Gesture Recognition on a Mobile Robot

Diploma Thesis
by
Stefan Waldherr
Pittsburgh/Kiel, September 1998

Carnegie Mellon

Carnegie Mellon University
School of Computer Science

Dr. Sebastian Thrun

Christian-Albrechts-University of Kiel
Institute of Computer Science and Applied Mathematics
Cognitive Systems Group
Prof. Dr. Gerald Sommer, Dr. Jörg Bruske

Abstract

For mobile robots to assist people in everyday life, they must be easy to instruct. This thesis describes a gesture-based interface for human-robot interaction, which enables people to instruct robots through easy-to-perform arm gestures. Such gestures might be static *pose gestures*, which involves only a specific configuration of the person's arm, or they might be dynamic *motion gestures*, that is, they involve motion (such as waving).

A fast, color-based tracking algorithm enables the robot to track and follow a person reliably through office environments with changing lighting conditions. Gestures are recognized in real-time at approximate frame rate, using a temporal template matcher. Results are reported in the context of an interactive clean-up task, where a person guides the robot to specific locations that need to be cleaned and instructs the robot to pick up trash.

Table of Contents

1. Introduction

The field of robotics is currently undergoing a change. While in the past, robots were predominately used in factories for purposes such as manufacturing and transportation, a new generation of "service robots" has recently begun to emerge. Service robots cooperate with people, and assist them in their everyday tasks. A landmark in service robots is Helpmate Robotics' Helpmate robot, which has already been deployed at numerous hospitals worldwide (see [King and Weiman, 1990]). Helpmate, however, does not interact with people other than by avoiding them. In the near future, similar robots are expected to appear in various branches of entertainment, recreation, health-care, nursing, etc., and it is expected that they interact closely with people.

This upcoming generation of service robots opens up new research opportunities. While the issue of *robot navigation* has been researched quite extensively ([Cox and Wilfong, 1990], [Kortenkamp et al., 1998], and [Borenstein et al., 1996] to name a few), considerably little attention has been paid to issues of *human-robot interaction*. However, many service robots will be operated by non-expert users, who might not even be capable of operating a computer keyboard. It is therefore essential that these robots are equipped with "natural" interfaces that facilitate the interaction between robots and people.

Experience with the deployment of a mobile tour-guide robot in a museum, where it gave interactive tours to visitors (see [Burgard et al., 1997b]), showed that most of the visitors were not previously exposed to robotics and, moreover, were unable to spend sufficient time to deepen their understanding as to how the machine worked. Consequently, they attempted to communicate with the robot by "natural means," such as speech and gestures. The tour-guide robot suffered from the lack of an appropriate interface for human robot interaction. This application, which is just one example of many anticipated service robot applications, motivates the need for research on innovative human-robot interfaces.

The need for more effective human-robot interfaces has been recognized by the research community. For example, Torrance developed in his M.S. thesis a natural language interface for teaching mobile robots names of places in an indoor environment (see [Torrance, 1994]). Due to the lack of a speech recognition system, his interface still required the user to operate a keyboard; however, the natural language component made instructing the robot significantly easier. More recently Asoh and colleagues developed an interface that integrates a speech recognition system into a phrase-based natural language interface (cf. [Asoh et al., 1997]). The authors successfully instructed their "office-conversant" robot to navigate to office doors and other significant places in their environment, using verbal commands.

Communication often involves more than simply spoken language. For example, it is far easier to point to an object than to verbally describe its exact location. Gestures are an easy way to give geometrical information to the robot. Hence, other researchers have proposed vision-based interfaces that allow people to instruct mobile robots via arm gestures (see Chapter 7). Most of these approaches, however, recognize only static pose gestures. They cannot recognize gestures that are defined through specific temporal patterns of arm movements, such as waving. Motion gestures, which are commonly used for communication among people, provide additional freedom in the design of gestures. In addition, they reduce the chances of accidentally classifying arm poses as gestures that were not intended as such. Thus, they appear better suited for human robot interaction than static pose gestures.

Mobile robot applications of gesture recognition impose several requirements on the system. First of all, the gesture recognition system needs to be small enough to fit on the robot — processing power is generally limited. Since both the human and the robot may be moving at all time, the system must not assume a static background or a fixed location of the human. Finally, the system must work at an acceptable speed for the human. Naturally, one would want that a 'Stop' gesture would immediately halt the robot. These requirements that apply to the domain of gesture recognition on mobile robots must be taken into consideration when designing the system.

This thesis presents a vision-based interface that has been designed to instruct a mobile robot through both pose and motion gestures (see [Waldherr et al., 1998b]). An adaptive dual-color tracking algorithm enables the robot to track and, if required, follow a person around at speeds of up to 30 cm per second while avoiding collisions with obstacles (see Chapter 3). This tracking algorithm quickly adapts to different lighting conditions.

Gestures are recognized by a real-time template-matching algorithm, which is described in Chapter 4. This algorithm works in two phases: one that recognizes static arm poses, and one that recognizes gestures (pose and motion). In the first phase, the algorithm computes the correlation of the image with a pre-recorded set of example poses or *pose templates*. In the second phase, the results of the first phase are temporally matched to previously recorded examples of gestures, called *gesture templates*, using a modified Viterbi algorithm (see Section 4.3). The gesture template matcher can recognize both pose and motion gestures. The result is a stream of probability distribution over the set of all gestures, which is then thresholded and passed on to the robot's high-level controller.

This approach has been integrated into our existing robot navigation and control software (see Chapter 2). Chapter 5 describes an interesting application, where a human instructed the robot to perform a "clean-up" task. Experimental results of the gesture recognition system are described in Chapter 6. The thesis concludes with a summary and an evaluation of the gesture recognition system in Chapter 8.

2. Background Information

This chapter presents a description of the robot and its controlling software BeeSoft that is used throughout this thesis project. Although this chapter is not essential for understanding the gesture recognition sytem described in Chapters 3 and 4, it will give the reader an idea of hard- and software issues, particularly with regard to the real-world integration as described in Chapter 5.

2.1. The Robot

Amelia, the robot used in our research, is manufactured by Real World Interface. The robot, which is roughly 106 cm in height and 53 cm in diameter, consists of two main parts: the base and the enclosure. The base uses a synchronous 4-wheel drive coupled with integral dead reckoning for highly accurate motion and positioning control. It allows the robot a top speed of 90 cm/s. The base also holds four, hot-swappable batteries as well as 32 tactile and 32 infrared sensors. Infrared sensors estimate the approximate proximity to an obstacle by emitting light and measuring the intensity of the reflection bounced back off the obstacle's surface. However, infrared sensors have a fairly short range, typically less than 50 cm, and the reflectance value is heavily dependent on the color of the object detected. Readings are sampled at about 10 Hz.

Between the base and the enclosure (see Figure 2.2) is a 180° SICK laser range finder, which uses light transit times to measure distances. It can detect obstacles in front of the robot in a distance of up to 30 m with an angular resolution of 0.5 degrees. However, the laser range finder has problems detecting either transparent or black obstacles. In the first case, the laser beam will not be reflected by the object, in the latter case the laser beam will be absorbed. Hence it is impossible to measure a light transit time.

Located above the base is the enclosure, which contains the two on-board computers that control the robot's devices. Both are 200 MHz Intel Pentium processors with 64 Mb of memory running RedHat Linux. They are connected on the robot through a regular Ethernet and to external workstations through a radio Ethernet link. One machine holds two Matrox Meteor frame grabbers that sample the camera images. The frame grabbers, which use the RGB color model, are capable of sampling camera images up to a rate of 30 frames per second. Their maximum resolution is 768×480 pixels.

The enclosure also holds a ring of 24 sonar proximity sensors that measure approximate echo distances with an effective range of 30 cm to 8 m. Each sonar detects obstacles in a cone-shaped range with an angle of about 15 degrees. The sonar software samples with a frequency of roughly 2 Hz per sensor. However, the object's surface characteristics and the impact angle of the sonar wavefront greatly affect the way the object is detected. Thus, the sonar sensors are not as precise as the laser range finder. They also fail to detect obstacles of low height.

Mounted within the enclosure is a 3-degree-of-freedom arm, whose joints are the mast, wrist and the gripper (see Figure 2.1). While the robot moves around the arm is generally stowed within the enclosure. In addition, the enclosure has 24 tactile and 24 infrared sensors.

Located on top of the enclosure is the camera head (a Direct Perception PTU-46-17.5 pan-tilt unit), which can pan approximately ± 160 degrees and tilt ± 40 degrees. Its maximum speed is 300°/s. The pan-tilt unit carries a Sony XC-999 color CCD camera. The CCD chip used in the camera has 752×582 picture elements. According to the amount of incident light, the camera automatically adjusts the shutter speed. In our research we used a wide-angle lens, which had a 125 degree field of view.

The devices mentioned above are controlled by a number of different programs, which will be described in the next section.

2.2. Integrated Robot Software BeeSoft

Real World Interface, Inc.'s BeeSoft package, which was developed in conjunction with Carnegie Mellon University and the University of Bonn, is the software that controls the robot (see [Thrun et al., 1997]). The software provides commands to interpret sensor data, move the robot around or control any of the on-board devices, such as the camera or the arm. The software's client-server architecture allows the user to easily write own programs that control any of the servers, which in turn control the robot's devices. Some servers must run on a specific machine on the robot, because they control a device directly attached to that machine – other servers, or *modules*, run on remote machines.

The key principles for the design of BeeSoft are as follows: Firstly, there is no single control unit: communication among modules, which is realized by TCX (as described in Section 2.3), is not centralized. Secondly, communication is asynchronous. Each module

Figure 2.1.: Amelia is a RWI B21 robot equipped with, among other devices, a color camera mounted on a pan-tilt unit, 24 sonar sensors, 56 infrared sensors and a 180° SICK laser range finder.

runs independently and hence does not require the presence of any other module. To resolve conflicts, certain modules (such as the on-board obstacle avoidance) can take priority over other modules in determining the robot's motion direction. Finally, BeeSoft was designed to accommodate sudden failures of most of its software components, i.e., it is fault tolerant. Almost all modules can be stopped and restarted at any time. Effective mechanisms ensure that restarted modules will immediately obtain the currently available global information.

Each module has a specific function, which we briefly explain at this point:

armServer The arm server controls the arm, which allows us to include manipulation tasks in our research. While moving around, the robot's arm is typically stowed within the enclosure. Upon deploying the arm, the user can control the 3-degree-of-freedom arm, e.g., to pick up a soda can (see Chapter 5).

baseServer The base server issues commands to the robot's wheeled base. Using the baseServer program, one can move the robot without using any obstacle avoidance routines. In addition, the base server provides control of and interface to the robot's sonar, tactile and infrared sensors.

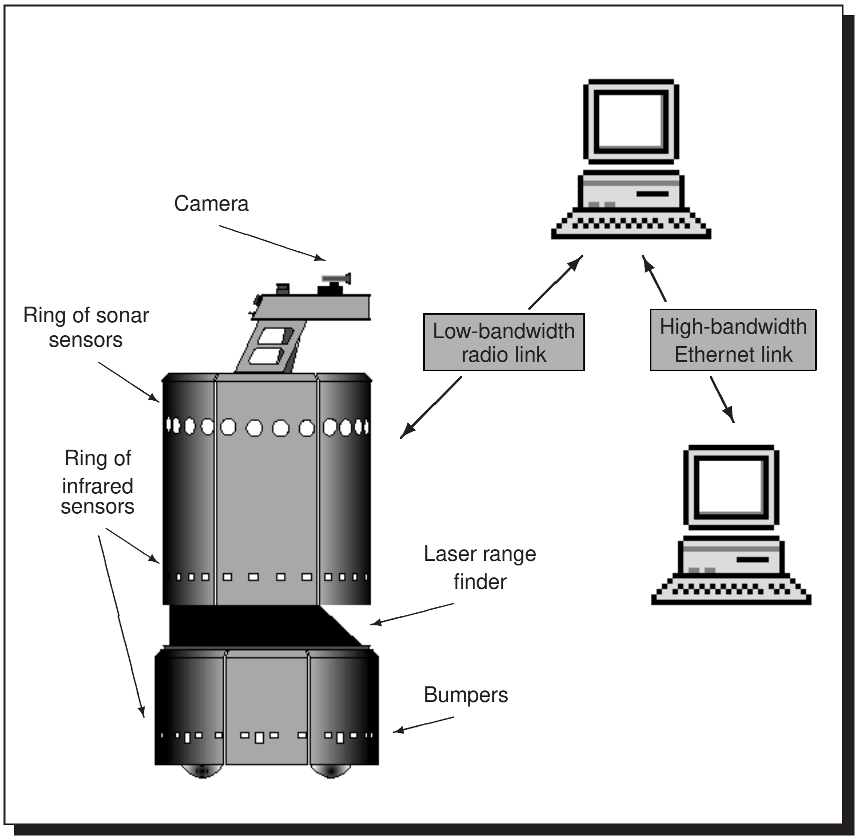


Figure 2.2.: The robot's sensors and it's connection to external workstations.

buttonServer The button server watches the states of the six buttons that can be accessed on top of the robot. The buttons can be used as an input device, e.g., to trigger a specific action whenever the user presses one of them. Through lighting they also provide a simple means to indicate, e.g. that a gesture was recognized.

cameraServer The camera server controls both Matrox Meteor frame grabbers on-board the robot. Other processes access the grabbed image through a shared memory segment. It is important to add that this access is asynchronous, i.e., other processes do not know, if or when the frame grabber received a new image. Yet, this access method has the advantage that it is both fast and transparent.

colliServer The colliServer program issues commands to the robot's wheeled base in *guarded motion* mode, that is, incorporating built-in collision avoidance. The user can set target points at which to aim, either a relative target point or an absolute target point. While simultaneously trying to avoid any obstacles it perceives, the robot then moves to these target points. colliServer's main source of information about its environment are the 24 sonar sensors, which can perceive obstacles at approximately table height. Additionally, colliServer uses the laser range finder. The colliServer program uses the baseServer to perform the actual motion commands. For further information on the algorithms used in the collision avoidance, see [Fox et al., 1997] and [Fox et al., 1998].

laserServer The laser server controls the SICK laser range finder. The laser range finder, in contrast to the sonar sensors, is a very precise instrument to detect obstacles. A client program typically subscribes to the laser readings, which the server broadcasts four times per second.

localizeServer The localizeServer program uses the map server and the current sensor readings, provided by colliServer, to determine the exact position of the robot. The algorithms to *actively* localize the robot, i.e., using the robot's effectors, are described in [Burgard et al., 1996] and [Burgard et al., 1997a].

mapServer With the map server the robot is able to learn and maintain a model of its environment. The program builds or extends existing maps, using sensor information provided by the colliServer program.

panTiltServer The panTiltServer program sends commands to the robot's pan-tilt unit. The user can control the speed, acceleration and power of the move, or send the head to any fixed location.

planServer Using the map that the map server provides, the user can set arbitrary target points within the world. The goal of the planServer program is to find a path that leads to this target point.

tcxServer The TCX server (cf. Section 2.3) is the core of the communications management software. It runs on one computer, but coordinates communication among all other modules on all, even remote machines.

