Sensor-based Distributed Control Scheme for Mobile Robots

Mohamed Dekhil, Tarek M. Sobh, and Alyosha A. Efros

UUSC-95-005

Department of Computer Science
University of Utah
Salt Lake City, UT 84112 USA

May 1995

Abstract

In this paper we present a sensor-based distributed control scheme for mobile robots. This scheme combines centralized and decentralized control strategies. A server-client model is used to implement this scheme where the server is a process that caries out the commands to be executed, and each client is a process with a certain task. The clients are running in parallel issuing commands to the server which selects the command to be executed based on a predefined priority scheme. In this scheme, a collision avoidance client is running all the time with the highest priority. This improves the performance of the other clients since it removes the burden of avoiding obstacles and allows each client to concentrate on performing its task. Some aspects of tolerance analysis are investigated and incorporated in the proposed scheme. The logical sensor approach is used to model the sensory system which provides different levels of data representation with tolerance measures and analysis. The simulation results of this model are presented with a brief discussion and conclusion on these results.

1 Introduction

In any closed-loop control system, sensors are used to provide the feedback information that represents the current status of the system and the environmental uncertainties. The main component in such systems is the transformation of sensor outputs to the decision space, then the computation of the error signals and the joint-level commands (see Figure 1). For example, the sensor readings might be the current tool position, the error signal the difference between the desired and current position at this moment, and finally, the joint-level command will be the required actuator torque/force.

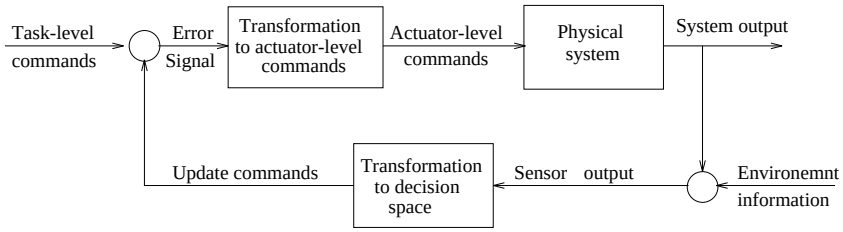

Figure 1: Closed loop control system.

The sensors used in the control scheme shown in Figure 1 are considered to be passive elements that provide raw data to a central controller. The central controller computes the next command based on the required task and the sensor readings. The disadvantage of this scheme is that the central controller may become a bottleneck when the number of sensors increases which may lead to longer response time. By response time we mean the time between two consecutive commands. In some applications the required response time may vary according to the required task and the environment status. For example, in autonomous mobile robot with the task of reaching a destination position while avoiding unknown obstacles, the time to reach to the required position may not be important, however, the response time for avoiding obstacles is critical and requires fast response.

Fast response can be achieved by allowing sensors to send commands directly to the physical system when quick attention is required. This is analogous to human reactions to some events. In the normal cases, the sensory systems in humans (e.g., eye, ear, nerves, etc.) sends perceived data to the brain (the central controller) which analyze this data and decides the next action to be taken based on the result of the analysis and the required task to be done. However, humans have a very fast contracting reaction when touching hot surfaces for example. In such cases, this reaction behavior is due to commands sent directly from the nerves at the skin spot where the touch

occurred to the muscles, bypassing the brain.

In this work, several controllers (clients) are working in parallel, competing for the server. The server selects the command to be executed based on a dynamically configured priority scheme. Each of these clients has a certain task, and can use the sensor readings to achieve its goal. A special client with the task of avoiding obstacles is assigned the highest priority. The clients needs to know the current state of the system and the command history to update their control strategy. Therefore, the server has to broadcast the selected command and the current state of the system.

Another aspect of this work is incorporating tolerance analysis and measures into the used sensory system. This provides quantitative measures for the accuracy of the location of measured points. It also serves as the basis for devising sensing strategies to enhance the measured data for localization and map construction.

The logical sensor approach, which we used to model the sensory system in our mobile robot, allows flexible and modular design of the controllers. It also provides several levels of data abstraction and tolerance analysis based on the sensor type and the required task. The initial work on this project is described in [4]. This approach is used to build high-level requests which may be used by the application program. These requests include measuring data points within a specific tolerance or within a certain time limit. This idea is demonstrated in the results in Section 5.

A brief background and related work in sensor-based control and mobile robots is presented in Section 2. The proposed control scheme is described in Section 3. Section 4 discusses the tolerance model for sonar sensors and the proposed methods to increase the accuracy of the sensory system. Some experiments and simulation results are presented in Section 5. Finally, our conclusions about this work and the future extensions are presented in Section 6.

2 Related Work

There has been a tremendous amount of research in the area of sensor-based control including sensor modeling, multisensor integration, and distributed control schemes for robotic applications in general and mobile robots in particular.

A sensor-based control using a general learning algorithm was suggested by Miller [14]. This approach uses a learning controller that learns to reproduce the relationship between the sensor outputs and the system command variables. Another technique for sensor-based obstruction avoidance for mobile robots was proposed by Ahluwalia and Hsu [2]. In their technique, the robot is able to move through an unknown environment while avoiding obstacles. Simulations were carried out assuming the robot had eight tactile sensors and the world is modeled as a two-dimensional occupancy matrix with 0's representing empty cells and 1's representing occupied cells. Another method for sensor-based obstruction avoidance was proposed by Gourley and Trivedi [5] using a quick and efficient algorithm for obstacle avoidance.

Hagar proposed a novel approach for sensor-based decision making system [6]. His approach is based on formulating and solving large systems of parametric constraints. These constraints describe both the sensor data model and the criteria for correct decisions about the data.

There has been a fair amount of research in developing languages for sensor-based control for robot manipulators. The goal of such languages is to provide an easy tool for writing adaptive robotic controller. Some of these languages are described in [16]. Several research activities for sensor-based control for robotic applications can be found in [11].

Lin and Tummala [12] described an adaptive sensor integration mechanism for mobile robot navigation. They divided the navigation process into three phases:

Sensing: firing different sensors then sending the perceived data to the data processor.

Integration: interpreting sensory data of different types into a uniform representation.

Decision: Deciding the action plan based on the current workspace representation.

Luo and Kay [13] conducted a survey on multisensor-based mobile robots. In their survey, the presented a number of control strategies that has been used in this area.

A distributed decentralized control scheme is proposed by Mutambara and Durrant-Whyte [15]. This scheme provides flexible, modular and scalable robot control network. This scheme uses a non-fully connected control components, which reduces the number of interconnections and thus reducing the number of required communication channels.

The idea of *smart sensing* was investigated by several researchers. Yakovleff et al. [21] represented a dual purpose interpretation for sensory information; one for collision avoidance (reactive control), and the other for path planning (navigation). The selection between the two interpretation is dynamic depending on the positions and velocities of the objects in the environment. Budenske and Gini [3] addressed the problem of navigating a robot through an unknown environment, and the need for multiple algorithms and multiple sensing strategies for different situations.

