

Mabel

Extending Human Interaction and Robot Rescue Designs

Thomas Kollar, Jonathan Schmid, Eric Meisner, Micha Elsner, Diana Calarese, Chikita Purav, Jenine Turner, Dasun Peramunage, Gautam Altekar, and Victoria Sweetser

Advised by Chris Brown, Ph.D.

Department of Computer Science



Artificial Intelligence (AI) has at its core the creation of intelligent systems. The holy grail of this entire subject is to create a general form of intelligence that can reason, learn and interact intelligently in its environment much as a human does. However, the solution to this problem has turned out to be more difficult than anyone could have imagined. In this project, therefore, we have tried to move toward a better understanding of what it takes to create a truly intelligent system.

Mabel (the Mobile Table) is a robotic system developed primarily by undergraduates at the University of Rochester that can perform waypoint navigation, speech generation, speech recognition, natural language understanding, face finding, face following, nametag reading, and robot localization. Mabel uses these components of intelligence to interact with its environment in two distinct ways. It can act as a robot host at a conference by giving people information about the conference schedule. Mabel can also act as a search and rescue robot by entering a mock disaster scene, locating mock human victims, mapping the victims' location, and returning safely out of the scene. Mabel was the winner of the robot host event and tied for third place in the robot search and rescue event at the 2003 International Joint Conference on Artificial Intelligence (IJCAI) in Acapulco, Mexico.

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 the 2002 American Association for Artificial Intelligence (AAAI) conference and the methods that we used to integrate the hardware and software into a usable system.¹

Robot Host

A general overview of the architecture for the information-serving robot can be seen in Figure 1. One can see that there are multiple layers in this system: the sensor level, the interpretation level, and the decision level.

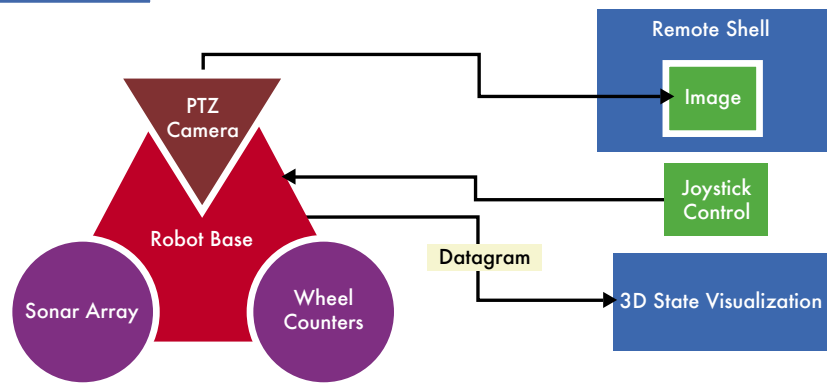
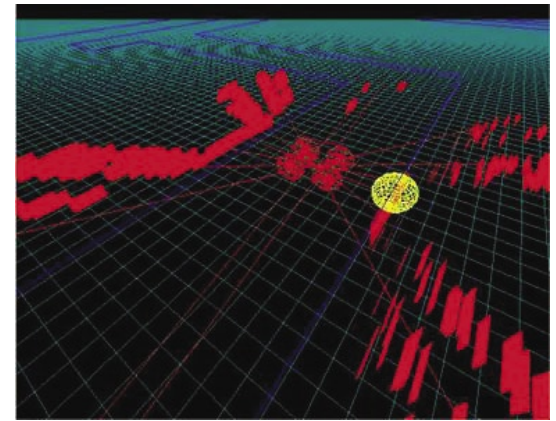
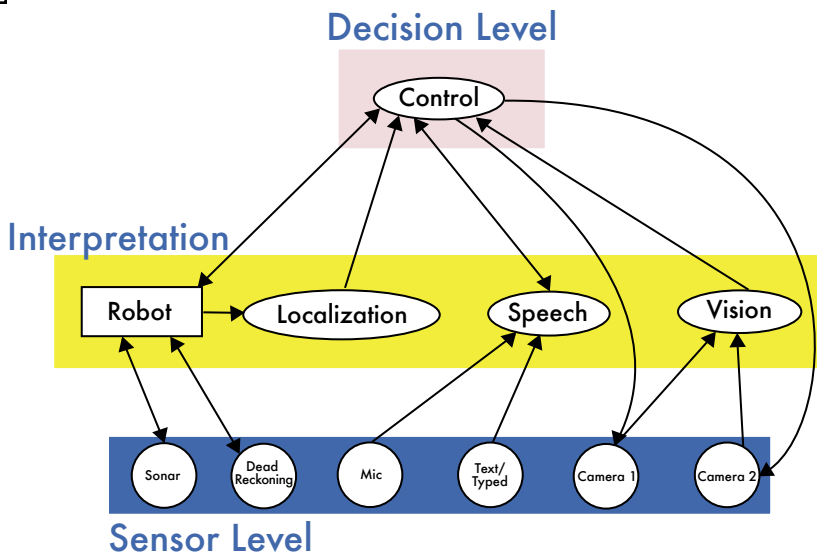
The sensors include sonar, cameras, keyboard, the dead reckoning position and a microphone. The drivers for using these sensors, as well as a library for writing behaviors, were provided with the ActivMedia robot.²