It is crucial that a reliable communication among the above programs exists at all time. We will examine some of the problems and implementation issues in the next section.

2.3. Interprocess Communication – TCX

To cope with the computational demand of gesture recognition, and for that matter robot applications in general, it is necessary that we use not only the CPU power on-board the robot but also the processing power of external workstations. Multiple processes on multiple machines, connected through low and high bandwidth networks, must be able to communicate among each other. Simple interprocess communication routines for building robotic architectures are available through TCX.[1]

TCX is an interprocess communication package designed to support mobile robotic applications across multiple hardware platforms, operating systems and programming languages. The goal of TCX is to provide routines for controlling the sending and receiving of information and maintaining channels of communication between cooperative processes. More specifically TCX provides the communication building blocks to coordinate flow of control and flow of data between planning, perception and real time control in mobile robots.

TCX is particular useful when it comes to distributing computational burden among different machines. We cannot run the entire gesture recognition system on-board the robot, since the machines are not powerful enough. We also cannot run the entire system on external machines, since this would mean that we have to transmit the camera images over the radio link – a communication channel, which clearly cannot handle that amount of data. Hence, we split the gesture recognition system into two parts. While vision processing is done on-board the robot, only feature values are transmitted over the radio link to external workstations.[2] Upon analyzing these feature values, a recognizeServer that runs on an external workstation (see Section 5.2) decides if and when a gesture was observed.

The program tcxServer is the core process which coordinates interprocess communications. Hence, the server knows which modules are connected at all time. The server attempts to maintain the network of communicating modules despite new modules entering and other modules leaving (including leaving due to failure or crash). However, the tcxServer only establishes connection between modules. Once the connection is established, modules will communicate with each other by themselves, bypassing the tcxServer.

Communication is provided by messages. In general, a message based communication system has facilities to establish communications between processes without access to shared variables. Hence, it enables interprocess communication even among processes running on different machines.

[1] See [Fedor, 1994].
[2] Features used for gesture recognition will be described in Chapter 4.

TCX makes use of message queues to provide some mixture of both direct and indirect message base communication. With direct communication, the sender must explicitly name the process which should receive the message. With indirect communication, messages are sent to and read from mailboxes. Mailboxes are abstract objects where messages can be placed and read by processes. This is especially convenient when it comes to status reports. For example, the laserServer stores its current readings into a “laser readings mailbox” to which other processes have immediate access.

Additionally, TCX provides an easy way for registering *call-back* functions. Call-back functions are executed upon receiving a particular message. Referring to the above example, a module could register a call-back function so that it is notified of any new laser reading. Thus, the module does not have to query the mailbox on its own.

Registering functions for all messages may make it a little awkward to wait for an incoming message. TCX also allows to receive messages in response to a read call, e.g., acknowledgement messages. This follows a simple query-reply scheme.

Finally, TCX provides limited ability to send high priority information by allowing a message to be inserted at the top of a particular message queue. In this manner the message will arrive via the same communication link or implementation as all other messages but will be processed ahead of those messages currently in the message queue. This is particularly useful for important messages such as “stop robot”.

3. Finding and Tracking People

3.1. Motivation

Finding and tracking people is the core of any vision-based gesture recognition system. After all, the robot must know with whom to communicate. Visual tracking of people has been studied extensively over the past few years (see [Darrel et al., 1996], [Crowley, 1997], and [Wren et al., 1997]). However, the vast majority of existing approaches assumes that the camera is mounted at a fixed location. Such approaches typically rely on a static background, so that human motion can be detected, e.g., through image differencing. In the case of robot-based gesture recognition, one cannot assume that the background is static. While the robot tracks and follows people, not only background but also lighting conditions can change considerably. In addition, processing power on a robot is usually limited, which imposes an additional burden.

We chose color as a feature to find or track people and extended an existing system (cf. [Yang and Waibel, 1995]), which was reported to track people extremely reliable with a camera mounted on a pan-tilt unit. The approach uses the person's face color as its main feature. Our approach tracks the person based on a combination of two colors, namely face color and shirt color. At a later stage of our algorithm, we use the person's shirt color to detect gestures.

3.2. Basics

Face color as a feature for tracking people has been used before and has several advantages. Processing color is faster than processing other, more complex features. Color is also orientation invariant under certain lighting conditions. However, tracking people using face color has some problems. The color of a human faces is influenced by many factors, such as the camera itself, clothes the person wears or different lighting conditions. It is therefore essential for successful tracking to cope with the changes in lighting conditions. We will see in Section 3.5 how the system adapts to different lighting conditions. Furthermore, human face color differs not only under changing lighting conditions but also from person to person. Although it seems that face colors of different people vary over a wide range, they differ mainly in brightness and not in color. Consequently, we use a color model that preserves an accurate but low dimensional color representation while removing brightness from the image.

Cameras typically use the RGB model, i.e., each pixel on the camera's retina is a triple $P = (R, G, B)$. In the RGB color space a pixel represents not only color but also brightness. If corresponding elements in two pixels P_1 and P_2 are proportional, that is

$$\frac{R_{P_1}}{R_{P_2}} = \frac{G_{P_1}}{G_{P_2}} = \frac{B_{P_1}}{B_{P_2}}, \tag{3.1}$$

the two pixels have the same color, but different brightness. Since brightness is not so important for characterizing face colors, we can represent face color in chromatic color space. Chromatic colors (see [Wyszecki and Styles, 1982]), also known as *pure colors* in the absence of brightness, are defined by a normalization process $\mathbf{R}^3 \mapsto \mathbf{R}^2$, where

$$r = \frac{R}{R+G+B} \tag{3.2}$$

and

$$g = \frac{G}{R+G+B}. \tag{3.3}$$

For N pixels in the image, a face color distribution can be represented by a Gaussian model $M = (\mathbf{\Sigma}, \mu_r, \mu_g)$, which is characterized by the means μ_r and μ_g

$$\mu_r = \frac{1}{N}\sum_{i=0}^{N} r_i, \tag{3.4}$$

$$\mu_g = \frac{1}{N}\sum_{i=0}^{N} g_i \tag{3.5}$$

and the covariance $\mathbf{\Sigma}$

$$\mathbf{\Sigma} = \begin{pmatrix} \sigma_r^2 & \sigma_{rg} \\ \sigma_{gr} & \sigma_g^2 \end{pmatrix}, \tag{3.6}$$

where

$$\sigma_r^2 = \sum_{i=0}^{N} (r_i - \mu_r)^2, \tag{3.7}$$

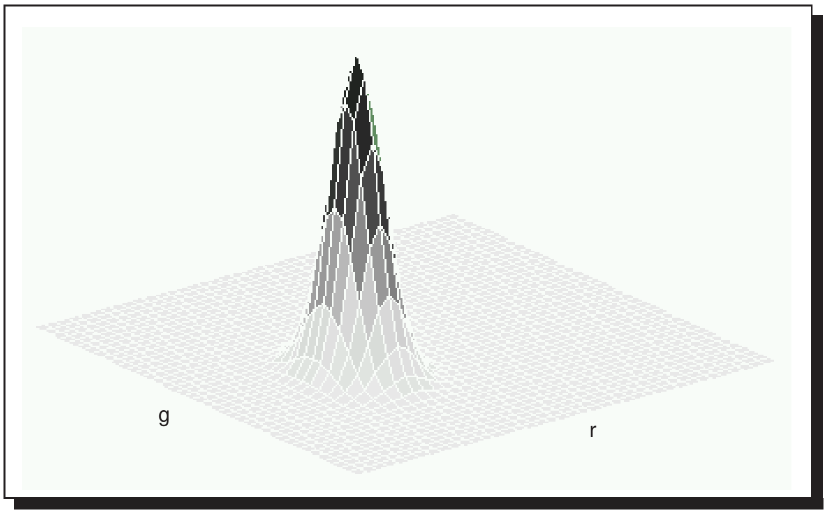


Figure 3.1.: Face-color distribution in chromatic color space. The distribution was obtained from the person's face shown in Figure 3.3(a).

$$\sigma_g^2 = \sum_{i=0}^{N} (g_i - \mu_g)^2 \tag{3.8}$$

and

$$\sigma_{rg} = \sigma_{gr} = \sum_{i=0}^{N} ((r_i - \mu_r)(g_i - \mu_g)). \tag{3.9}$$

A color distribution of human face color is clustered in a small area of the chromatic color space, i.e., only a few of all possible colors occur in a human face. Figure 3.1 shows such a face color distribution for the person in Figure 3.3(a). The x and y axis represents the normalized r and g values, the z axis is the response to a color model. The higher the response, the better did the person's face match the color model.

Representing face color in chromatic color space has the advantage that human face colors of different people have similar Gaussian distributions in chromatic color space. Deviations are smaller than they would have been in RGB color space. Figure 3.2 shows a face color distribution obtained from analyzing the face color of a number of people. The x and y axis represents the normalized r and g values. The darker a pixel the more often we observed this particular color combination.

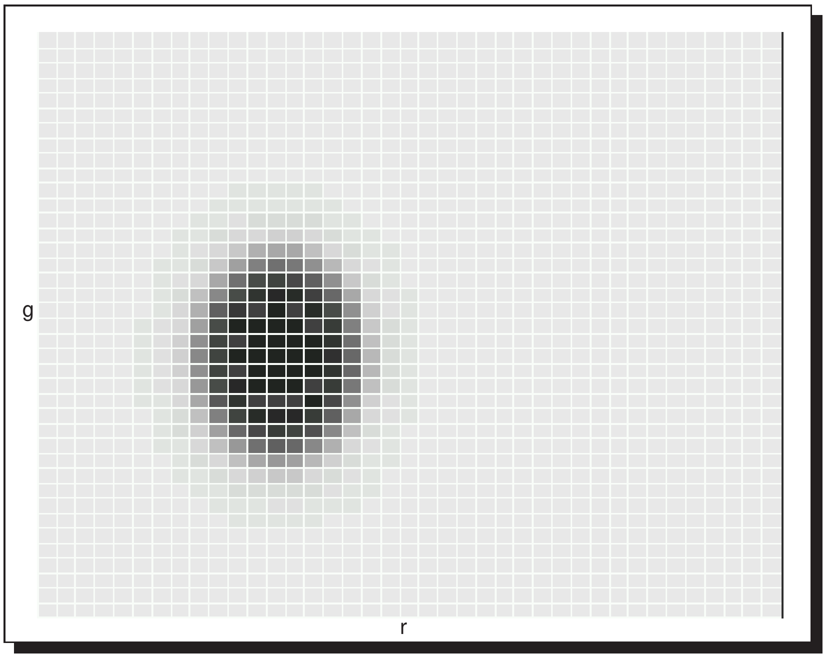


Figure 3.2.: Face-color distribution obtained from a set of people. The darker a pixel, the more often has this particular color combination been observed.

For an arbitrary pixel i, one can rewrite Equation (3.6) to

$$f(r_i, g_i) = \frac{1}{2\pi\sigma_r\sigma_g\sqrt{1-\varrho_{rg}^2}} \exp\left(-\frac{1}{2(1-\varrho_{rg}^2)} \left(\frac{(r_i-\mu_r)^2}{\sigma_r^2} - 2\frac{\varrho_{rg}(r_i-\mu_r)(g_i-\mu_g)}{\sigma_r\sigma_g} + \frac{(g_i-\mu_g)^2}{\sigma_g^2}\right)\right) \tag{3.10}$$

with correlation coefficient ϱ_{rg} as defined by

$$\varrho_{rg} = \frac{\sigma_{rg}}{\sigma_r\sigma_g}. \tag{3.11}$$

We use not only face color to track people, but also the person's shirt color, which we later use for gesture recognition. Hence we have two color models:

$$M_{\text{face}} = (\boldsymbol{\Sigma}_{\text{face}}, \mu_{r_{\text{face}}}, \mu_{g_{\text{face}}}) \tag{3.12}$$

and

$$M_{\text{shirt}} = (\boldsymbol{\Sigma}_{\text{shirt}}, \mu_{r_{\text{shirt}}}, \mu_{g_{\text{shirt}}}) \tag{3.13}$$

Figure 3.3.: Tracking a person: Raw camera image (a), face-color filtered image (b) and shirt-color filtered image (c). Projection of the filtered image onto the horizontal axis (within a search window) (d). Face and shirt center, as used for tracking and adaptation of the filters (e). Search window, in which the person is expected to be found in the next image (f).

For any give camera image, we first map each pixel into chromatic color space and then compute the response to the face and shirt color filter, i.e., $f_{\text{face}}(r_i, g_i)$ and $f_{\text{shirt}}(r_i, g_i)$ (see Equation (3.10)). For the raw camera image in Figure 3.3(a) one can see the response to the face color filter in Figure 3.3(b) and to the shirt color filter in Figure 3.3(c). The better the pixel matches our color model, the higher is the response to the color filter and hence the darker the pixel is in the image.

Yet, one question remains: how do we determine these models? This question is closely related to the task of initially finding the person, which we will examine in the next section.

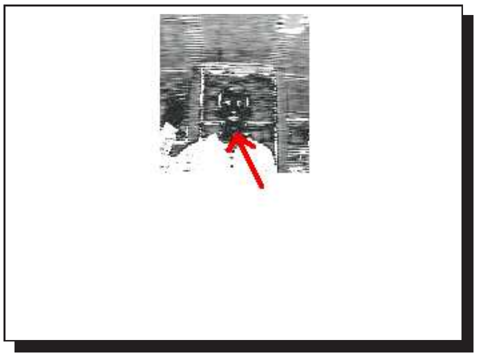


Figure 3.4.: Response to the average face color model used to find the person initially.

3.3. Finding People

To be found by the robot, the person presents himself somewhere in front of the robot. Using a color model M_{face}, which represents an average face color, the system determines the response to this average face color in the upper half of the image's middle third. With a simple algorithm we then determine the center of gravity in this particular area and thus the center of the person's face. Figure 3.4 shows the Gaussian response to the average face color as well as the center of the face.