Discrete Event Systems (DES) is used as a platform for modeling the robot behaviors and tasks, and to represent the possible events and the actions to be taken for each event. A framework for modeling robotic behaviors and tasks using DES formalism was proposed by Košecká et al. [10]. In this framework, there are two kinds of scenarios. In the first one, reactive behaviors directly connects observations (sensor readings) with actions. In the second, observations are implicitly connected with actions through an observer.

In our proposed control scheme, the sensory system can be viewed as passive or *dumb* element which provides raw data. It can be viewed as an *intelligent* element which returns some "analyzed" information. Finally it can be vised as a *commanding* element which sends commands to the physical system. Each of these views is used in different situations and for different tasks. A detailed description of the proposed control scheme is presented in the following section.

3 The Proposed Control Scheme

The robot behavior can be described as a function $\mathcal{F}$ that maps a set of events $\mathcal{E}$ to a set of actions $\mathcal{A}$. This can be expressed as:

$$\mathcal{F}\colon \mathcal{E} \longrightarrow \mathcal{A}$$

The task of the robot controller is to realize this behavior. In general we can define the controller as a set of pairs:

$$\{(e_1, a_1), (e_2, a_2), \ldots, (e_n, a_n)\}$$

where $e_i \in \mathcal{E}$, and $a_i \in \mathcal{A}$

The events can be defined as the interpretation of the raw data perceived by the sensors. Let's define the function $\mathcal{T}$ which maps raw data $\mathcal{R}$ to events $\mathcal{E}$:

$$\mathcal{T}\colon \mathcal{R} \longrightarrow \mathcal{E}$$

The functions $\mathcal{T}$ and $\mathcal{F}$ can be closed form equations, lookup tables, or inference engine of an expert system. This depends on the kind of application and the complexity of each transformation.

3.1 Abstract Sensor Model

We can view the sensory system using three different levels of abstractions (see Figure 2.)

1. **Dumb sensor:** which returns raw data without any interpretation. For example, a range sensor might return a real number representing the distance to an object in inches, and a camera may return an integer matrix representing the intensity levels of each pixel in the image.

2. **Intelligent sensor:** which interprets the raw data into an event using the function $\mathcal{T}$. For example, the sensor might return something like "will hit an object," or "a can of Coke is found."

3. **Controlling sensor:** which can issue commands based on the received events. for example, the sensor may issue the command "stop" or "turn left" when it finds an obstacle ahead. In this case, the functions $\mathcal{F}$ and $\mathcal{T}$ should be included in the abstract model of the sensor.

The dumb sensor can be used as a source for the feedback information required by the control system. It can be also used to gather measurements to construct a map for the surrounding environment. The process that uses a dumb sensor as a source of information needs to know the type of that sensor, the format of the data the sensor returns, and the location of the sensor, to be able to interpret the perceived data. The intelligent sensor may be used for monitoring activities.

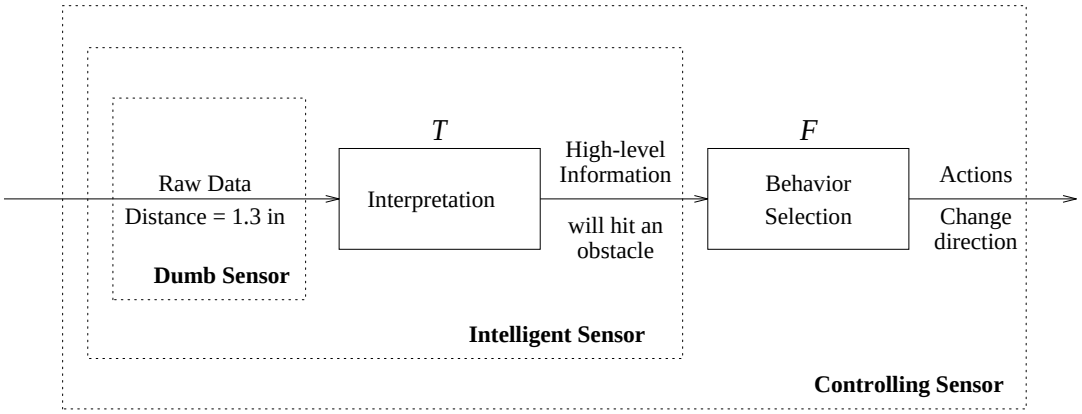

Figure 2: Three levels to view a sensor module.

The process that uses an intelligent sensor. needs to know only the event domain and maybe the location of the sensor. On the other hand, the commanding sensor is considered to be a "client" process that issues commands to the system.

3.2 A Distributed Control Architecture

Several sensors can be grouped together representing a logical sensor [7, 19]. We will assume that each logical sensor is represented as a client process which sends commands through a chanel to a multiplexer (the server process) which decides the command to be executed first. Besides these logical sensors, we might have other processes (general controllers) that send commands to the server process to carry out some global goals. Figure 3 shows a schematic diagram for the proposed control scheme.

Let's call any process that issues commands to the server a *client process*. In this figure, there are three types of clients:

1. Commanding sensors, that are usually used for reaction control and collision avoidance.
2. General Controllers, that carry out a general goal to be achieved (e.g., navigating from one position to another.)
3. Emergency exits, which bypass the multiplexer in case of emergencies (e.g., emergency stop when hitting an obstacle.)

In most cases, the general controllers require feedback information to update their control parameters. This information is supplied by dumb sensors in form of raw data, or by intelligent