We produced a map of the competition areas in advance 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 interpret the sensors to provide high-level assessments about the environment and intentions of the person with whom the robot is interacting. For example, if the vision component detects a face in the scene, it reports the person's location and name, based on his nametag. The intentions of the patron can also be determined from dialog. For example, the robot can terminate a conversation if a patron says "goodbye."

At the top level is the control aspect (which is basically a finite state machine). If the robot detects someone in front of it then it starts dialog with him. Our system can detect a person's presence if he speaks, types, or if his face is being tracked by the camera. If a person is not detected by the robot then it will wander around to various points on our map of the competition area searching for people, all the while the robot is keeping track of its location (using Monte Carlo Localization, which is discussed later). Should it find a person, then it will continue driving toward him until it successfully reaches him, or, if unsuccessful, it will go back to wandering to various points on the map.

When a person interacts with the robot, many things can happen. Immediately, the robot starts trying to find the person's nametag. Once it has attained that information it uses the information when speaking or displaying other information on the screen. At the same time, the robot continues to track the patron's face with the higher camera. Anytime the patron speaks to it, Mabel parses the words using Sphinx speech recognition then uses a Bayes net to decide what the person's intentions are when talking to it.



Above, Figure 1: The Mabel system structure consists of three layers: sensory input and output, interpretation of sensor data, and decision-making about what to do given the analysis of input data. **Above Right, Figure 2:** 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. **Right, Figure 3:** Overview of the robot search and rescue system.

Given this information, the robot can perform a query on a database of events and produce a response about an event, speaker, time that an event occurred, time of day, or date. 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.

Our system consists of three core components: the control, mapping, and joystick applications. An overview of this system can be seen in Figure

3. The control application, which interfaces directly to the robot's 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 runs completely autonomously on the robot and there is no need for the intervention of the teleoperator.

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 $\langle x, y, \theta \rangle$ to a 3D world. One can see an example of the mapping interface, as rendered by OpenGL, in Figure 2. The teleoperator is also able to load maps of the environment or save maps for printing.

Monte Carlo Localization

Localization is the problem of determining the spatial position of a robot from its sensor readings (vision, sonar, and dead reckoning). It is implemented here using a technique

called Monte Carlo Localization (MCL).³ As in other works, we are only concerned with the readings from the robot's wheel encoders and range devices.

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. There are two problems usually solved by the global techniques: the wake-up robot problem and the kidnapped robot problem.^{3,4} 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 occurrence 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, since without it a robot will have an inaccurate internal representation of its current position.^{3,4} 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.

MCL is a relatively new method that provides a solution to the global localization problem. We implemented the MCL algorithm with distance filtering as per other studies.³ The major benefits of using MCL over the other approaches are that it can globally localize the robot, it greatly reduces the memory requirements in comparison to Markov Localization, it is more accurate than Markov Localization, and it is implemented easily.⁴

For MCL, we divide the new sensor data into two groups: a new odometry reading and a new range sensor reading. $S = \{s_i\}$ such that $i = 1, \dots, N$ 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: $\langle x, y, \theta \rangle, p$. Moreover, we assume $\sum p_i = 1$.

Each time the robot obtains 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, so that "each sample is generated by randomly drawing a sample from the previously computed sample set $[S]$, with likelihood determined by their p -values [probability]."^{3,4} 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.

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 $\langle l, p \rangle$ be a sample and we recalculate p such that $p \leftarrow \alpha P(s|l)$. This can be done in $O(N)$ time.⁴

The probability $P(s|l)$ is 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.⁴ This causes the robot to lose its location, never to recover. There are some techniques to prevent this from happening, but many of these methods are not necessarily mathematically sound.³ The solutions usually involve introducing artificially high amounts of noise into the sensor measurements or by generating samples near 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 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, "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 sub-sample the sensor readings and use a reduced set for localization."³ Thus, we sub-sampled the sonar readings using a distance filter to get the readings that correspond to the world and not the dynamic obstacles in the world.