Now that we have found the position of the person in the image, the next step is to determine initial face and shirt color models. We compute the means μ_r, μ_g and covariance $\mathbf{\Sigma}$ according to Equations (3.4), (3.5), and (3.6) in a window W_{face} of fixed size around the face center. For simplicity reasons, we assume that the person is between one and two meters away from the robot so that we can choose the size of the window to be sufficiently small. If the window is too big, the variance in color will increase and our model would not be as exact.

We use the x coordinate of the face center and subtract n pixels of its y coordinate to determine the center of the person's body.[1] Again, we compute the means and covariance in a fixed window W_{shirt} around the shirt center. Both windows W_{face} and W_{shirt} are depicted in Figure 3.3(e). Note that we didn't need any previous knowledge about the person's shirt color in order to find the person. The system acquires its initial shirt color model based on a region below the face. Thus, it can locate and track people with arbitrary shirt colors, as long as they are sufficiently coherent.

[1]Since we know that the person is roughly between one and two meters away from the robot, we can determine n experimentally.

3.4. Tracking people

In the initial task of finding the person, we first determined the center of the face and then the center of the shirt. However, treating each filter response independently in order to track the person is not as advantageous as combining the responses of both filters. Consequently, we are still able to track the person even if the response of one filter is not as good as expected.

Locating the person in the image consists of two steps: firstly, we determine the position of the person in the image horizontally, i.e., the person's x coordinate in the image. Secondly, we determine the center of the face and shirt vertically along this x coordinate.

To decrease the search space in which we look for the person, we focus our search around the position where the person was previously observed. Hence, we use two search windows which we construct around the person's face and body: S_{face} and S_{shirt}. Both windows are shown in Figure 3.3(f).

1. We compute the response to the face and shirt color model in the respective search windows S_{face} and S_{shirt}. In a second step, our system searches for co-occurrences of vertically aligned face and shirt color. This step rests on the assumption that the face is always located above the body. We then sum the color filter's response horizontally over m neighbors and average over an entire column. Figure 3.3(d) shows the resulting two vectors, which are expanded vertically. The grey-level in the two regions S_{face} and S_{shirt} graphically indicates the horizontal density of face and shirt color, respectively. The darker the pixel, the better the match. Finally, both vectors are multiplied component-wise. To determine the estimated horizontal coordinate of the person, we search for the maximum value of this product and use its index as the person's x coordinate.

2. In order to determine the vertical coordinates of the face and shirt center, we search for the largest response in S_{face} and S_{shirt} along the previously found x coordinate. Though, we first smooth this column to avoid noise, and therefore a strong deviation from the previous position.

Figure 3.5 summarizes the finding and tracking task in pseudo-code. In the next section we will take a closer look at the importance of adapting to changing lighting conditions.

```
personFound = FALSE
for (;;) {
   if (!personFound) {
      compute filter response to average face color model
      determine center of filter response, i.e., face center
      /* determine windows */
      determine W_face
      determine W_shirt below the face center
      determine S_face
      determine S_shirt
      /* acquire initial models */
      compute means and covariance for face color in W_face
      compute means and covariance for shirt color in W_shirt
      /* person found */
      personFound = TRUE
   } else {
      compute filter response to face color model in S_face
      compute filter response to shirt color model in S_shirt
      determine new face and shirt center
      determine W_face
      determine W_shirt
      determine S_face
      determine S_shirt
      adapt face color model in  W_face
      adapt shirt color model in  W_shirt
   }
}
```

Figure 3.5.: The finding and tracking task described in pseudo-code.

3.5. Adaptation

While following a person, the appearance with respect to color can change drastically. Figures 3.6(a) to 3.6(d) show images that were recorded while the robot followed a person through a building. Note the different illumination of the person in the lab and in the hallway. Even under the same lighting conditions, changes in background colors may

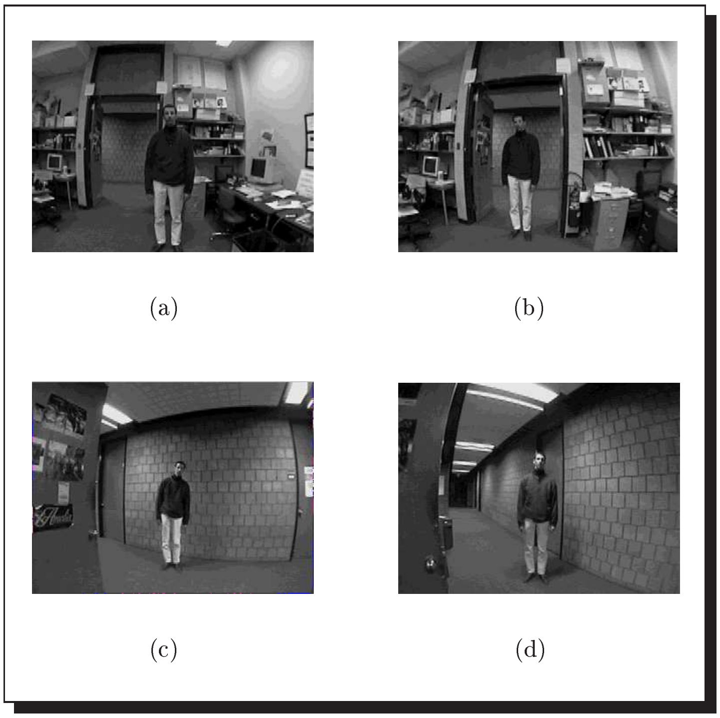


Figure 3.6.: This sequence of camera images, recorded in a single run, illustrates the potential speed and magnitude of changes in lighting conditions, which make gesture recognition on mobile robots difficult.

influence face-color appearance. Similarly, if the robot follows the person, the apparent face color change as the person's position changes relative to camera or lighting. Consequently, it is essential for the tracking system to cope with different lighting conditions. The aim of adaptation is to transform a previously determined color model to the new environment. Since our color model consists of two means and the covariance, we adapt the model as follows:

$$\boldsymbol{\Sigma}^t_{\text{face}} = \alpha\boldsymbol{\Sigma}^*_{\text{face}} + (1-\alpha)\boldsymbol{\Sigma}^{t-1}_{\text{face}} \tag{3.14}$$

$$\mu^t_{r_{\text{face}}} = \alpha\mu^*_{r_{\text{face}}} + (1-\alpha)\mu^{t-1}_{r_{\text{face}}} \tag{3.15}$$

$$\mu^t_{g_{\text{face}}} = \alpha\mu^*_{g_{\text{face}}} + (1-\alpha)\mu^{t-1}_{g_{\text{face}}} \tag{3.16}$$

The shirt color model is adapted in the same fashion:

$$\boldsymbol{\Sigma}^t_{\text{shirt}} = \alpha\boldsymbol{\Sigma}^*_{\text{shirt}} + (1-\alpha)\boldsymbol{\Sigma}^{t-1}_{\text{shirt}} \tag{3.17}$$

$$\mu^t_{r_{\mathrm{shirt}}} = \alpha\mu^*_{r_{\mathrm{shirt}}} + (1-\alpha)\mu^{t-1}_{r_{\mathrm{shirt}}} \tag{3.18}$$
$$\mu^t_{g_{\mathrm{shirt}}} = \alpha\mu^*_{g_{\mathrm{shirt}}} + (1-\alpha)\mu^{t-1}_{g_{\mathrm{shirt}}} \tag{3.19}$$

Here, $\mathbf{\Sigma}^*_{\mathrm{face}}$, $\mu^*_{r_{\mathrm{face}}}$ and $\mu^*_{g_{\mathrm{face}}}$ denote values that are obtained from the most recent image only, whereas $\mathbf{\Sigma}^{t-1}_{\mathrm{face}}$, $\mu^{t-1}_{r_{\mathrm{face}}}$ and $\mu^{t-1}_{g_{\mathrm{face}}}$ are values of the model at a previous time-step. The adaptation rate α, which we set to 0.1 in all our experiments, determines how quickly we want the model to adapt to changes in lighting conditions.

4. Gesture Recognition

4.1. Motivation

The primary goal of this work has been to devise a vision-based interface that is capable of recognizing both pose and motion gestures. Pose gestures involve a static configuration of a person's arm, such as the 'Stop' gesture shown in Figure 4.1(a), whereas motion gestures are defined through specific motion patterns of the arm, such as the 'Follow' gesture shown in Figure 4.1(b).

(a) (b)

Figure 4.1.: Example gestures: 'Stop' gesture (a) and 'Follow' gesture (b). While the 'Stop' gesture is a static gesture, the 'Follow' gesture involves motion, as indicated by the arrows.

Our approach consists of two phases: in the first phase, we resolve the question which arm-position the person has in the image; this requires that the person's arm has the same color as the person's shirt. We refer to this process as *pose analysis*. In the second phase, we find out how the position of the person's arm changes over time, and if the observed pattern is a gesture. We record sequences of arm poses from a stream of images and temporally match this sequence with a gesture prototype, or *gesture template*. Hence, we call the process *temporal template matching*.

4.2. Pose Analysis

The approach employs a mechanism for analyzing arm poses, which compares arm poses to a set of pre-recorded *pose templates*. More specifically, the color-filtered image is correlated with a set of R pre-recorded templates of arm poses. Each pose template $\mathbf{T}$ consists of three regions: an *excitatory* region, which specifies where the arm is to be expected for a specific pose, an *inhibitory* region, where the arm should *not* be for this pose, and a *don't-know* region, which is not considered when computing the correlation.

Figure 4.2 shows examples of pose templates. Here, excitatory regions are shown in black, whereas inhibitory regions are shown in grey. The templates are constructed from labeled examples of human arm poses (one per template), where the excitatory region is extracted from the largest coherent region in the filtered image segment, and a simple geometric routine is employed to determine a nearby inhibitory region. Consequently, a pixel $p \in \mathbf{T}$ can have the following values

$$p = \begin{cases} +1 & \text{Excitatory region,} \\ -1 & \text{Inhibitory region,} \\ 0 & \text{Not considered.} \end{cases} \tag{4.1}$$

Since we expect the response of the shirt color model to be high, i.e., close to one in the region of the shirt, the value of a pixel in the pose template is also close to one. The inhibitory region on the other hand indicates that we do not expect the response of the shirt color model to be high.

The correlation $\varrho_{\mathbf{TI}}$ between a pose template $\mathbf{T}$ and the response $\mathbf{I}$ to shirt color model is defined by the correlation coefficient

$$\varrho_{\mathbf{TI}} = \frac{\sigma_{\mathbf{TI}}}{\sigma_{\mathbf{T}}\sigma_{\mathbf{I}}} \tag{4.2}$$

and is computed as follows:

$$\varrho_{\mathbf{TI}} = \frac{\sum\limits_{p \in \mathbf{T}, p \neq 0} (p - \mu_{\mathbf{T}})(p_{\mathbf{I}} - \mu_{\mathbf{I}})}{\sqrt{\sum\limits_{p \in \mathbf{T}, p \neq 0} (p - \mu_{\mathbf{T}})^2 \sum\limits_{p \in \mathbf{T}, p \neq 0} (p_{\mathbf{I}} - \mu_{\mathbf{I}})^2}} \tag{4.3}$$

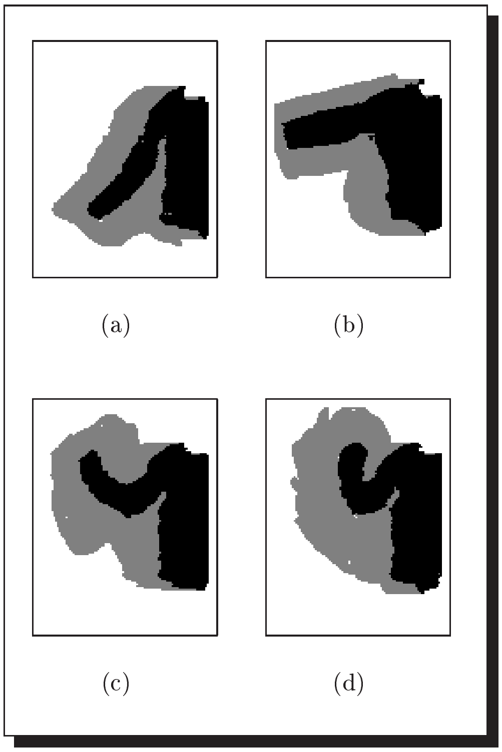


Figure 4.2.: Examples of pose templates. The excitatory region is shown in black and the inhibitory in grey. White regions are not considered in the matching process.

We limit the correlation between the pose template and the filter response to pixels of the pose template that are either $+1$ or -1. Hence the restriction to $p \in \mathbf{T}, p \neq 0$ in the summations. This also holds for the response to the shirt filter — $p_{\mathbf{I}}$ denotes a pixel in the filter response with the same coordinates as the pixel p in the pose template. The means $\mu_{\mathbf{T}}$ and $\mu_{\mathbf{I}}$ are only computed in the area of the pose template.

Note that the terms $(p - \mu_{\mathbf{T}})$ in the numerator and $\sum_{p \in \mathbf{T}, p \neq 0}(p - \mu_{\mathbf{T}})^2$ in the denominator only have to be determined once, whereas all other computations have to be made for each frame.

At this point it is important to note that we can only correlate a pose template with the response to the shirt filter if we have a ledger — a checkpoint that is fixed. Such a point is the person's face center, which we determined earlier (see Section 3.4). Only the knowledge about the person's face center allows us to correlate the response to the shirt filter and a pose template properly.

As mentioned before, the approach correlates the response to the shirt filter to a set of R pre-recorded pose templates. The result is a correlation vector, or *feature* vector, of R components — one for each pose template as depicted in Figure 4.3. The resulting vector is shown as a Hinton diagram. In a Hinton diagram, a vector's component is represented by a square whose size is associated with the magnitude and whose color indicates the sign (black is negative, white is positive). We will use this graphical representation throughout the rest of this thesis.