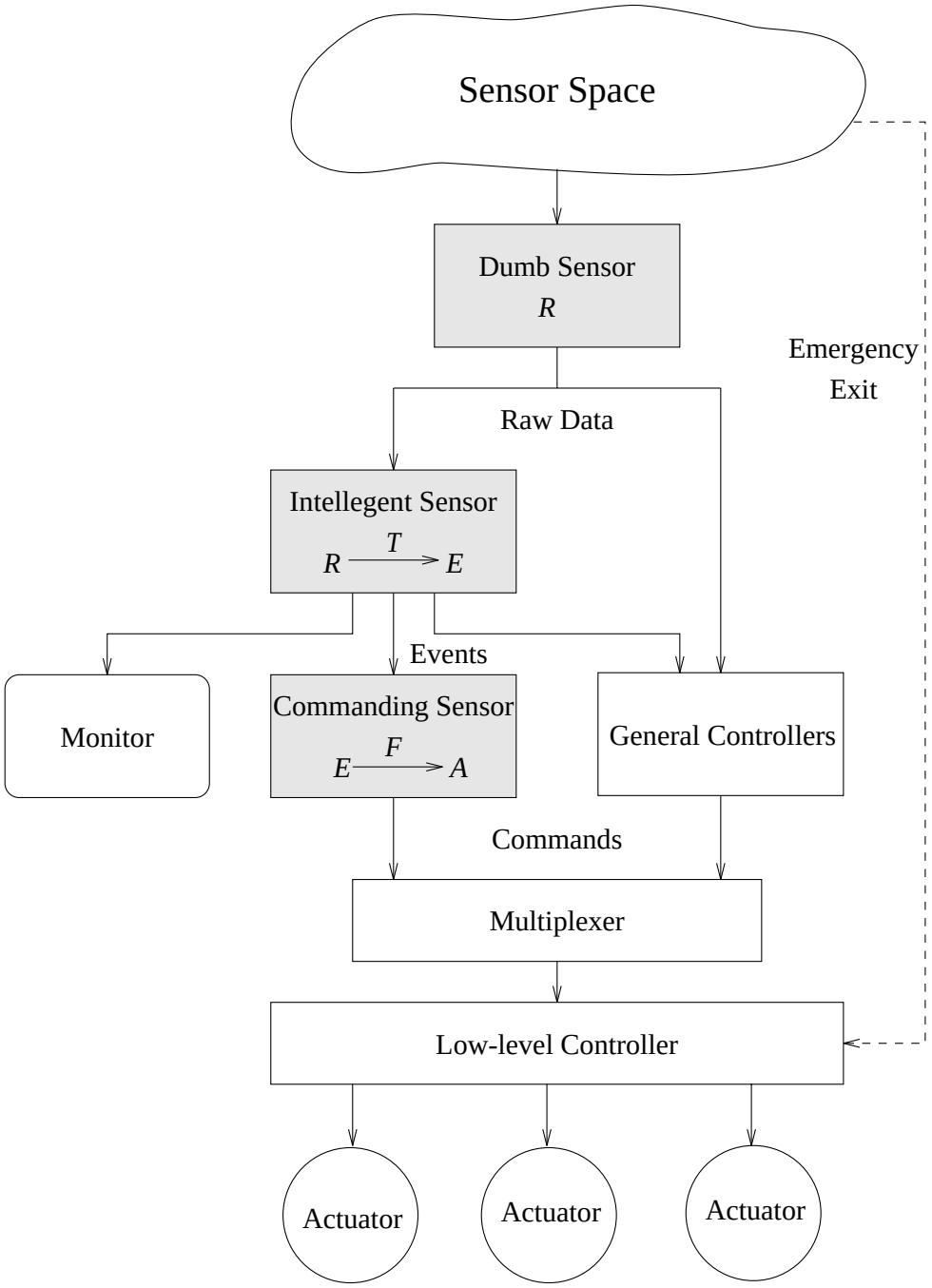

Figure 3: The proposed control scheme.

sensors in form of events. On the other hand, a monitoring process might use only intelligent sensors as a source of "high-level" events instead of raw data. All clients (except for the emergency exists) send the commands to a multiplexer. The multiplexer selects the command to be executed based on a priority scheme which depends on the current state of the system and the type of operation the client is performing. Once a command is selected, all other commands can be ignored, since the state of the system will change after executing the selected command.

The low-level controller, shown in Figure 3, translates the high-level commands into low-level instructions which drive the system's actuators. The low-level controller receives its commands either form the multiplexer or from an emergency exit. After the command is executed, the system state is updated, and the sensor space is changed. New sensor readings are received and the cycle is repeated.

3.3 Communication Protocols

In the proposed control scheme, there are several clients sending commands asynchronously to the server. Therefore, we need to define a communication protocol to organize these commands, and to set a priority scheme for selecting the command to be executed first. In most cases, the clients need to know the current state of the system and the command history to update their control strategy. Therefore, the server has to broadcast the selected command and the current state of the system.

Each client may send commands to the server (through multiplexer) at any time. Each command is associated with the signature of the sender. This signature includes the name and type of the sender, and the priority value. In most cases, the reaction commands (usually from a commanding sensor to avoid collision) has a higher priority than any other client. The priority among the client may be specified by the user and/or by the current state of the system. Emergency exits should always bypass the multiplexer and sends its commands directly to the low-level controller.

The message passing paradigm is used for process communication. This allows processes to be running on different platforms without the need for shared memory. In our implementation, MPI, Message-Passing Interface [1] was used because of its portability and to workstation clusters and heterogenous networks of workstations. It also provides an easy-to-use library functions to carry out the required communication protocols.

3.4 Time vs. Accuracy

The most important criteria in any sensory system are *time* and *accuracy*. Time is the time elapsed between issuing a read request to the logical sensor and the reply to that request. This time depends on the physical aspects of the sensory system, and on the sensing strategy implemented in the logical sensor. Tolerance is defined in this scheme as the region in which the measurement resides.

The following are some variables that will be used in the tolerance analysis for our experiment.

- v_s : sound velocity.
- y_{max} : maximum distance in our indoor environment.
- y_{min} : minimum distance in our indoor environment.
- t_m : the maximum time to get a measurement by the physical sonar sensor.

$$t_m = 2y_{max}/v_s$$

- v_r : the linear velocity of the robot in meter/sec.
- ω_r : the angular velocity of the robot in rad/sec.
- t_d : decision time; the time to decide the next action based on the current reading.

  In most cases, we cannot satisfy both requirement at the same time. Since the physical sensor has its accuracy limitations, therefore, we might need to get several readings regarding the same measured point to increase the accuracy. This of course with increase the time of measurement. In case of multisensor system, the accuracy can be increased by considering the readings from more than one sensor. In such cases, we should consider the time of the data fusion algorithms used.

4 Tolerance Analysis for Sonar Sensors

Sonar sensors have been widely used in several robotic applications. This is due to their low cost and reasonable reliability. However, sonar sensors, like most ultrasonic sensors, have several drawbacks. One problem is that the data measurements depends on the speed of sound, which vary according to the atmospheric conditions such as temperature and humidity. This usually results in inaccurate and inconsistent readings. Another problem is that the ultrasonic echoes might cause the sensor to measure totally incorrect values.

The use of ultrasonic sensors for mobile robots has been investigated by a lot of researcher [3, 8, 9, 17]. The main goal is to increase the accuracy and reliability of these sensors, and to filter the noise and echoes to get more consistent data.

In this analysis, we will ignore measurement noise, and errors due to the interference between the sensors. Also, we will use a simplified beam pattern for the sonar sensors which is a conic shape centered at the center of the sensor. Our goal here is to be able to determine the location of the measured point within a certain tolerance. Also, we would like to locate edges and door ways within a reasonable tolerance. First, the case of using one sensor will be considered, then the use of multiple sensors to get more accurate measurements will be discussed.