Thereby the sensor readings that come from unmodeled obstacles were ruled out.⁴

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 allows 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 did not 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.

Vision

Mabel's vision system is focused on finding people in the environment, and reading their nametags. Both techniques make use of a general filter cascade architecture that searched for 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.

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.⁵ We apply a cascade of these filter-classifiers to each sub-window, testing for the presence of several simple rectangular features. While most filter cascade architectures to date have utilized only features within grayscale images, we applied color models to create additional

feature channels as shown in the two central images of Figure 4.

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 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 4. 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 4) 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 4).

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 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 rectangle covering 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. 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 180 x 120 to a full 760 x 480. Ten frames of the nametag are stored at high resolution and read by the Abbyy Fine Reader engine. The most frequent occurrence of the first two string tokens is assumed to be the person's first and last names.

The nametag filter cascade consists of four levels, 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-windows in the image, thereby narrowing down our options significantly.

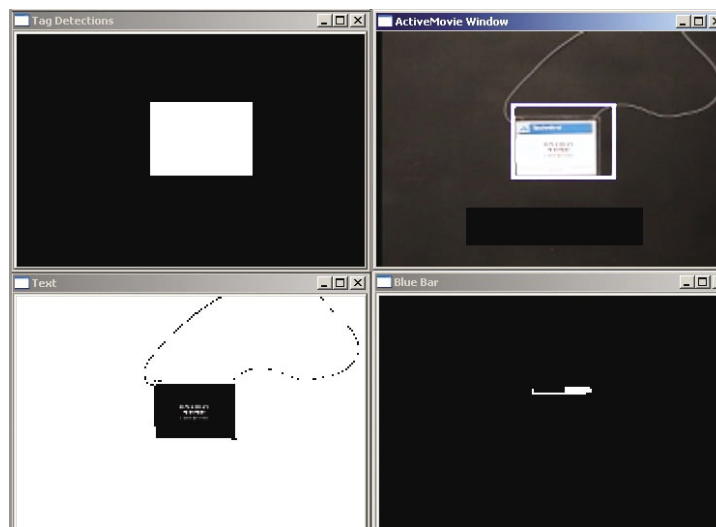
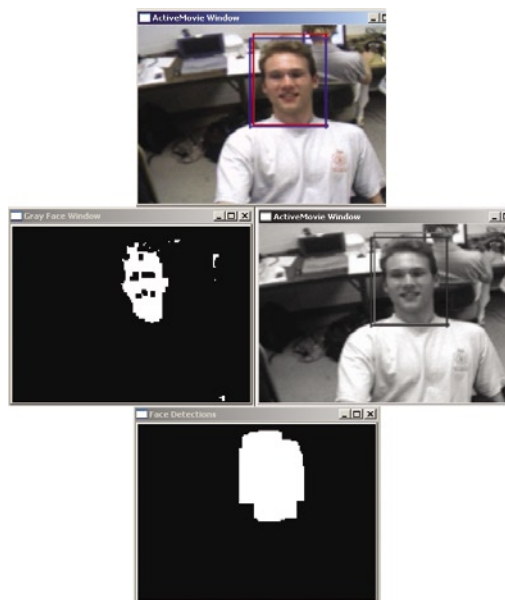
The second level checks for the presence of a colored bar at the top of the nametag (see the bottom right of Figure 5). We distinguish a bar from other objects through the use of a hue/saturation color model for the bar, which was created by sampling several images containing the bars. Once the second level is complete, the majority of the sub-windows are centered vertically on the tag, but remain uncentered horizontally.

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.

Speech

Interaction is initiated whenever the robot recognizes that someone is in front of it, when someone speaks to the robot loudly enough to trigger speech



Left, Figure 4: The process of finding a face. **Above, Figure 5:** The process of finding a nametag.

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 is terminated 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.¹ 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.⁶

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, which was hand-constructed from the conference 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 using 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.¹

Conclusion

This is the second of the Undergraduate Robot Research Team's papers, which now have appeared two years in a row in the AAAI robotics workshop proceedings. Throughout the years the personnel has been diverse, having an unusually high proportion of women for computer science fields. Another unique aspect of this project has been to have a student-run class created with the help of Professor Chris Brown's continuous and overwhelming support. The result has been an incredible experience for undergraduates: an opportunity to work in a team environment, to work on unique research, interact

with professors, and to gain personal recognition while an undergraduate.

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.

During the fall of 2002 and the spring and summer of 2003, there were 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.