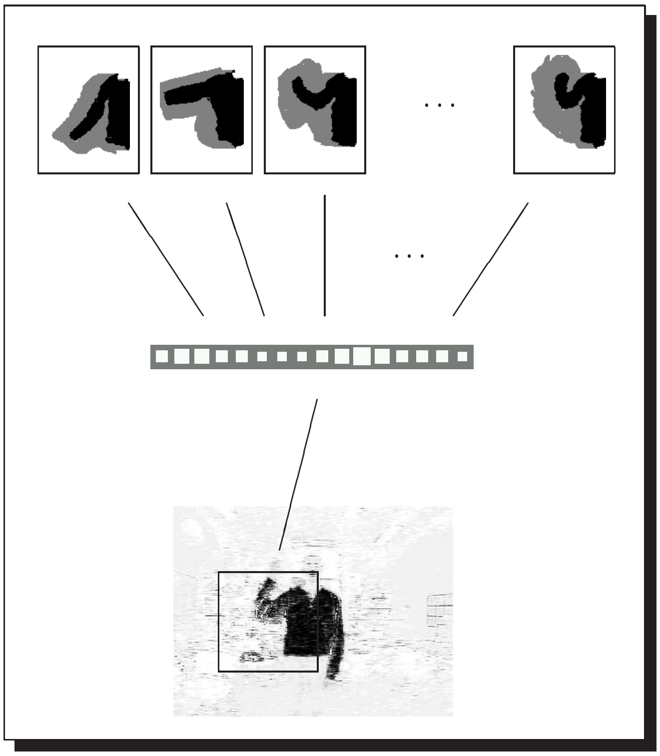
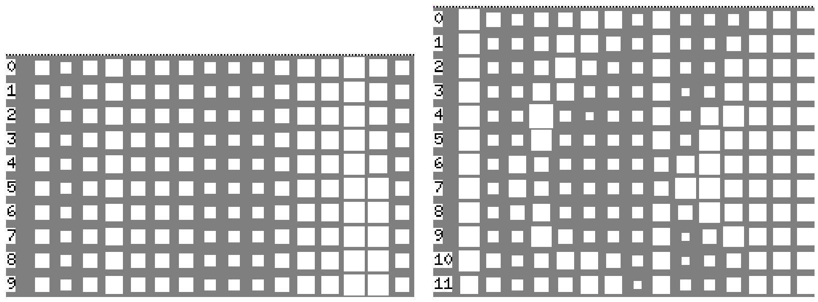


Figure 4.3.: The matching process between the response of the shirt color model and each of the arm-position templates in the database. The result of the matching process, a *feature vector*, is shown in the middle of this figure.

For each new frame observed, we compute the feature vector as described above. These feature vectors form the basis of the temporal template matching. The temporal template matching, in which we analyze the stream of feature vectors, and the problems in coping with gestures of different length will be described in the next section.

4.3. Temporal Template Matching

In the second phase of our approach, a temporal template matcher compares the stream of feature vectors with a set of prototypes, or *gesture templates*, of individual gestures that have been previously recorded. Figure 4.4 shows examples of gesture templates, for the gestures ‘Stop’ (Figure 4.1(a)) and ‘Follow’ (Figure 4.1(b)). Each of these templates is a sequence of prototype feature vectors, where time is arranged vertically. For graphical clarity, only feature vectors with 16 components are shown here. As can be seen in this figure, the ‘Stop’ gesture is a pose gesture (hence all feature vectors look alike), whereas the ‘Follow’ gesture involves motion. Both types of gestures — pose gestures and motion gestures — are encoded through such gesture templates.



(a) Stop (b) Follow

Figure 4.4.: Examples of gesture templates. Gesture templates are sequences of prototype feature vectors. Shown here are gesture templates for (a) a ‘Stop’ gesture (does not involve motion), (b) a ‘Follow’ gesture (involves motion, as indicated by the change over time).

How do we obtain these prototypes or gesture templates? To teach the robot a new gesture, the person presents itself in front of the robot and executes a gesture several times (e.g., five), in pre-specified time intervals. Our approach then segments these examples into pieces of equal length, and uses the average feature vector in each time segment as a prototype segment. Figure 4.5 shows the segmentation process — for simplicity reasons the figure only depicts one training example. Note that this process compresses an observed gesture, typically by the order of four to five: For example, a gesture that was observed over 85 frames (and hence consists of 85 vectors), can be represented with 20 feature vectors.[1]

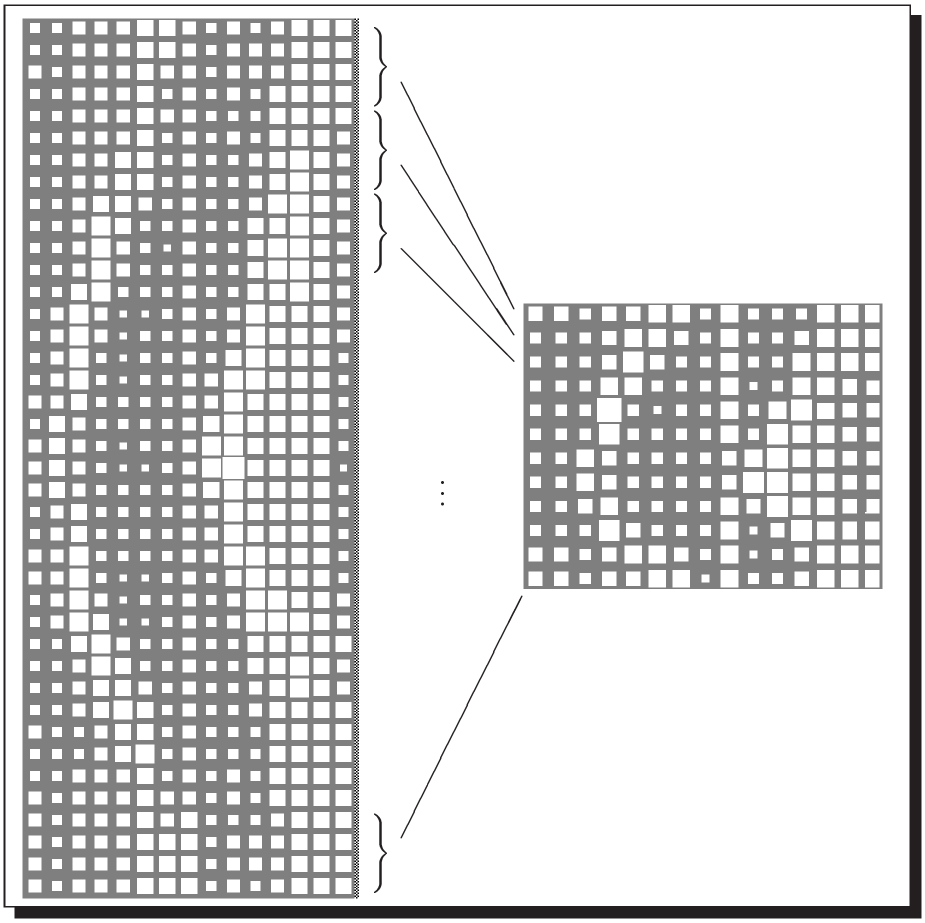


Figure 4.5.: Creating a gesture template by example: a stream of observed feature vectors on the left is compressed into the gesture template on the right.

To compensate differences in the exact timing when performing gestures, our approach uses the Viterbi algorithm for time alignment. The Viterbi alignment employs dynamic programming to find the best temporal alignment between the feature vector sequence and the gesture template (as part of the Hidden Markov Model or HMM, see [Rabiner and Juang, 1986]). It has become highly popular in the speech recognition community (cf. [Waibel and Lee, 1990]),

[1] With the system's performance of roughly 20 frames per second the gesture is approximately four seconds long.

since it compensates for variations in the timing of spoken language (and gestures). To explain the matching process, we will take a brief, yet formal, digression into Hidden Markov Models, in which we focus on discrete Hidden Markov Models for simplicity reasons. We complete the digression with an introduction of the Viterbi algorithm and how we use a similar algorithm to recognize gestures.

> "A Hidden Markov Model is a doubly stochastic process with an underlying stochastic process that is *not* observable (it is hidden), but can only be observed through another set of observed symbols."[2]

A Hidden Markov system consists of a finite number of states and state transitions. At regularly spaced, discrete times, the system undergoes a change of state according to a set of probabilities associated with the state. Furthermore, at each state one can observe an event as a probabilistic function of the state. The resulting model is a doubly embedded stochastic process with an underlying stochastic process that is hidden but can be observed only through another set of stochastic processes that produce the sequence of observations. In case of the discrete HMM, the observation symbol per state is drawn from a finite set of observation symbols. A Hidden Markov Model $\lambda = (A, B, \pi)$ is formally characterized by the following:

1. The number N of states in the model. Generally the states are connected in such a way that any state can be reached from any other state. We label the individual states as $\{1, 2, \ldots, N\}$, and denote the state at time t as q_t.
2. The number S of distinct symbols per state, i.e., the discrete alphabet size. We denote the individual symbols as $V = \{\mathbf{v}_1, \mathbf{v}_2, \ldots, \mathbf{v}_S\}$.
3. The state-transition probability distribution $A = (a)_{ij}$ where

$$a_{ij} = P[q_{t+1} = j | q_t = i], \qquad 1 \le i, j \le N. \tag{4.4}$$

Transition probabilities should satisfy the normal stochastic constraints,

$$a_{ij} \ge 0, \qquad 1 \le i, j \le N \tag{4.5}$$

and

$$\sum_{j=1}^{N} a_{ij} = 1, \qquad 1 \le i \le N. \tag{4.6}$$

[2]See [Rabiner and Juang, 1986], page 5.

4. The observation symbol probability distribution, $B = \{b_j(k)\}$, in which

$$b_j(k) = P[\mathbf{o}_t = \mathbf{v}_k | q_t = j], \qquad 1 \leq k \leq S, \tag{4.7}$$

defines the symbol distribution in state j $(j = 1, 2, \ldots, N)$. The following stochastic constraints must be satisfied:

$$b_j(k) \geq 0, \qquad 1 \leq j \leq N, 1 \leq k \leq S, \tag{4.8}$$

and

$$\sum_{k=1}^{S} b_j(k) = 1, \qquad 1 \leq j \leq N. \tag{4.9}$$

5. The initial state distribution $\pi = \{\pi_i\}$, in which

$$\pi_i = P[q_1 = i], \qquad 1 \leq i \leq N. \tag{4.10}$$

Given the form of Hidden Markov Model above, three basic problems of interest must be solved for the model to be useful in real-world applications. These problems are the following:[3]

Problem 1 Given the observation sequence $\mathbf{O} = (\mathbf{o}_1\mathbf{o}_2 \ldots \mathbf{o}_T)$, and a model $\lambda = (A, B, \pi)$, how do we efficiently compute $P(\mathbf{O}|\lambda)$, the probability of the observation sequence, given the model.

Problem 3 Given the observation sequence $\mathbf{O} = (\mathbf{o}_1\mathbf{o}_2 \ldots \mathbf{o}_T)$, and the model λ, how do we choose a corresponding state sequence $\mathbf{q} = (q_1q_2 \ldots q_T)$ that is optimal in some sense (i.e., best "explains" the observations)?

Problem 3 How do we adjust the model parameters $\lambda = (A, B, \pi)$ to maximize $P(\mathbf{O}|\lambda)$?

Problem 1 is an evaluation problem; namely given a model and a sequence of observations, how do we compute the probability that the observed sequence was produced by the model. One can also view the problem from the point of how well a given model matches a given observation sequence. Solving Problem 2 is an attempt to uncover the hidden part of the model — that is, to find the "correct" state sequence. It should be obvious that there is no "correct" state sequence in general. Typical uses might be to learn about the structure of the model or to find optimal state sequences. Finally, in Problem 3, we attempt to optimize the model parameters to create best models for real phenomena.

[3] Taken from [Rabiner and Juang, 1986].

We will focus on the Viterbi algorithm that is used to solve Problem 2. However the algorithm is very similar to the techniques used to tackle Problem 1, which matches our goal. After all, we want to know how well a gesture template (the model) matches a sequence of feature vectors (the sequence of observations).

As mentioned before, the Viterbi algorithm is used to find an optimal state sequence, given an observation and the model. In case of this approach, we use the Viterbi algorithm to align the sequence of feature vectors with a gesture template, i.e., the algorithm determines which part of the sequence of feature vectors belong to which feature vector of the template. It not only gives us a "timely" mapping between observation and template but also the corresponding probability of the state sequence, which can be viewed as the probability of observing this gesture.

To find the single best state sequence, $\mathbf{q} = (q_1 q_2 \dots q_T)$, for the given observation sequence $\mathbf{O} = (\mathbf{o}_1 \mathbf{o}_2 \dots \mathbf{o}_T)$, we define the quantity

$$\delta_t(i) = \max_{q_1, q_2, \dots, q_{t-1}} P[q_1 q_2 \dots q_{t-1}, q_t = i, \mathbf{o}_1 \mathbf{o}_2 \dots \mathbf{o}_t | \lambda] \tag{4.11}$$

that is, $\delta_t(i)$ is the highest probability along a single path, at time t, which accounts for the first t observations and ends in state i. Induction yields

$$\delta_{t+1}(j) = [\max_i \delta_t(i) a_{ij}] \cdot b_j(\mathbf{o}_{t+1}). \tag{4.12}$$

To actually retrieve the state sequence, we keep track of the argument that maximized Equation (4.12), for each t and j. The algorithm is then described as follows:

1. Initialization

$$\delta_1(i) = \pi b_i(\mathbf{o}), \quad 1 \leq i \leq N \tag{4.13}$$
$$\psi_1(i) = 0. \tag{4.14}$$

2. Recursion

$$\delta_t(j) = \max_{1 \leq i \leq N} [\delta_{t-1}(i) a_{ij}] b_j(\mathbf{o}_t), \quad 2 \leq t \leq T, 1 \leq j \leq N \tag{4.15}$$
$$\psi_t(j) = \operatorname*{argmax}_{1 \leq i \leq N} [\delta_{t-1}(i) a_{ij}], \quad 2 \leq t \leq T, 1 \leq j \leq N. \tag{4.16}$$

3. Termination

$$P^* = \max_{1 \leq i \leq N} [\delta_T(i)] \tag{4.17}$$
$$q_T^* = \operatorname*{argmax}_{1 \leq i \leq N} [\delta_T(i)]. \tag{4.18}$$

4. State sequence tracking

$$q_t^* = \psi_{t+1}(q_{t+1}^*), \quad t = T-1, T-2, \ldots, 1. \tag{4.19}$$

After this rather formal introduction of Hidden Markov Models in general and the Viterbi algorithm in particular, the reader will see that we use both in a fairly straightforward way.

To determine if a gesture has been observed, the temporal template matcher continuously analyzes the stream of incoming feature vectors for the presence of gestures. We match a gesture template (the model) with the most recent T feature vectors (the observation sequence), for varying numbers of T (in our implementation, $T = 40, 50, \ldots, 120$); notice that the gesture templates are much shorter than T (see Figure 4.6).