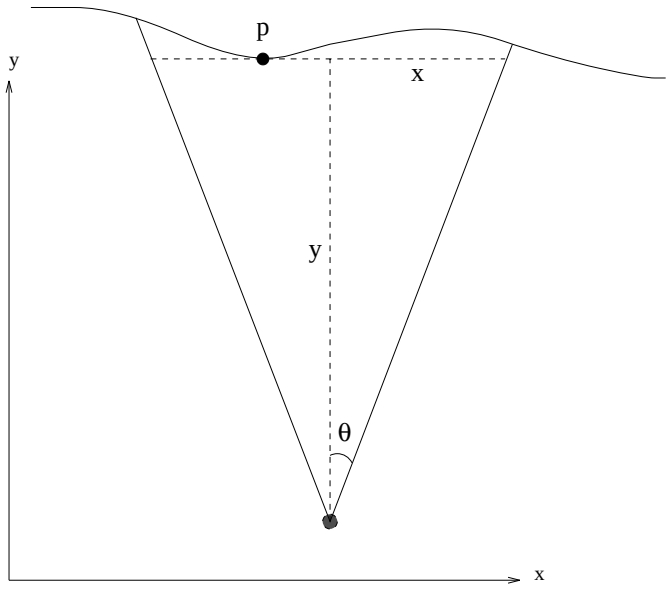

Figure 4: The simplified sonar model.

4.1 Using One Sensor

Figure 4 shows the simplified beam pattern of a sonar sensor. We assume that the sensor will return the distance y of the closest point p from the center of the sensor within a tolerance region $2x$, where $x = y \tan \theta$.

It is clear that the tolerance area depends on the angle θ and the distance y. However, the angle is fixed for most sonar sensors (It may vary according to the operating frequency.) In our experiment for example, $\theta = 11$. Also, there is physical limitations on the minimum distance y_{min} which means there is an upper limit on the accuracy that we can get with the sensor. The upper limit is:

$$x_{min} = y_{min} \tan \theta$$

.

There are several ways to minimize the tolerance region and to detect the existence of edges within this region. One way is to move in the y direction towards the measured point. Another way is to move small movements in the x direction, and a third technique is to rotate with small angle ϕ. In each case, the readings are combined to get smaller tolerance region. Now, let's discuss each of these techniques in more details.

Translation in the y direction

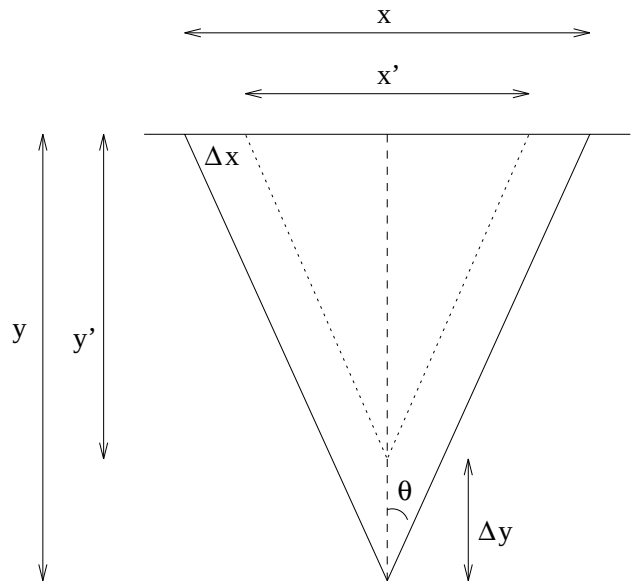

Figure 5: Translation in the y direction.

Moving Δy in the y direction towards the measured point, the tolerance region is decreased by:

$$\Delta x = \Delta y \tan \theta$$

This is shown in Figure 5 where y is the initial distance and x is the initial tolerance region, and y' is the new distance and x' is the new tolerance region. The time to make this movement t_y is equal to:

$$t_y = \Delta y / v_r$$

and total time to make this reduction in the tolerance region is equal to

$$t_y + t_d$$

If the new distance y' is different than $y - \Delta y$, this means that we encountered an edge or a door way. Figure 6 shows different situations in which this may occur.

In this case, the edge can be located with tolerance $2\Delta x$ since the edge ma be at either sides as in cases 1 and 3 of Figures 6 or at both side as in case 2 of the same figure. To determine on which side the edge is located, we can move the robot very small distances in the x direction to left and to the right, and by combining these readings we can determine the edge location within Δx tolerance. Another way to determine the edge location is by rotating the robot clockwise and

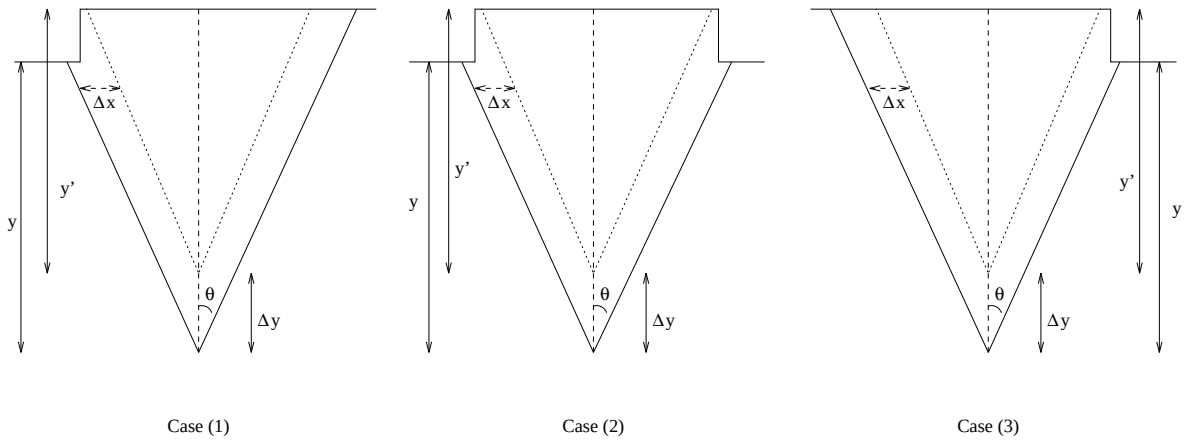

Figure 6: Different situations when moving in the y direction.

counter-clockwise using small angles, and combining the readings as before to determine the edge location.

Translation in the x direction

Moving the robot in the x direction will result in an overlapping region equals to:

$$y \tan\theta - 2\Delta x$$

as shown in Figure 7. The time needed for this movement is equal to:

$$t_x = \Delta x / v_r$$

However, in our experiment, the robot can only move forward and backward. To move in the x direction, we need to rotate the robot 90 degrees, then move forward Δx, and finally rotate back 90 degrees again. Therefore, the time needed to move Δx is:

$$t_x = \pi\omega + \Delta x$$

and by adding the time to decide taking this movement, the total time will be:

$$t_x + t_d$$

If the two readings are the same (i.e., $y' = y$), then we have two possibilities; the measured point is in the overlapping region, or it is outside the overlapping region. Figure 8 shows the two cases. The probability that the two readings correspond to points in the overlapping region is:

$$\frac{y \tan\theta - 2\Delta x}{y \tan\theta + \Delta x}$$

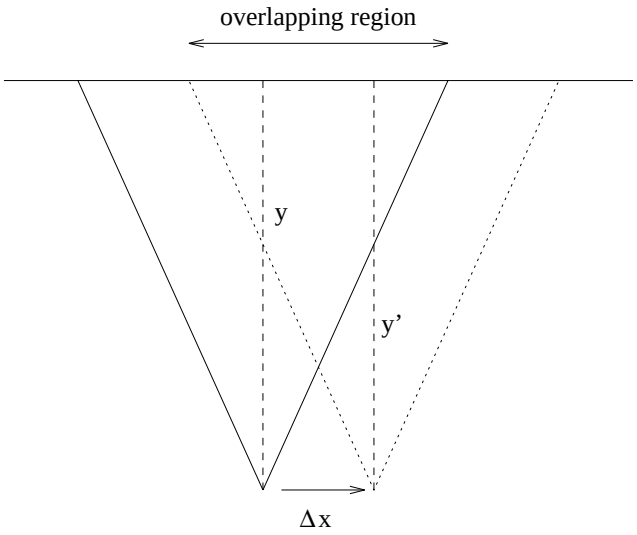

Figure 7: Translation in the x direction.

