text to speech engines were all written in C++. The GUI and the decision making were implemented in Python and interfaced to all the other parts through SWIG. In fact, our design was also simplified to not even need a network connection to communicate.

Acknowledgments

Thanks to Abby Finereader for providing their optical character recognition at a greatly discounted price. This research was funded by NSF Research Infrastructure and associated Research Experiences for Undergraduates grant "Spatial Intelligence", number EIA-0080124.

References

ActivMedia ARIA robot control language, http://robots.activmedia.com/aria/

Brown, C.B. (ed.) (1995). Tutorial on Filtering, Restoration, and State Estimation. Technical Report 534, University of Rochester, Department of Computer Science.

CMU Sphinx, http://fife.speech.cs.cmu.edu/sphinx/

Frank Dellaert, Dieter Fox, Wolfram Burgard, and Sebastian Thrun, "Monte Carlo Localization for Mobile Robots," *IEEE International Conference on Robotics and Automation*, 1999.

Matthias Denecke. An Information-based Approach for Guiding Multi-Modal Human-Computer Interaction. Proceedings of IJCAI-97, 1997.

Dieter Fox, Wolfram Burgard, Frank Dellaert, Sebastian Thrun, "Monte Carlo Localization: Efficient Position Estimation for Mobile Robots, *AAAI*, 1999.

Dieter Fox, Wolfram Burgard, Sebastian Thrun, "Markov Localization for Mobile Robots in Dynamic Environments," *Journal of Artificial Intelligence Research*, Vol. 11, pp. 391-427, 1999.

IBM ViaVoice SDK for Windows, http://www-3.ibm.com/software/speech/dev/

Intel Open Source Computer Vision Library, http://www.intel.com/research/mrl/research/opencv/

Patric Jensfelt, Steen Kristensen, "Active Global Localization for a Mobile Robot Using Multiple Hypothesis Tracking," *IEEE Transactions on Robotics and Automation*, Vol. 17, No. 5, 2001.

Kortenkamp, D., Bonasso, R., and Murphy, R. eds. 1998. *AI and Mobile Robots*. Menlo Park, Calif.: AAAI press/MIT press.

Maxwell, B., Meeden, L., Addo, N., Brown, L., Dickson, P., Ng, J., Olshfski, S., Silk, E., and Wales, J. 1999. Alfred: The Robot Waiter Who Remembers You. Proceedings AAAI Workshop on Robotics. Menlo Park, Calif.: AAAI Press.

Michael Montemerlo, Sebastian Thrun, Daphne Koller, Ben Wegbreit, "FastSLAM: A Factored Solution to the Simultaneous Localization and Mapping Problem," *AAAI 2002 Conference Proceedings*, pp. 593-598, 2002.

Stuart Russel and Peter Norvig, Artificial Intelligence: A Modern Approach, Prentice Hall, 1995.

Schmid, J., Kollar, T., Meisner, E., Sweetser, V., Feil-Seifer, D., Brown, C., Atwood, B., Turner, J., Calrese, D., Cragg, S., Chaudhary, H., and Isman, M. 2002. Mabel: Building a Robot Designed for Human Interaction. Proceedings of the AAAI Workshop on Robotics: AAAI Technical Report WS-02-18. p. 24-31. AAAI Press.

Simple Wrapper and Interface Generator, http://www.swig.org

Sebastian Thrun, "AAAI Tutorial on Probabilistic Robotics," *Web: www-2.cs.cmu.edu/~thurn/tutorial/index.htm*, 2002.

Sebastian Thrun, Dieter Fox, Wolfram Burgard, Frank Dellaert, “Robust Monte Carlo localization for mobile robots,” *Artificial Intelligence,* Vol. 128, pp. 99-141, 2001.

Ryuichi UEDA, Takeshi FUKASE, Yuichi KOBAYASHI, Tamio ARAI, Hideo YUASA, Jun OTA, “Uniform Monte Carlo Localization – Fast and Robust Self-localization Method for Mobile Robots,” *Proceedings of the 2002 IEEE International Conference on Robotics and Automation*, pp. 1353-1358, 2002.

Viola, P. and Jones, M. Robust real-time object detection. Technical Report 2001/01, Compaq CRL, February 2001.