The Viterbi algorithm will compute the state sequence, that is, which feature vector $\mathbf{o}_k$ of the observation is mapped to what vector $\mathbf{p}_j$ of the gesture template. Furthermore, the algorithm will determine the probability of observing this sequence, i.e., how well do the last T observations actually correspond to the expected gesture. We use a Viterbi-like algorithm to align the observation $\mathbf{o}$ with a gesture template $\mathbf{p}$ (of size N, i.e., it contains N compressed feature vectors) using a matrix $(e)_{k,j}$, which is similar to the matrix $\delta_t(j)$ used in Equation (4.15).

The matrix $(e)_{k,j}$ is of size $T \times N$, and is computed from the top left to the bottom right as follows:

$$e_{k,j} = \begin{cases} ||\mathbf{o}_k - \mathbf{p}_j|| & k = 0 \wedge j = 0, \\ e_{k,j-1} & k = 0 \wedge j \neq 0, \\ e_{k-1,j} + ||\mathbf{o}_k - \mathbf{p}_j|| & k \neq 0 \wedge j = 0, \\ e_{k,j-1} & e_{k,j-1} < e_{k-1,j} + ||\mathbf{o}_k - \mathbf{p}_j||, \\ e_{k-1,j} + ||\mathbf{o}_k - \mathbf{p}_j|| & e_{k,j-1} \geq e_{k-1,j} + ||\mathbf{o}_k - \mathbf{p}_j||. \end{cases} \tag{4.20}$$

While computing $(e)_{k,j}$, we keep track of the sequence of states using the last two conditions of Equation (4.20), similar to $\psi_t(j)$ and Equation (4.19). Consequently, the element $e_{T,N}$ will determine the sequence's probability, that is, how well do the last T observations match the gesture template. To compute the match, $||\mathbf{o}_k - \mathbf{p}_j||$, between a feature vector of the observation and a feature vector of the gesture template, one could use the correlation coefficient or the Euclidean distance.

Before we introduce an application of the gesture recognition system in the next chapter, let us briefly summarize the recognition task. We recognize gestures as follows: for each incoming frame, we align the last T feature vectors with the gesture template and determine the probability $e_{T,N}$ according to Equation (4.20). If the probability is above a certain threshold, the gesture has been observed.

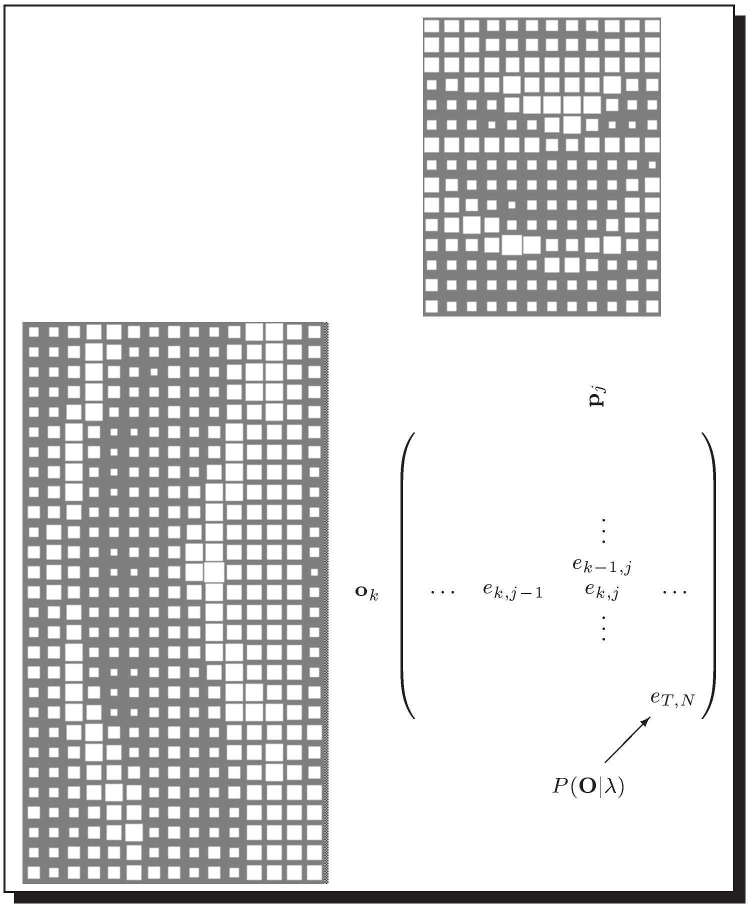


Figure 4.6.: Aligning the last T observations with a gesture template. Shown on the left are the last T feature vectors that are temporally aligned, using the matrix $(e)_{k,j}$, with the gesture template (shown on the top right, rotated 90 degrees counterclockwise).

5. Integration

5.1. Motivation

We have demonstrated the usability of the gesture recognition system in a real world scenario. After having integrated the gesture recognition system into the existing robot navigation and control software BeeSoft, a human operator can now instruct the robot to

- perform direct motion commands, e.g., stopping the robot,
- guide the robot to places which it can memorize,
- point to objects on the floor,
- initiate clean-up tasks, in which the robot searches for trash, picks it up, and delivers it to the nearest trash-bin.

In a pilot system, we have instructed our robot Amelia to pick up soda cans scattered in an office building and to deposit them into a trash-bin. This task was motivated by the "clean-up an office" task, designed for the AAAI-94 mobile robot competition (see [Simmons, 1995]). Our scenario differs, however, in that a human interacts with the robot and initiates the clean-up task, which is then performed autonomously by the robot. While this is only an example application designed to test the utility of dynamic gestures in human-robot interaction, we believe that our interface is applicable to a much larger range of upcoming service robots.

In the next section we will present a brief overview about the task, examine the different components of the system, in particular the controlling process, and conclude with an example of a successful clean-up operation.

5.2. Application

We tested the effectiveness of the gesture-based interface in the context of a clean-up task that involves human-robot interaction and mobile manipulation. The system had to detect the following gestures:

Follow The person makes as a 'Follow' gesture upon which the robot starts to follow the person around. The 'Follow' gesture involves waving the arm, as indicated in Figure 4.1(a).

Stop If the robot is shown a 'Stop' gesture, it immediately stops. The 'Stop' gesture, part of it shown in Figure 4.1(b), is a gesture that covers the raising of the hand to the 'Stop' position and the lowering of the hand to a position in which the arm is relaxed. Hence it is a dynamic gesture.

Pointing If the person points towards an object on the ground, the robot starts searching for an object within its visual field. However, the pointing gesture is only used to trigger the 'search the object' routine and not to locate the object. If the robot succeeds, it moves towards the object, picks the object up, and then returns to the nearest trash-bin. The location of trash-bins is known to the robot. In our implementation, the pointing gesture is actually a dynamic gesture, which involves moving the arm from the body towards the pointing direction and back.

The clean-up task is controlled by several software components and supervised by a single process, the *stateMachine*, which determines the next action based on the current state of the system and the input it receives. Input may come from a number of processes running on different machines, both on-board and off-board the robot as depicted in Figure 5.1.

The part of the gesture recognition system that processes the camera images to determine the feature vectors, the *gestureServer*, runs on-board the robot. Its connection to the cameraServer is made through a shared memory segment, hence enabling the system do be reasonably fast. The computed feature vectors are sent over the radio Ethernet link to an external workstation, where the recognizeServer analyzes them. The recognizeServer determines if and when a gesture has been observed and reports this to the state machine. Additionally, the buttonServer, armServer and pantiltServer run on-board the robot. They are directly supervised by the state machine.

Finally, the colliServer and baseServer, which control the motion and sensor readings on-board the robot, communicate with a second external workstation. This workstation runs the navigation software, responsible for localization, mapping and planning. The planServer reports to the state machine if a target point, e.g. a trash-bin, has been reached.

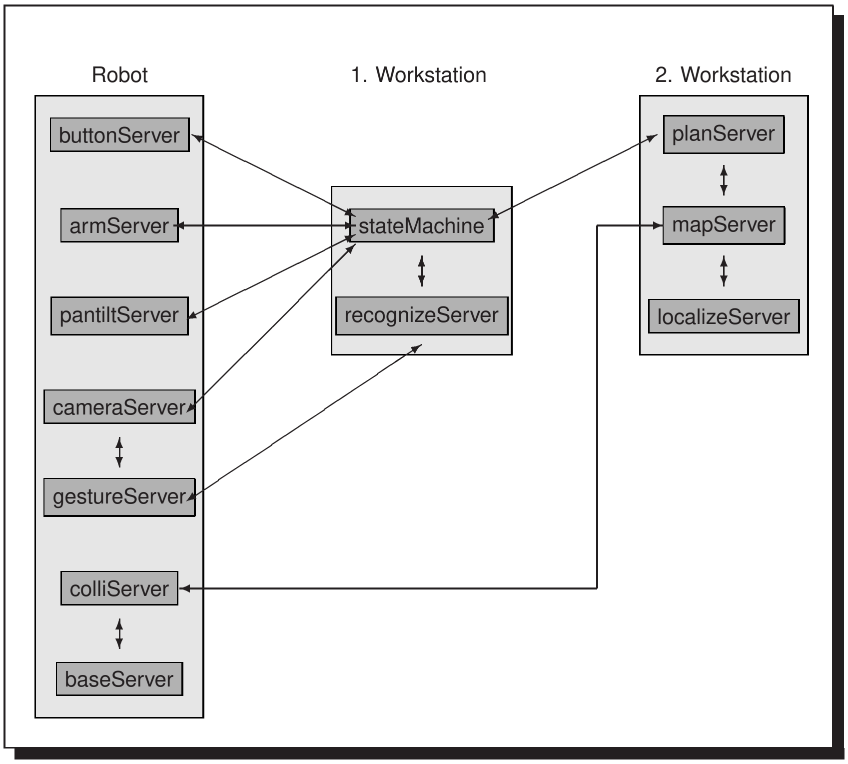


Figure 5.1.: The software used in the application and its distributed architecture.

Since the stateMachine process is the key element, which controls the entire task, we will examine it closer. The controlling process is a finite state machine with states as described in the next paragraph and state transitions as depicted in Figure 5.2. Its initial state is smIdle.

smIdle The robot roams the hallways until a person walks up to the robot, presses the green button and positions itself in front of the robot's camera.

smFindingPerson Upon finding the person in the camera image according to the algorithms described in Section 3.3, the gesture recognition process starts, i.e., the system determines and analyzes the feature vectors. The feature vectors and only the feature vectors are sent to the recognizeServer.

smWaitingForFollowGesture The recognizeServer analyzes the feature vectors until it observes either a 'Follow' gesture or an 'Abort' gesture. In the first case the robot starts following the person and the state machine proceeds to the next state. In case the system recognized an ‘Abort’ gesture, the state machine will proceed to state ‘smReturningToBase’, which means the robot will move to the point where the green button was most recently pressed.

smWaitingForStopGestureWhileFollowing The robot is tracking and following the person. The recognizeServer analyzes the feature vectors until it observes either a ‘Stop’ gesture or an ‘Abort’ gesture. Upon recognizing the ‘Stop’ gesture, the robot will stop and the state machine will proceed to the next state. An ‘Abort’ gesture is handled as mentioned above.

smWaitingForPointingGesture While still tracking the person (but not following it), the recognizeServer analyzes the feature vectors until it observes either a ‘Pointing’ gesture or an ‘Abort’ gesture. If the observed gesture is a ‘Pointing’ gesture, the state machine issues a command to point the pan-tilt unit towards the floor and proceeds to the next state. The ‘Abort’ gesture is handled as usual.

smAdjustingPantilt Point the pan-tilt unit downwards, i.e., the robot looks to the floor. Upon receiving an acknowledgement message that the camera looks towards the floor, the state machine proceeds to the next state.

smLookingForObject The robot searches in the camera image for an object with the typical size and color of a soda-can. This process repeats until an object meets a predefine ‘soda-can’ threshold. For locating the object in the image we don’t use a wide-angle lens since the object of desire would have been too small in the image. Hence, we use two lenses in this particular task — a wide-angle lens for tracking and gesture recognition and a regular lens for object detection. If the object has been found, the state machine issues a command to the colliServer to approach the object while simultaneously avoiding obstacles.

smMovingTowardsObject The robot approaches the soda-can up to a distance of approximately 40 cm. Upon receiving an acknowledgement by the colliServer that the goal point has been reached, the state machine initiates the deployment of the arm.

smDeployingArm If the arm has been completely deployed the state machine will send a message to the armServer to open the gripper.

smOpeningGripper Upon receiving a message that the gripper is open, the state machine will induce the armServer to lower the mast.