This formula shows that the probability of the point to be in the overlapping region decreases by increasing Δx

If the two readings are different (i.e., $y' \neq y$), then again, we have two cases as shown in Figure9; $y' > y$ which means that there is an edge within the left Δx region, and $y' < y$ which means that there is an edge within the right Δx region.

Rotating the robot ϕ degrees

Rotation is similar to moving in the x direction. By rotating a small angle ϕ, where

$$-2\theta < \phi < 2\theta$$

and with rotation radius r, there will be an overlapping area as shown in Figure 10. This overlapping area starts at a distance d_o which can be calculated as follows:

$$d_o = ec - gc$$

where

$$ec = r\sin(\phi)\tan(\phi + \frac{\pi}{2} - \theta)$$
$$gc = r - r\cos(\phi) = r(1 - \cos(\phi))$$

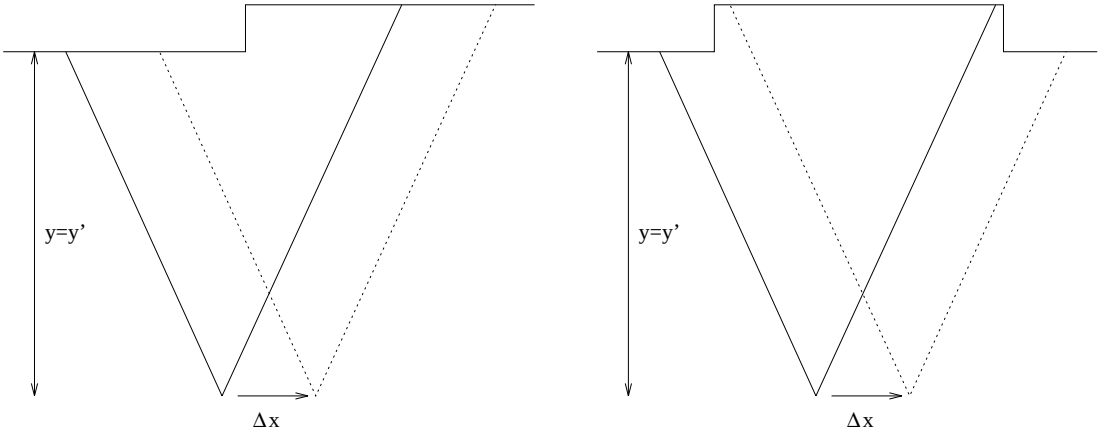

Figure 8: moving in the x direction with $y' = y$

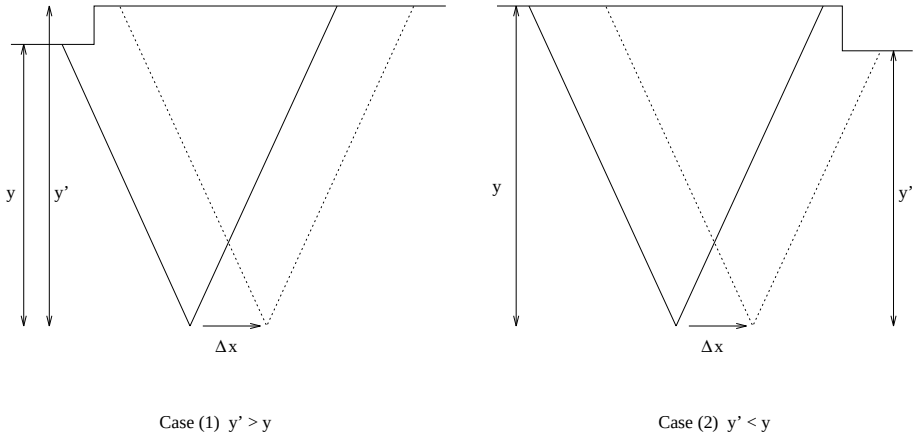

Figure 9: moving in the x direction with $y' \neq y$

finally, we have:

$$d_o = r[\sin(\phi)\tan(\phi + \frac{\pi}{2} - \theta) + \cos(\phi) - 1]$$

and for small values of ϕ, d_o can be approximated by:

$$d_o = \frac{r\phi}{\tan(\theta)}$$

Therefore, if the sensor reading is y, we should rotate the robot such that $y > d_o$, otherwise, the reading will be outside the overlapping region. In other words, the rotation angle ϕ is limited by the sensor reading as follow:

$$\phi < \frac{y\tan(\theta)}{r}$$

Using case analysis similar to what we did for moving the robot in the x direction, with substituting Δx with $r\phi$, we can get new tolerance regions with probabilities associated to them in the same way we did before for the translation in the x direction.

The time needed to rotate ϕ degrees t_{rot} is equal to $\omega\phi$ and adding the decision time t_d we get the total time.

4.2 Using Multiple Sensors

This case is exactly the same as rotating the robot, except for the fact that the angle ϕ is fixed. In case where the sensors are arranges in a circle and distributed on equal spacing angles, ϕ depends on the number of sensors used. For example, if we have 24 sensors, then $\phi = \frac{2\pi}{24}$. To have an overlapping region, ϕ should be less than 2θ. Also, to consider this overlapping region, the sensor readings y for both sensors should be grater than d_o as discussed before.

Again, the case analysis that we did for translation in the x direction can be used here by replacing Δx with $r\phi$. This way we can get smaller tolerance areas with certain probabilities. The probability that the reading is in the overlapping area depends on the value of the readings and on the angles ϕ and θ.