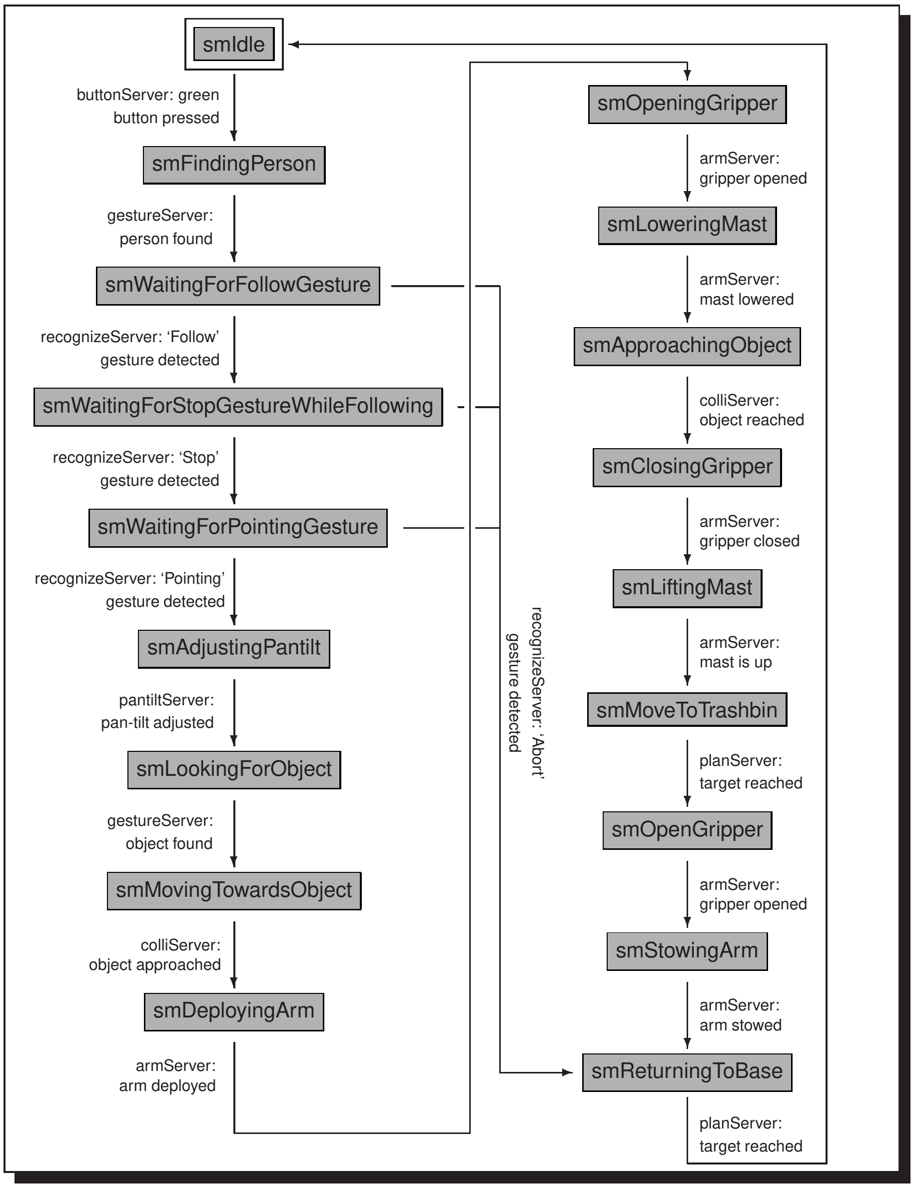


Figure 5.2.: The controlling process stateMachine, it's input and state transitions.

smLoweringMast The robot lowers the mast until the gripper is at soda-can height. The state machine will then initiate the actual 'grasping' of the object — it uses the colliServer to fully approach the object.

smApproachingObject Upon reaching the soda-can, the robot will close the gripper. We made sure that the colliServer 'overshoots' when approaching the soda can by about 5 to 10 cm. Hence we can almost guarantee that once the robot approached the can, it can grab it.

smClosingGripper Close the gripper. If the state machine receives an acknowledgement by the armServer, it will initiate the lifting of the mast.

smLiftingMast Lift the mast so that it no longer impairs the laser's view. Upon completion, send the target coordinates of the closest trash-bin to the planServer and proceed to the next state.

smMoveToTrashbin While avoiding any obstacles along the way (even with a deployed arm), the robot will approach the closest trash-bin whose coordinates in the world are known. Using the colliServer and localizeServer, the robot updates its knowledge of the environment, in particular its position, constantly. If the planServer reports that the trash-bin has been reached, proceed to the next state.

smOpenGripper Open the gripper and release the soda-can. Initiate that the arm is stowed into the robot's enclosure.

smStowingArm Upon receiving a message that the arm has been stowed, the state machine sends the coordinates of the 'base' position, i.e., the position at which a person most recently pressed the green button, to the planServer. The planServer in turn uses the mapServer and localizeServer to guide the robot to the target point.

smReturningToBase Approach the 'base' target point using the localizeServer, mapServer and planServer. If the target point has been reached, the state machine returns to its initial state 'smIdle'.

Figure 5.3 shows an example run, in which Amelia is instructed to pick up a soda can.[1] Shown there is a map of the the robot's environment, constructed using an occupancy grid technique (see [Elfes, 1989], [Moravec, 1988], and [Thrun, 1998]), along with the actual path of the robot and the (known) location of the trash-bin.

[1] The reader can see a video of the example in [Waldherr et al., 1998a].

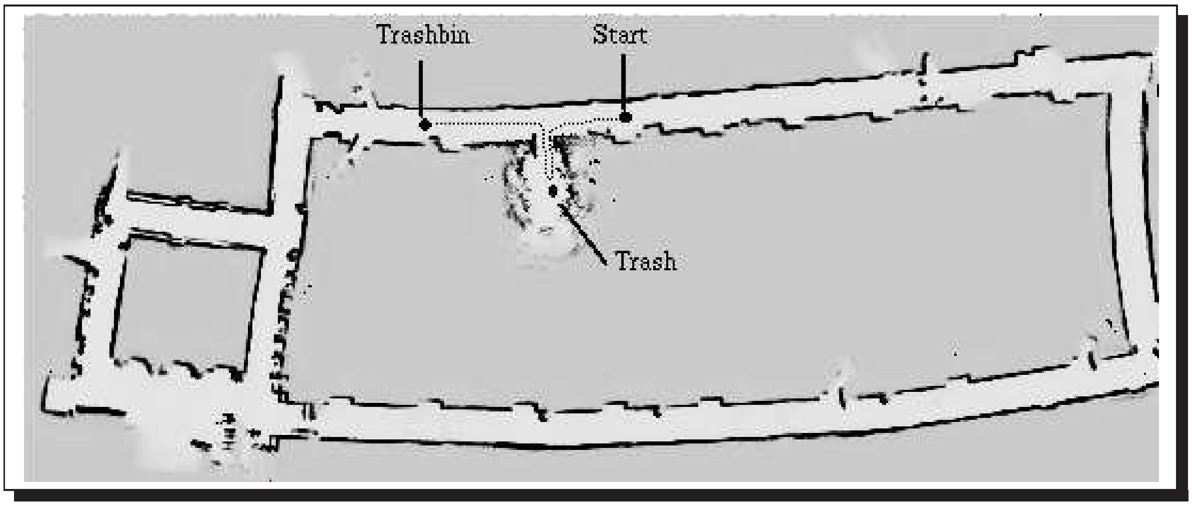


Figure 5.3.: Map of the robot's operational range (80×25 meters) with trace of a specific example of a successful clean-up operation. The robot waited in the corridor, was then guided by a human into a lab, where it picked up a can and later deposited it into a trash-bin.

Initially, the robot waited in the corridor for a person. Upon pressing the green button on the robot, the gesture recognition system started. The person instructed the robot to follow him into the lab (using the 'Follow' gesture), where it first stopped the robot (using the 'Stop' gesture) and then pointed at a piece of trash (a soda can) on the floor. The robot located the soda can, picked it up, and returned to the corridor where the robot deposited the soda can into a trash-bin.

We found that operating the robot at low speed (25 cm per second) made it easier to instruct the robot. Tests at higher speeds (45 cm per second) made it difficult to reliably position the robot close to a soda can, even though our software can manage this speed.

6. Experimental Results

We present experimental results — analogous to the gesture recognition system — in three parts: after examining the performance of finding and tracking people, we will see how well the system recognizes gestures. Finally, preliminary results in the integration task are described.

6.1. Finding and Tracking People

It is important to underline that finding the person differs subtly from tracking the person. Hence, we will first discuss some issues involved in finding the person and later describe experimental results with respect to tracking.

6.1.1. Finding People

The main difference between the finding and the tracking of a person is that we do not use the shirt color to find the person. Remember that we use both the shirt and face color for tracking. Having no previous knowledge about the person's shirt and using only an average face color model prove to be very reliable.

Experiments were conducted in a typical office building, that is, few windows and artificial light being the main source of lighting. Due to the wide angle lens of the camera, the person does not have to stand directly in front of the robot. Additionally, the person can stand between one and two meters away.

Of the more than 250 experiments we conducted, the system failed to determine the exact location of the person's face in less than 5% of the cases. Even in most of these cases, the tracking algorithm was able to cope with the deviation, quickly adapted, and relocated the face center within its search window.

However, there are drawbacks in using an average face color model. The model that we use is somewhat tailored towards humans that belong to the Caucasian ethnic group.

Hence, it will initially not behave as well with people of Indian or African American descent. Yet, the model is able to adapt.

There are simple other means of finding the person initially. Motion, detected through image differencing, could be used to detect and locate a person that steps into the view of the robot. Of course, the robot may not move during the detection. We wanted the robot to be able to move at all time. Therefore we decided against using motion and found the average face color model reliable enough.

6.1.2. Tracking People

We use a typical Intel 200 MHz Pentium machine that grabs images which are 320×240 pixels in size. By focusing the computation of the response to the color models to a narrow window around the person, we were able to obtain a performance of roughly 20 frames per second. More specifically, the Gaussian responses to the models, i.e., Equation (3.10) for M_{face} and M_{shirt} are computed only in the region of S_{face} and S_{shirt}, respectively (see Figure 3.3(f)). Furthermore, the Gaussian responses were only computed for every second pixel horizontally and vertically. Hence, dividing the computational effort by four. Adapting the covariances and means in the regions W_{face} and W_{shirt}, both of fixed size, does not pose a computational problem since both windows are relatively small.

We tested the tracking algorithm while a human guided the robot through an office building, more specifically, through rooms and hallways with different lighting conditions. The video mentioned in Section 5.2 also shows the tracking of a person in a hallway/room. Our experiments have shown that it does not pose a major restriction if we require the person's head to always be above its shoulder. We found the tracking algorithm highly reliable, even for people that moved hastily in front of the robot.

After locating the person in front of the robot, the system creates an initial model of the person's shirt color. Although the system needs some time (in the order of 30 frames) to adapt fully to the new shirt color, the system can track people with arbitrary shirt colors, as long as they are sufficiently coherent. Thus far, we tested the tracker only for various types of single-color shirts with long sleeves; experiments for multi-colored shirts were not yet conducted.

It is, however, important that the person keeps a fixed distance of about 2 meters ($\pm$ 20 cm) to the robot. Since we use windows of fixed size to adapt the face and shirt color models, it is essential that these windows contain only the face and the shirt, respectively. For example, if the distance between the person and the robot gets too large, the adaptation window W_{face} might not only capture the face but also parts of the background. Hence the color model, in particular the variance, adapts to the wrong color and it becomes more likely that the system determines the face center incorrectly. Consequently, it will not be able to track the human any longer.

6.2. Gesture Recognition

Our approach performs the recognition task in two phases. After correlating the response to the shirt color model with a set of pose templates, the system aligns the resulting feature vectors with a gesture template. We will first see, how well the system correlates the pose templates and eventually examine the results of the temporal template matcher.

6.2.1. Pose Analysis

In an example run of about 500 frames, we analyzed each feature vector, which is the result of the correlation process, to find the one pose template with the highest correlation. Figure 6.1 shows the computed pose template superimposed on the person. We recorded that

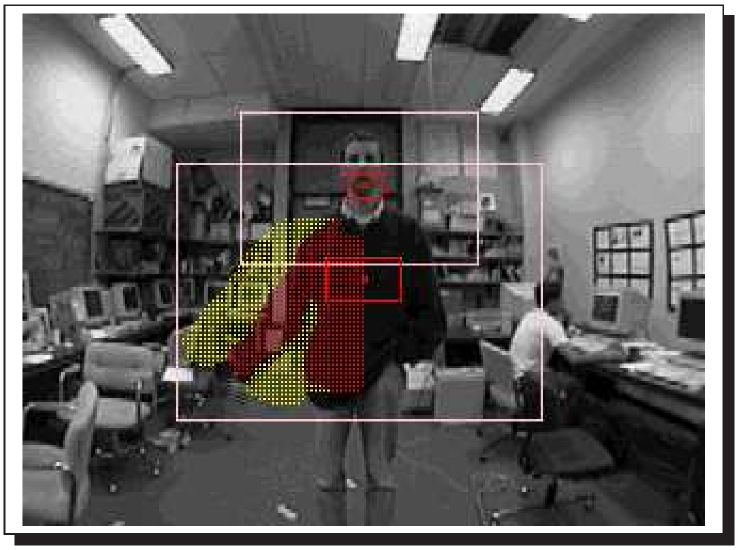


Figure 6.1.: Tracking of the person and pose analysis. Shown here are the search and adaptation windows. Additionally, the pose template whose component in the feature vector has the highest correlation is superimposed on the person.

the correct pose was identified in 85% of the cases; in 10% a nearby pose was recognized, and in only 5% the pose template matcher produced an answer that was clearly wrong.

This information, however, is of only limited use — we do not use it to recognize gestures. Instead of analyzing a sequence of most likeliest arm positions (that means, the index of the pose template with highest correlation), we use the entire correlation vector to recognize gestures. On the other hand, we wanted to include these results to demonstrate the system's performance at an intermediate step.

6.2.2. Temporal Template Matching

We present recognition results using five different motion gestures, which we briefly describe at this point.

Stop The initial arm position of the person is shown in Figure 6.2(a). The person then moves the arm to the 'Stop' position (Figure 6.2(b)), holds it there for a moment, and returns to the initial arm position.

Follow Different to the 'Stop' gesture, the initial arm position is not shown in Figure 6.2(a), but in Figure 6.2(d). Lowering the arm to the position shown in Figure 6.2(d) and raising it to the initial arm position marks the entire sequence of a 'Follow' gesture.

Point.Vertical Starting from the initial arm position depicted in Figure 6.2(a), the person moves the arm up to the position shown in Figure 6.2(f), holds it there for a brief moment, and returns to the initial arm position.

Pointing Again, starting in the initial arm position of Figure 6.2(a), the person points to an object on the floor, as shown in Figure 6.2(e), and returns to the initial arm position.

Pendulum Starting from the usual initial arm position, the person moves its arm to the position shown in Figure 6.2(f), holds this position for a moment, and returns to the initial arm position.

Note that all gestures are motion gestures, hence a gesture template consists of an entire sequence of different arm positions and *not* just a single arm-position observed over multiple frames.

The observed feature vectors were constantly analyzed by the recognizeServer and aligned to a set of gesture templates using the Viterbi-like algorithm, as described in Section 4.3. Each gesture is recognized whenever the result $e_{T,N}$ of the alignment overcomes a certain threshold. A gesture that has been recognized once can only be recognized again if $e_{T,N}$ has fallen below a second threshold. Furthermore, once a gesture has been


(a)


(b)


(c)


(d)


(e)


(f)

Figure 6.2.: Images of arm positions that are contained in the five example gestures. Each image marks a specific point in time — for example the start, middle or end position of a motion gesture.

recognized, the system does not accept a gesture in the next 50 frames (approximately 4s). This has been implemented simply to avoid recognizing the same gesture more than once in the same time window.

To cope with slightly different gestures of the same kind, we allow our system to combine several gestures into a *class* of gestures. A gesture class may only contain several gestures of the same kind. Figure 6.3 shows the display window of the recognizeServer. Shown there are the recognition results for three different classes of gestures: ‘Stop’, ‘Follow’ and

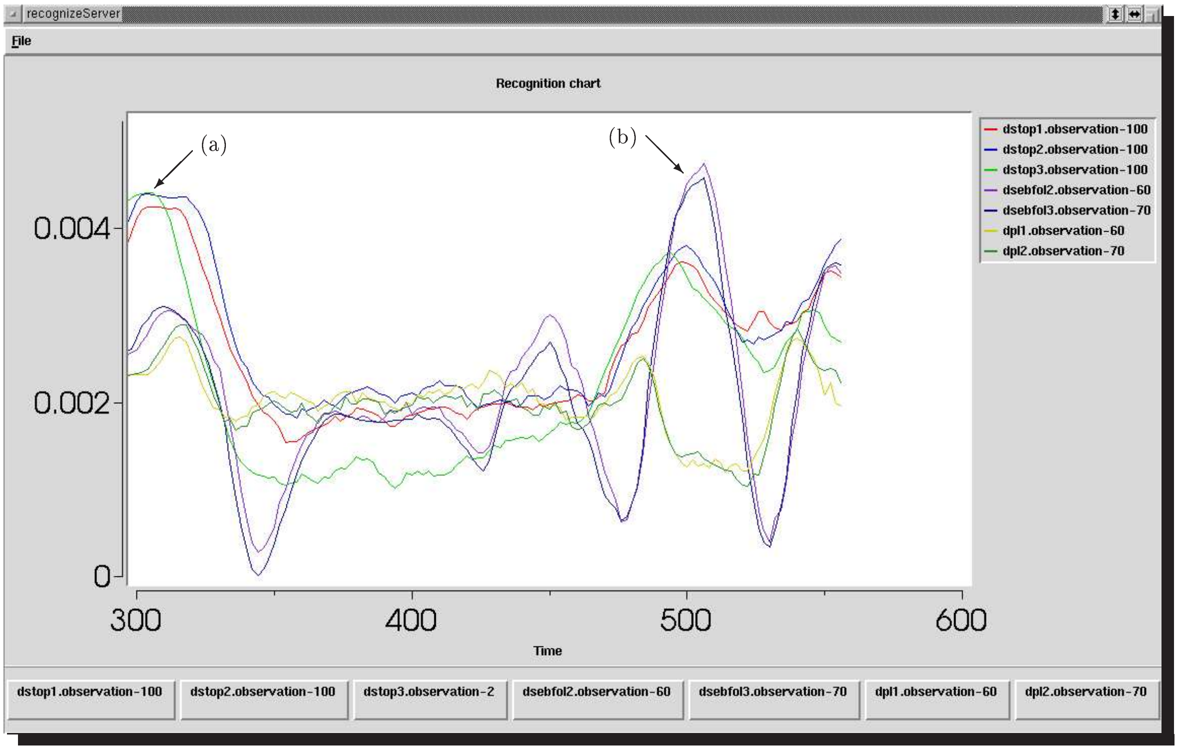


Figure 6.3.: The display window of the recognizeServer. Shown here are the results $e_{T,N}$ of the matching process over time, normalized to the length of each template.

'Pointing'. While the 'Stop' class contains three different templates (dstop1, dstop2 and dstop3), both the 'Follow' and 'Pointing' class contain only two templates. The system recognized a 'Stop' gesture at point (a) and 'Follow' gesture at point (b). See Figure 6.4 for the observed feature vectors from time $t = 400$ to $t = 508$ in between the 'Follow' gesture was given.

Additionally, a gesture template holds information about how many feature vectors should be aligned – it is important to note that the observed feature vectors can be aligned multiple times. One can see in the Figure 6.3 that the 'Follow' and 'Pointing' class have templates that in one case align the last 60, in the other case align the last 70 observed feature vectors. The observed feature vectors that have been used to detect the 'Follow' gesture are shown in Figure 6.4, the gesture template is shown in Figure 4.4(b).

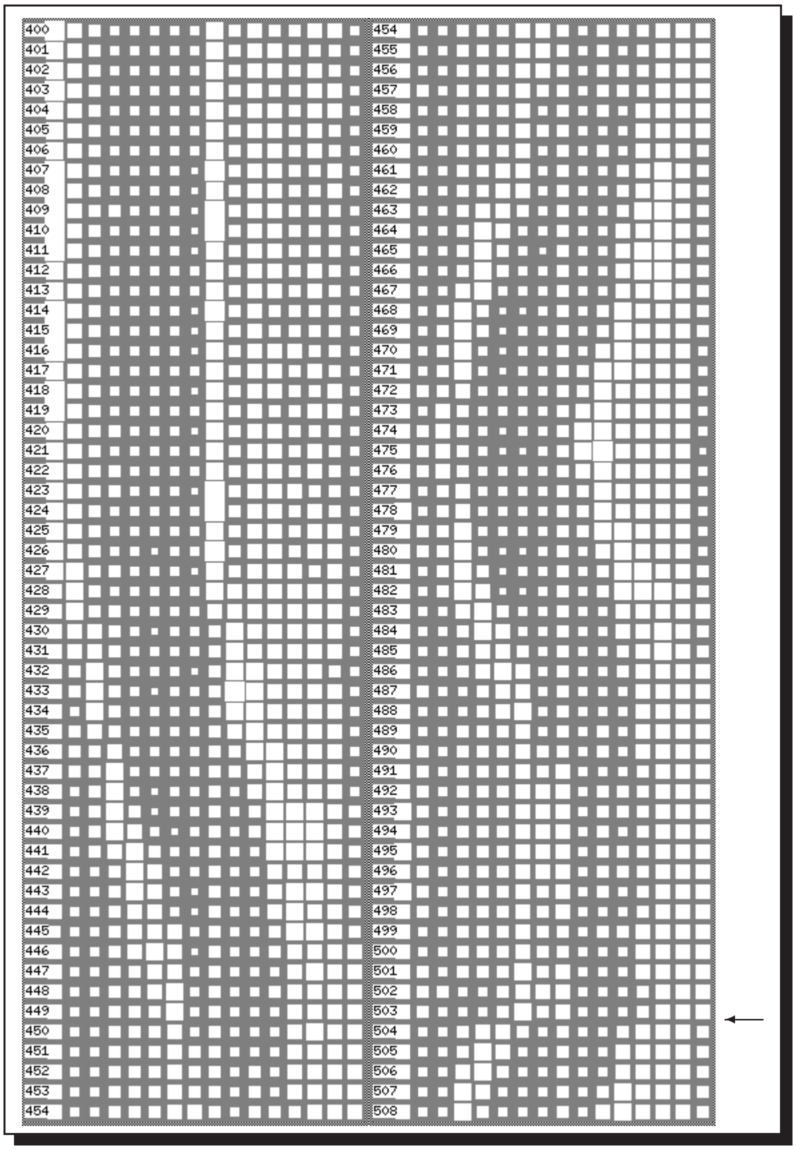


Figure 6.4.: Example of a feature vector stream that contains a 'Follow' gesture. The arrow indicates the point in time at which 'Follow' was recognized (see Figure 6.3). One of the gesture templates used in the matching is depicted in Figure 4.4(b), the output of the temporal template matcher is shown in Figure 6.3.

To test the system, a person performed a number of different gestures in front of the robot, while keeping approximately a fixed distance to the robot. However, the person could move perpendicular to the camera's line of sight.

Tables 6.1 and 6.2 show the the recognition accuracy. Each row corresponds to a number of experiments, in which a human subject presented a specific gesture. In some experiments, no gesture was shown, to test the robot's ability to detect gestures only if the person actually performed one. Since false-positives (the robot recognizes a gesture that was not shown by the instructor) are generally worse than false-negatives (the robot fails to recognize a gesture), we tuned our thresholds in such a way that the number of false-positives was small.

Gestures given		Gestures recognized					
		Stop	Follow	Point.Vert	Pointing	Pendulum	No Gesture
	261	41	60	45	42	16	57
Stop	40	39		1			
Follow	59		59				
Point.Vert	50			44			6
Pointing	42				42		
Pendulum	20	2	1			16	1
No Gesture	50						50

Table 6.1.: Recognition results. Each row shows the recognition results for examples of a specific gesture (including 50 tests where the person did not perform any gesture).

As can be seen in Tables 6.1 and 6.2, our approach recognizes gestures fairly accurately. In 211 experiments where a human showed a gesture, and an additional 50 experiments where the human did not show a gesture, the robot classified 96% of the examples correctly.

6.3. Integration

Using the scenario described in Section 5.2 we found that the robot, while following the person around, tracked the person very well. With the distributed computing as described in Sections 2.2 and 5.2 the system was able to operate at 20 frames per second.

Type	Correctly classified	Falsly classified	Not detected
Stop	98%	2%	
Follow	100%		
Point.Vert	88%		12%
Pointing	100%		
Pendulum	80%	15%	5%
No Gesture	100%		

Table 6.2.: Recognition results for five different gestures in percent. The column 'Falsely classified' marks false-positive gestures.

It is, however, crucial that the person keeps a fixed distance to the robot. This is due to the problem of fixed window sizes as mentioned in Section 6.1.2. More importantly, the system will not recognize gestures as well as it would if the human does not keep this fixed distance. Where does this distance problem stem from?

The pick-up task requires that the robot recognizes the 'Follow' gesture initially. At this point, both the robot and the human are not moving. This also holds for recognizing the 'Pointing' gesture. However, while trying to recognize the 'Stop' gesture, both the robot and the human are moving. Hence the distance between robot and human varies. Since the pose templates are of fixed size (in particular: they were recorded when the human was a certain distance away from the robot), the template matcher will not perform as well if the human is too far or too close to the robot. More specifically, the template matcher will not overcome the defined thresholds.

7. Related Work

7.1. Kortenkamp et al.

In [Huber and Kortenkamp, 1995] and [Kortenkamp et al., 1996] Kortenkamp et al. describe a system that uses stereo vision and optical flow to model people and recognize gestures. The system is capable of recognizing up to six distinct pose gestures such as pointing and hand signals and then interprets these gestures within the context of an intelligent agent architecture.

Gestures are recognized by modeling the person's head, shoulders, elbows, and hands as a set of proximity spaces. Each proximity space is a small region in the scene measuring stereo disparity and motion. The proximity spaces are connected with joints/links that constrain their position relative to each other. The robot recognizes pose gestures by examining the angles between links that connect the proximity spaces. Confidence in a gesture is build up logarithmically over time as the angles stay within limits for the gesture.

The gesture recognition system of Kortenkamp et al. runs completely on-board the robot, while our system uses off-board processing power. We are also bound to a single camera and do neither use stereo vision nor any dedicated vision hardware to process the camera images. While these issues are only differences in the physical setup of the system, our approach has a key advantage that it can recognize motion gestures.

Both systems have a high level control architecture. Our approach uses a simple state machine, Kortenkamp et al. have integrated their system into Firby's Reactive Action Package (RAP)[1]. The RAP system takes into consideration the robot's task, its current state and the state of the world and then selects a set of skills that should run to accomplish the tasks.

Kortenkamp et al. have extended their work in [Wong et al., 1995] to recognize people's faces on a mobile robot architecture.

[1] See [Firby, 1994].

7.2. Perseus

The Perseus[2] system addresses the task of recognizing when people point and finding the objected pointed to. Perseus uses a variety of techniques, called feature maps (such as intensity feature maps, edge feature maps, motion feature maps, etc.), to reliably solve this visual problem in non-engineered worlds. Perseus provides interfaces for symbolic higher level systems like the RAP reactive execution system mentioned before.

The first part of the task requires recognizing if a person is in the scene and locating him. Perseus assumes that people are the only moving objects in the scene. Hence motion can be used to segment the person from the background. If the person is segmented, the color of the clothing is known and may be used for tracking the person in the future. Knowledge about the task and environment is used at all stages of processing to best interpret the scene for the current situation. Next the important parts of his body, such as the hands and head, are found and tracked. By fusing the above mentioned feature maps, Perseus is able to reliably track parts of the body. A pointing gesture is recognized if the person's arm has moved to the side of the body and remained stationary for a short time. After the person points, the position of the head and hand is used to determine which area is pointed to. This area is searched for an object.

[Franklin et al., 1996] describes Perseus in the context of building a robot 'waiter'. In this task, Perseus has been extended to recognize a 'reach' gesture (similar to a pointing gesture) of the person's arm. The person's hand may or may not contain a soda can. Depending on this information and the current context, the robot either delivers a soda can to the person or receives a soda can from the person.

Unlike our approach, Perseus is restricted to the domain of recognizing pointing gestures. These gestures are static configuration of the person's arm and do not include motion gestures. On the other hand, Perseus is not only able to detect pointing gestures, but also capable of identifying the object pointed to. While our approach uses only color information, Perseus uses multiple independent types of information, among them intensity feature maps and disparity feature maps. Since Perseus does not model the person's arm as a whole, distance between robot and person may vary.

7.3. Pfinder

Wren et al. have extended the Artificial Live IVE (ALIVE) project (see [Maes et al., 1997]) to produce a model of people for a variety of tasks (cf. [Wren et al., 1997]). The ALIVE

[2] See [Kahn, 1996], [Kahn et al., 1996], and [Firby et al., 1995].

project creates a virtual world for a person in which the person can interact with virtual agents. While the person watches his image interacting with animated agents on a television monitor, ALIVE allows the person to interact with the agent by performing gestures such as waving for an agent to come to him.

Pfinder, or Person Finder, models people as connected sets of blobs: one for each arm and leg, the head, the torso, and the lower body. Each blob has a spatial and color Gaussian distribution associated with it. These distributions are used to track the various parts of the person.

One interesting application of this blob representation is described by Starner and Pentland in [Starner and Pentland, 1995]. The authors have associated the spatial statistics of the users hands, together with Hidden Markov Models, to interpret a 44 word subset of the American Sign Language (ASL). They were able to produce a real-time ASL interpreter with a 99% sign recognition accuracy.

The above mentioned systems have clear differences to the domain of mobile robots. Most of them assume a static background and controlled lighting conditions — assumptions which cannot hold when the robot and thus the camera is moving. Furthermore, processing power and network bandwidth in case of our mobile platform is limited. However, our approach uses similar techniques for coping with the temporal characteristics of a gesture.

7.4. Wilson, Bobick et al.

In [Wilson and Bobick, 1995a] Wilson and Bobick model gestures as view-point dependent motions of the person's body. Since gestures are intended to communicate information, the authors assume that people will actively try to make them easy to understand by orienting them in a standard way with respect to the recipient. This premise allows them to use a Hidden Markov Model to model a motion gesture by observing examples of the gesture from a single viewpoint. One major difference to our system is the assumption that the location of the person's body (hands, etc.) is known to the system. For example when recognizing gestures that are performed with the hands, the input to the system were the joint angles returned by a DataGlove.