5 Experiments and Simulation Results

A simulator called *XSim* has been developed to examine the applicability of the proposed control scheme. This simulator is based on a mobile robot called "LABMATE" designed by Transitions Research Corporation [20]. This simulator displays the robot on the screen and accepts actual LABMATE commands like *go*, *turn*, *read-sonars*, etc. In this environment, moving from the

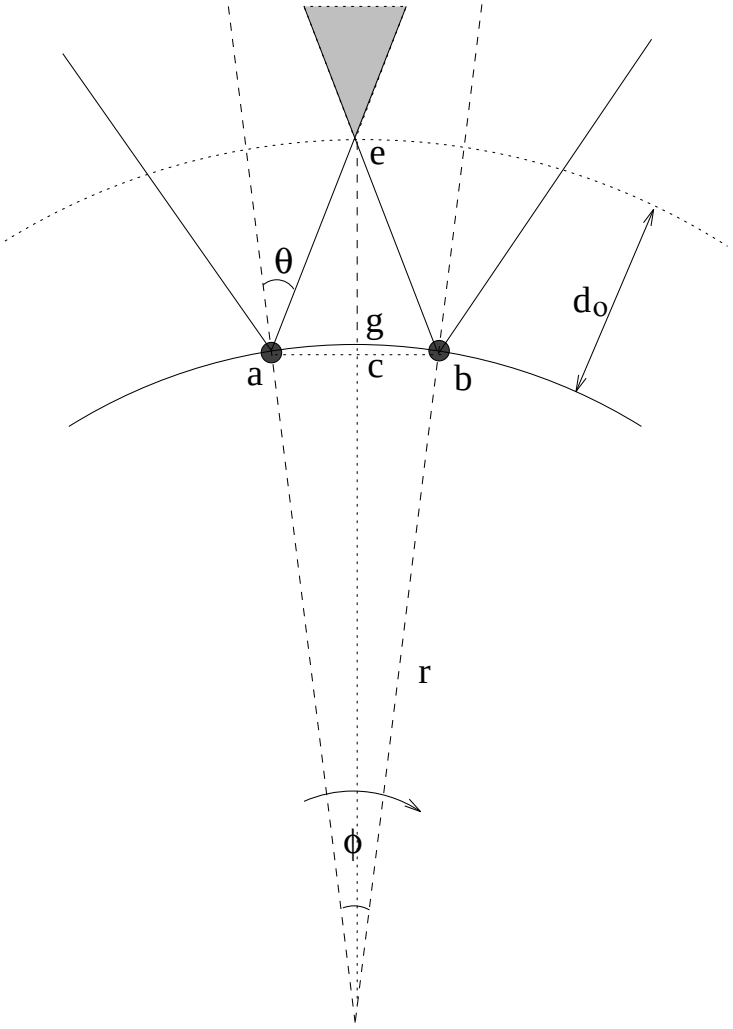

Figure 10: Rotating the robot ϕ degrees.

simulation to the real robot is simply a matter of compiling the driver program with the LABMATE library rather than the simulation library.

The LABMATE was used for several experiments at the Department of Computer Science, University of Utah. It also entered the 1994 AAAI Robot Competition [18]. For that purpose, the LABMATE was equipped with 24 sonar sensors, eight infrared sensors, a camera and a speaker. [1] Figure 11 shows the LABMATE with its equipment, and Figure 12 shows the graphical simulator for the LABMATE.

In all previous experiments, the LABMATE was controlled using a conventional control strategy in which there is a central process (the controller) that does everything. This controller receives raw data from the "dumb" sensors, interprets the data, plans for the next move based on these readings and on the global goal it has to achieve, Tries to avoid obstacles, and finally issues the required commands. Beside that, the central controller may also produce an output for monitoring purposes. The following are some drawbacks for this scheme:

- The central controller has to know the type and location of each sensor.
- It also needs to know the data format for each sensor type.
- It may take long time to issue the required command. This time depends on the interpretation procedure for the data received from each sensor, and on the time to select the next command.
- Adding or removing any sensor requires modifying the central controller.

5.1 Modeling the System

The sensors in the old scheme are used only as dumb sensors, while in the proposed scheme, sensors are used in three different levels. They are used as dumb sensors to provide feedback information for a general navigator. They are also used as intelligent sensors providing information to a monitoring process (e.g., a speaker as an output device.) Finally they are used as commanding sensors (clients) for collision avoidance. The emergency exits are hardware bumpers that command the robot to stop if it touch any object. There is also a general controller for navigation and map construction. The commands that can be issued are:

- **GO-FRWD** d**:** move forward distance d inches, where d is a non-negative real number. When $d = 0$, the robot will keep moving forward until other command is issued.
- **GO-BKWD** d**:** move backward distance d inches, where d is a non-negative real number. When $d = 0$, the robot will keep moving backward until other command is issued.

[1] The LABMATE preparations, the sensory equipments, and the software and hardware controllers were done by L. Schenkat and L. Veigel at the Department of Computer Science, University of Utah.

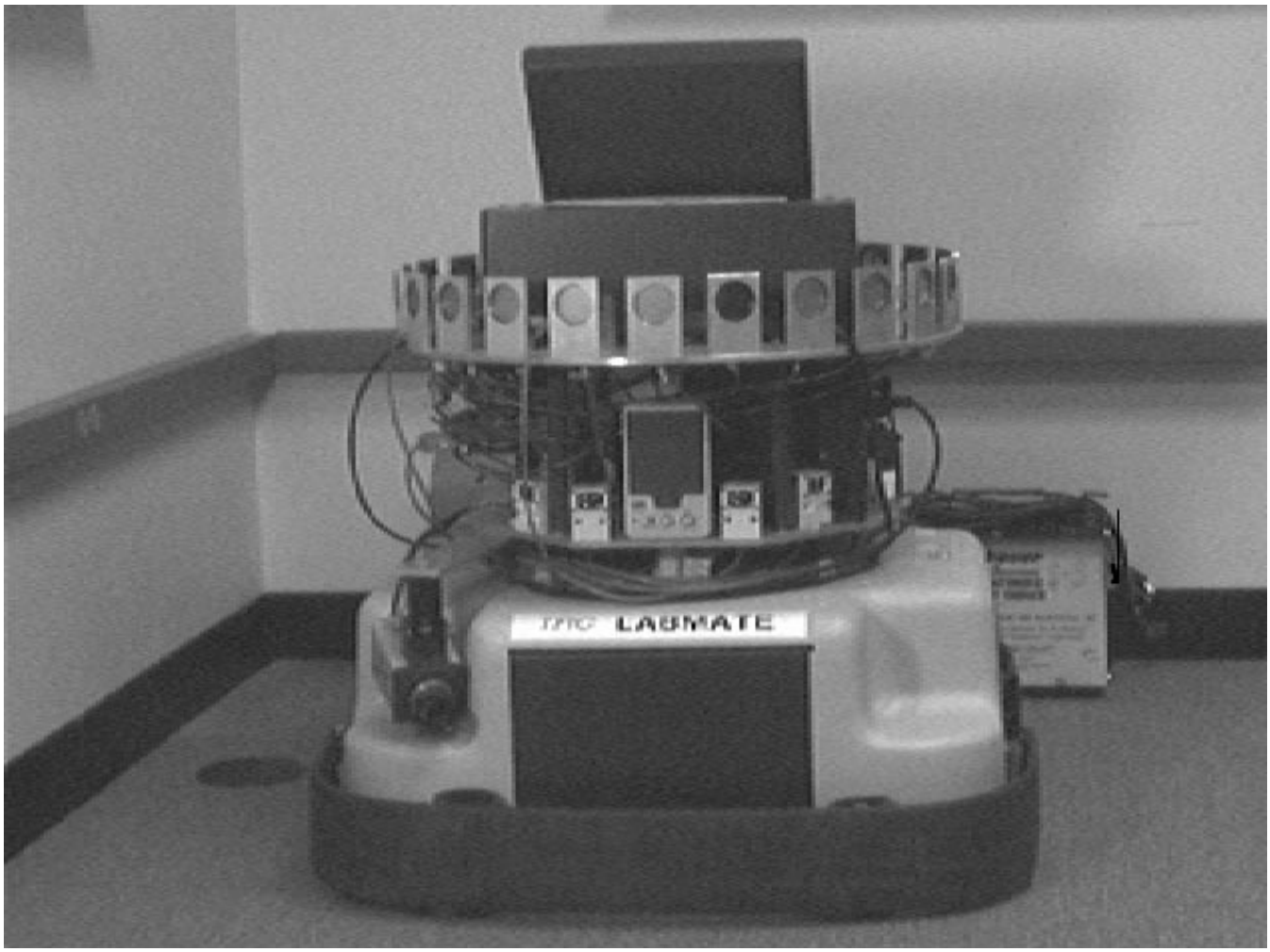

Figure 11: The LABMATE robot with its equipments.

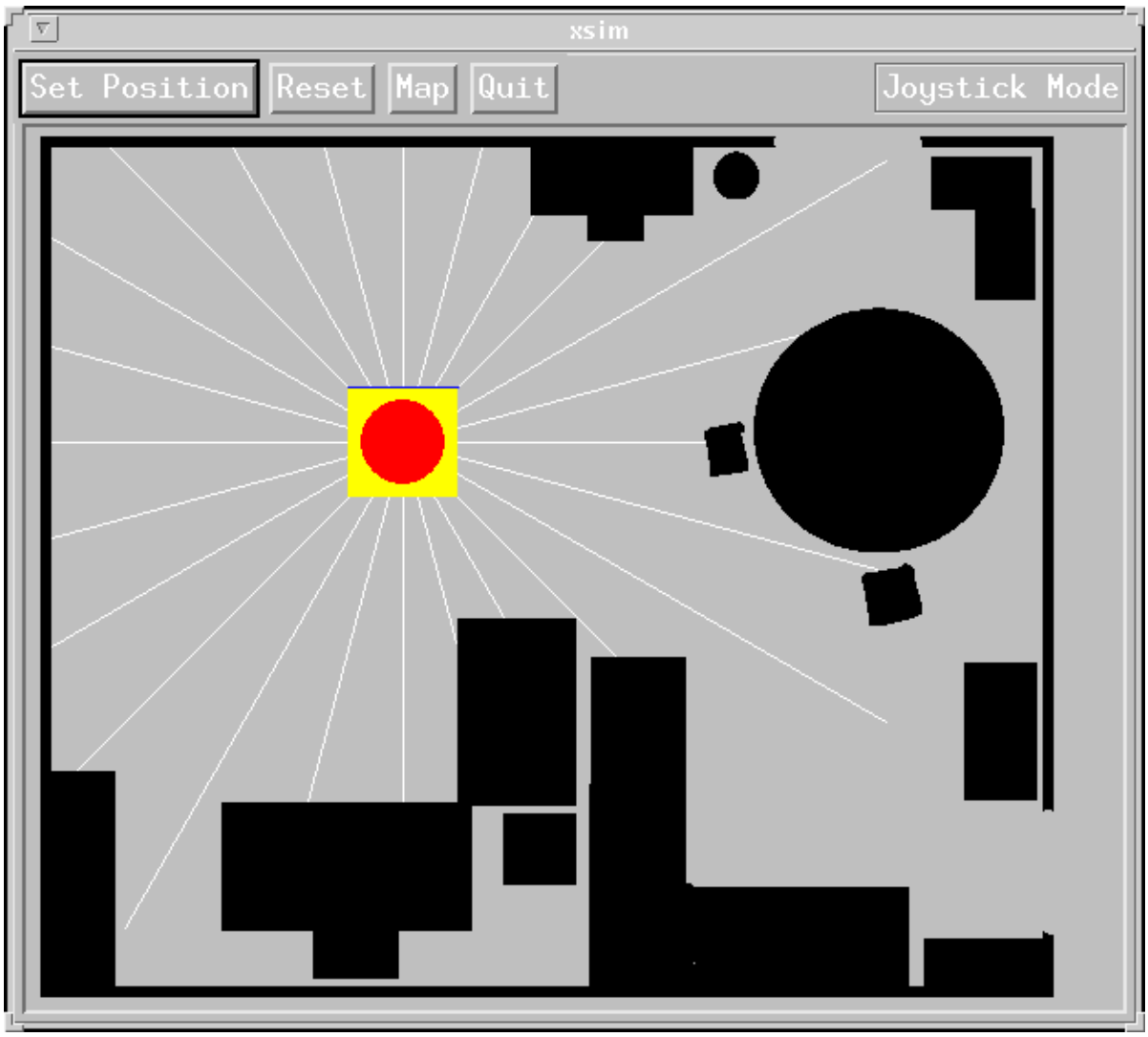

Figure 12: A graphical simulator for the LABMATE.

- **TURN-RIGHT θ:** turn right θ degrees, where θ is a positive real number.
- **TURN-LEFT θ:** turn left θ degrees, where θ is a positive real number.
- **STOP:** stop moving (or turning).
- **RESET:** restart operation after a fault.
- **READ-SONAR:** read the sonar data.
- **GET-POSITION:** get the current position of the robot.

The system can be in any of the following states:

- **IDLE:** the robot is not moving.
- **FORWARD:** the robot is moving forward.
- **BACKWARD:** the robot is moving backward.
- **RIGHT:** the robot is turning right.
- **LEFT:** the robot is turning left.
- **FAULT:** the robot hit an obstacle.

Figure 13 shows a state diagram for the system. This figure shows that the robot has to go to the idle state when the command is changed. For example, if the command *GO-FORWARD* is issued, the system will go to the *FORWARD* state and will remain there as long as the following commands are *GO-FORWARD*. Once the next command is different, the system will go to the *IDLE* state first, then it will go to the state corresponding to the current command. This is analogous to what happens in controlling the LABMATE. The LABMATE has to stop first before changing direction. Notice that the command *READ-SONAR* is not present in that figure since it can be executed at any state.

5.2 Commanding Sensors and Reaction Control

To simplify our model, the 24 sonar sensors are divided into four logical sensors as shown in Figure 14.

1. **LS-FRWD** consists of the front 6 sensors.