Wilson, Bobick et al. introduce a system that uses the 3D position of the head and hands, as determined by 3D vision system, to recognize 18 different T'ai Chi gestures (see [Campbell et al., 1996]). T'ai Chi is a form of martial arts, its movements are slow, smooth, and gentle. To cope with the temporal characteristics of the gesture, the authors use a five state linear Hidden Markov Model.

In [Wilson and Bobick, 1995b] and [Wilson et al., 1996] the authors focus on representing the temporal characteristics of a gesture. They present a state-based technique for the summarization and recognition of gestures. Gestures are defined to be a sequence of states in a measurement or configuration space. Again, the authors assume known gesture-related sensory data (e.g., the movement of the hand as measured by a magnetic spatial sensor).

As with Pfinder, limitations to the domain of mobile robots, such as changing lighting conditions, do not apply here. For example, Wilson et al. assume that the background does not change. Unlike our system, they also use dedicated vision hardware to process the stereo images.

8. Summary and Conclusions

This thesis described a gesture-based interface for human-robot interaction. An approach, consisting of an adaptive color-filter and a template matcher, for recognizing human arm gestures from streams of camera images was described. The approach is capable of recognizing both static pose gestures, and dynamic motion gestures. The thesis demonstrated the usefulness of the interface in the context of a clean-up task, where a person cooperated with the robot in cleaning up trash.

We believe that finding "natural" ways of communication between humans and robots is of importance for the field of robotics, as a variety of recent changes in both robotic hardware and software suggests that service robots will soon become possible, and commercially viable. This research is intended to be a step towards this goal.

There are several open questions that warrant further research. First, our approach has several limitations. For example, the tracking module is currently unable to deal with multi-colored shirts, or to follow people who do not face the robot. We believe, however, that the robustness can be increased by considering other sources of information, such as shape and texture, when tracking people.

One main drawback of the gesture recognition system is that the human has to keep a fixed distance to the robot for good recognition results. The pose templates which are of fixed size imply this. However, one can think of ways to scale the pose templates according to the distance between human and robot. The distance can be determine easily and precisely with the laser sensor. Again, the key to more robustness seems to be the fusion of multiple sources of information.

Also interesting is the question which pose templates maximize the information that can be used for gesture recognition? It is obvious that similar looking pose templates will only increase the computational burden but will not help to distinguish between different gestures.

Secondly, our approach currently lacks a method for teaching robots new gestures. This is not really a limitation of the basic gesture-based interface, as it is a limitation of the robot's finite state machine that controls its operation. Future work will include providing the robot with the ability to learn new gestures, and to associate those with specific actions and/or locations.

Together with the existing mapping and navigation software, this should lead to a robot that can be taught a collection of tasks in novel indoor environments. Thus, by providing the robot with the ability to learn new things, we seek to explore further the practical utility of gesture-based instruction in the context of mobile service robotics. Finally, we believe it is worthwhile to augment the interface by a speech-based interface, so that both gestures and speech can be combined when instructing a mobile robot. Gestures can help to clarify spoken commands, particularly in such situations in which noise makes communication difficult if not impossible.

Acknowledgements

I want to make the acknowledgements as short as possible. I would like to thank Sebastian Thrun for being a good friend and providing excellent guidance throughout my time as a visiting scientist at Carnegie Mellon. A thesis in this form would certainly not have been possible without his support and enthusiasm.

List of Figures

List of Tables

References

[Asoh et al., 1997] Asoh, H., Hayamizu, S., Hara, I., Motomura, Y., Akaho, S., and Matsui, T. (1997). Socially embedded learning of office-conversant robot jijo-2. In *Proceedings of IJCAI-97*. IJCAI, Inc.

[Borenstein et al., 1996] Borenstein, J., Everett, B., and Feng, L. (1996). *Navigating Mobile Robots: Systems and Techniques.* A. K. Peters, Ltd., Wellesley, MA.

[Burgard et al., 1997a] Burgard, W., Fox, D., and Hennig, D. (1997a). Fast grid-based position tracking for mobile robots. In *Proceedings of the 21th German Conference on Artificial Intelligence*, pages 289–300, Berlin. Springer Verlag.

[Burgard et al., 1996] Burgard, W., Fox, D., Hennig, D., and Schmidt, T. (1996). Estimating the absolute position of a mobile robot using position probability grids. In *Proceedings of the Thirteenth National Conference on Artificial Intelligence*, Menlo Park, CA. AAAI, AAAI Press/MIT Press.

[Burgard et al., 1997b] Burgard, W., Fox, D., Lakemeyer, G., Hähnel, D., Schulz, D., Steiner, W., Thrun, S., and Cremers, A. (1997b). Real robots for the real world — the RHINO museum tour-guide project. (submitted for publication).

[Campbell et al., 1996] Campbell, W., Becker, A., Azarbayejani, A., Bobick, A., and Pentland, A. (1996). Invariant features for 3-d gesture recognition. Technical Report 379, M.I.T. Media Laboratory Perceptual Computing Section.

[Cox and Wilfong, 1990] Cox, I. and Wilfong, G., editors (1990). *Autonomous Robot Vehicles.* Springer Verlag, Berlin.

[Crowley, 1997] Crowley, J. (1997). Vision for man-machine interaction. *Robotics and Autonomous Systems*, 19:347–358.

[Darrel et al., 1996] Darrel, T., Moghaddam, B., and Pentland, A. (1996). Active face tracking and pose estimation in an interactive room. In *Proceedings of the IEEE Sixth International Conference on Computer Vision*, pages 67–72.

[Elfes, 1989] Elfes, A. (1989). *Occupancy Grids: A Probabilistic Framework for Robot Perception and Navigation.* PhD thesis, Department of Electrical and Computer Engineering, Carnegie Mellon University.

[Fedor, 1994] Fedor, C. (1994). TCX: An interprocess communication system for building robot architectures. Robotics Institute, School of Computer Science, Carnegie Mellon University.

[Firby et al., 1995] Firby, R., Kahn, R., Prokopowicz, P., and Swain, M. (1995). An architecture for active vision and action. In *Proceedings of IJCAI-95*, pages 72–79.

[Firby, 1994] Firby, R. J. (1994). Task networks for controlling continous processes. In *Proceedings of the Second International Conference on AI Planning and Application.* 1994.

[Fox et al., 1997] Fox, D., Burgard, W., and Thrun, S. (1997). The dynamic window approach to collision avoidance. *IEEE Robotics and Automation*, 4(1).

[Fox et al., 1998] Fox, D., Burgard, W., and Thrun, S. (1998). A hybrid collision avoidance method for mobile robots. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA).* to appear.

[Franklin et al., 1996] Franklin, D., Kahn, R., Swain, M., and Firby, J. (1996). Happy patrons make better tippers. University of Chicago, Computer Science Department.

[Huber and Kortenkamp, 1995] Huber, E. and Kortenkamp, D. (1995). Using stereo vision to pursue moving agents with a mobile robot. In *Proceedings of the IEEE International Conference on Robotics and Automation.*

[Kahn, 1996] Kahn, R. (1996). *Perseus: An Extensible Vision System For Human-Machine Interaction.* PhD thesis, University of Chicago.

[Kahn et al., 1996] Kahn, R., Swain, M., Prokopowicz, P., and Firby, R. (1996). Gesture recognition using the perseus architecture. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 734–741, San Francisco, CA.

[King and Weiman, 1990] King, S. and Weiman, C. (1990). Helpmate autonomous mobile robot navigation system. In *Proceedings of the SPIE Conference on Mobile Robots*, pages 190–198, Boston, MA. Volume 2352.

[Kortenkamp et al., 1998] Kortenkamp, D., Bonassi, R., and Murphy, R., editors (1998). *AI-based Mobile Robots: Case studies of successful robot systems*, Cambridge, MA. MIT Press. to appear.

[Kortenkamp et al., 1996] Kortenkamp, D., Huber, E., and Bonasso, P. (1996). Recognizing and interpreting gestures on a mobile robot. In *Proceedings of AAAI-96*, pages 915–921. AAAI Press/The MIT Press.

[Maes et al., 1997] Maes, P., Blumberg, B., Darrel, T., and Pentland, A. (1997). The ALIVE system: Full body interaction with animated autonomous agents. ACM Multimedia Systems.

[Moravec, 1988] Moravec, H. P. (1988). Sensor fusion in certainty grids for mobile robots. *AI Magazine*, pages 61–74.

[Rabiner and Juang, 1986] Rabiner, L. and Juang, B. (1986). An Introduction to Hidden Markov Models. In *IEEE ASSP Magazine*, pages 4–16. 1986(1).

[Simmons, 1995] Simmons, R. (1995). The 1994 AAAI robot competition and exhibition. *AI Magazine*, 16(1).

[Starner and Pentland, 1995] Starner, T. and Pentland, A. (1995). Real-time american sign language recognition from video using hidden markov models. In *Proceedings of International Symposium on Computer Vision*. IEEE Computer Society Press.

[Thrun, 1998] Thrun, S. (1998). Learning maps for indoor mobile robot navigation. *Artificial Intelligence*. to appear.

[Thrun et al., 1997] Thrun, S., Bücken, A., Burgard, W., Fox, D., Fröhlinghaus, T., Hennig, D., Hofmann, T., Krell, M., and Schimdt, T. (1997). Map learning and high-speed navigation in RHINO. In Kortenkamp, D., Bonasso, R., and Murphy, R., editors, *AI-based Mobile Robots: Case studies of successful robot systems*. MIT Press, Cambridge, MA. to appear.

[Torrance, 1994] Torrance, M. C. (1994). Natural communication with robots. Master's thesis, MIT Department of Electrical Engineering and Computer Science, Cambridge, MA.

[Waibel and Lee, 1990] Waibel, A. and Lee, K.-F., editors (1990). *Readings in Speech Recognition*. Morgan Kaufmann Publishers, San Mateo, CA.

[Waldherr et al., 1998a] Waldherr, S., Thrun, S., Margaritis, D., and Romero, R. (1998a). `http://www.waldherr.org/research/aaai.mpeg`.

[Waldherr et al., 1998b] Waldherr, S., Thrun, S., Margaritis, D., and Romero, R. (1998b). Template-based recognition of pose and motion gestures on a mobile robot. In *Proceedings of the Fifteenth National Conference on Artificial Intelligence*. 1998.

[Wilson and Bobick, 1995a] Wilson, A. and Bobick, A. (1995a). Learning visual behavior for gesture analysis. Technical Report 337, M.I.T. Media Laboratory Perceptual Computing Section.

[Wilson and Bobick, 1995b] Wilson, A. and Bobick, A. (1995b). Using configuration states for the representation and recognition of gestures. Technical Report 308, M.I.T. Media Laboratory Perceptual Computing Section.

[Wilson et al., 1996] Wilson, A., Bobick, A., and Cassell, J. (1996). Recovering the temporal structure of natural gesture. Technical Report 388, M.I.T. Media Laboratory Perceptual Computing Section.

[Wong et al., 1995] Wong, C., Kortenkamp, D., and Speich, M. (1995). A mobile robot that recognizes people. In *Proceedings of the IEEE International Conference on Tools with Artificial Intelligence*.

[Wren et al., 1997] Wren, C., Azarbayejani, A., Darrell, T., and Pentland, A. (1997). Pfinder: Real-time tracking of the human body. *IEEE Transactions on Pattern Analysis and Machine Learning*, 19(7):780–785.

[Wyszecki and Styles, 1982] Wyszecki, G. and Styles, W. (1982). *Color Science: Concepts and Methods, Quantitative Data and Formulae.* John Wiley & Sons.

[Yang and Waibel, 1995] Yang, J. and Waibel, A. (1995). Tracking human faces in real-time. Technical Report CMU-CS-95-210, School of Computer Science, Carnegie Mellon University.

Index

Nomenclature

$(e)_{k,j}$ Matrix used in the Viterbi algorithm to align observations with a gesture template.

α Determines how quickly the system adapts to changing lighting conditions.

$\delta_t(i)$ Highest probability along a single path, at time t, which accounts for the first t observations and ends in state i.

λ A Hidden Markov Model.

$\mathbf{\Sigma}$ Covariance as defined in Equation (3.6).

$\mathbf{I}$ The Gaussian response to the current shirt color model in the region of a pose template.

$\mathbf{O}$ Observation sequence.

$\mathbf{o}$ The last T observations, i.e. feature vectors.

$\mathbf{o}_k$ Feature vector k of the current observation.

$\mathbf{p}$ A gesture template.

$\mathbf{p}_k$ Feature vector k of the gesture template.

$\mathbf{q}$ State sequence.

$\mathbf{T}$ A pose template.

$\mu_{\mathbf{I}}$ Mean of the response to the current shirt color model.

$\mu_{\mathbf{T}}$ Mean of a pose template.

μ_g Mean of all normalized green values.

μ_r Mean of all normalized red values.

π Initial state distribution.

$\psi_t(j)$ Matrix to keep track of the state sequence.

$\varrho_{\mathbf{T}I}$ Coefficient to determine the correlation between the pose template and the response to the current shirt color model.

ϱ_{rg} Coefficient to determine the correlation of normalized red and green colors in an image.

A State-transition probability distribution $(a)_{ij}$.

B Observation symbol probability distribution $b_j(k)$.

g A pixel's green value in chromatic color space.

M Number of states.

M Statistical representation of a color model.

$M_{\textsf{face}}$ Color model for the face color.

$M_{\textsf{shirt}}$ Color model for the shirt color.

P Probability of the observation sequence, given the model λ.

q_t State at time t.

R Number of pose templates.

r A pixel's red value in chromatic color space.

S Number of distinct symbols per state.

$S_{\textsf{face}}$ Window in which we compute the Gaussian response using the face color model.

$S_{\textsf{shirt}}$ Window in which we compute the Gaussian response using the shirt color model.

T Number of most recent feature vectors that are aligned with a gesture template.

V Set of individual symbols.

$W_{\textsf{face}}$ Window in which we compute or adapt the face color model.

$W_{\textsf{shirt}}$ Window in which we compute or adapt the shirt color model.

Erklärung:

Hiermit versichere ich, daß ich die vorliegende Arbeit selbständig verfaßt und ausschließlich die angegebenen Hilfsmittel und Quellen verwendet habe.

Pittsburgh/Kiel im September 1998.

(Stefan Waldherr)