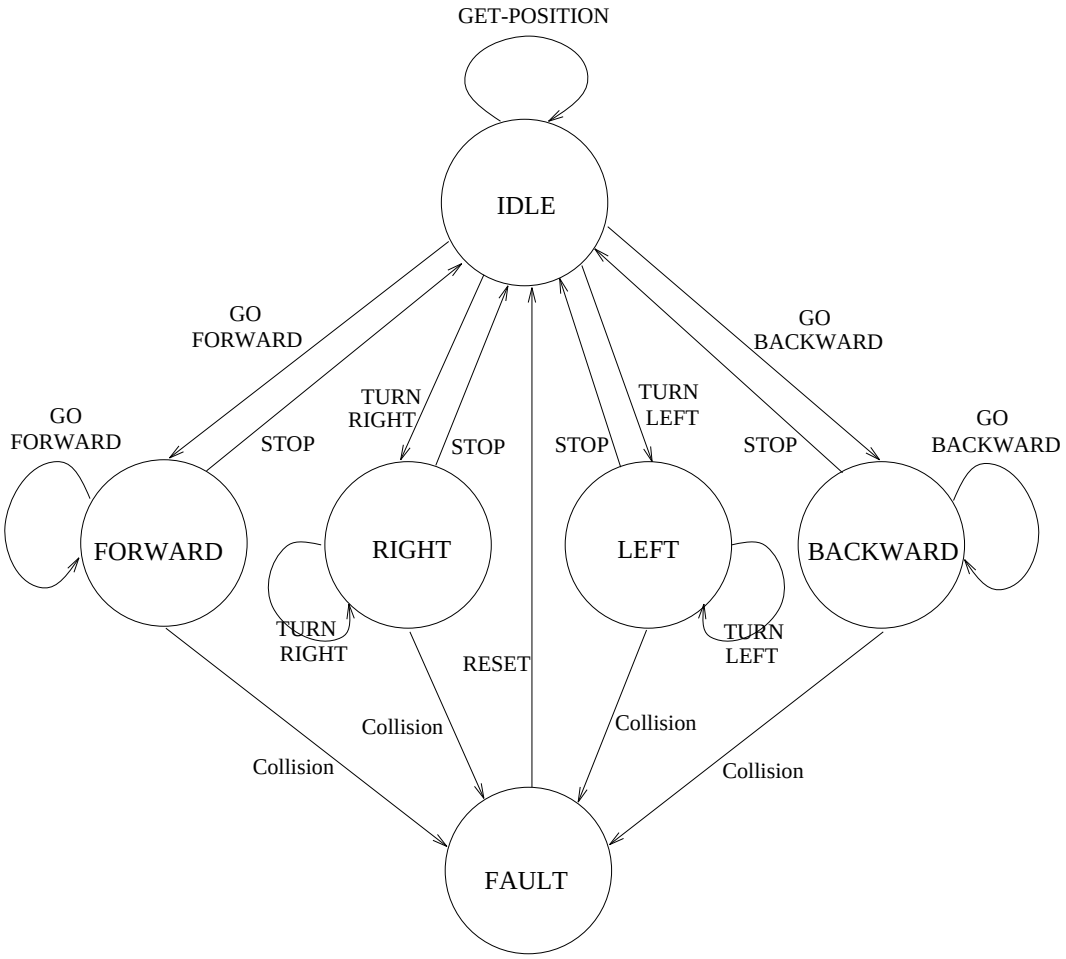

Figure 13: The relation between the system states and the commands.

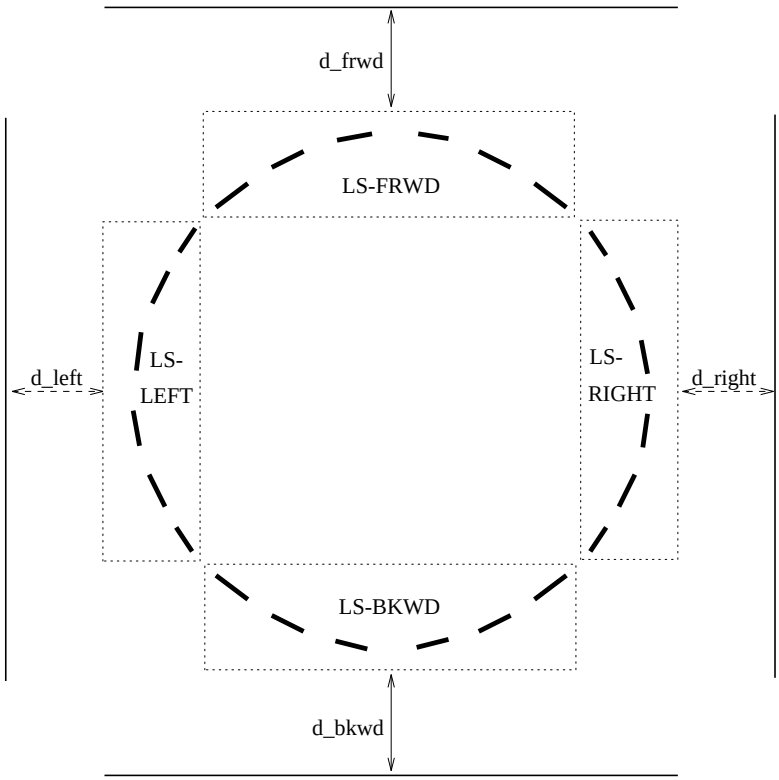

Figure 14: Dividing the sonar sensors into four logical sensors.

2. **LS-BKWD** consists of the rear 6 sensors.

3. **LS-RIGHT** consists of the right 6 sensors.

4. **LS-LEFT** consists of the left 6 sensors.

These logical sensors communicate with each other to decide the command to be issued. This makes the job of the multiplexer easier, since it will deal with the four logical sensors as one client. The goal of the reactive control in this experiment is two fold:

1. Avoid obstacles.

2. Keep the robot in the middle of hallways, specially when moving through narrow corridors.

We will define two abstract values: *close* (c) and *far* (f). These two values represent the distance between the robot and the closest object at any of the four sides. The range for c and f are usually user defined values. The command to be issued as a reaction control depends on the current state of the system and the distance value at each side. There are several ways to define a command function $\{$ to achieve the required goal. The assumption here is that there is always enough space for the robot to rotate left of right, therefore there is no need to define any reaction control when the robot is rotating. One such function is shown in Table 1.

In this table, *TURN-L/R* means the command can be either *TURN-LEFT* or *TURN-RIGHT*, and a dash "—" means no command is issued. Notice that, in case of d_left and d_right have different values, the values for d_frwd and d_bkwd are not important. This is because we need to balance the distance to the left and to the right of the robot, and if, for example, the distance in front (d_frwd) is c, and the robot state is *FORWARD*, then moving to the left (or to the right) will serve both; avoiding the object in front, and balancing the distance on both sides. In the first case of the table, when the distance is c in all sides, the robot will not be able to move anywhere, and the sensor readings will not change. This will result in a deadlock which requires external help by moving at least one of the obstacles for the robot to be able to move. Figure 15 shows graphically the different cases when the system state is *FORWARD*, and Figure 16 shows the same cases when the system state is *BACKWARD*.

5.3 The Priority Scheme

In this system, there are several clients for the server. Beside these clients, there are two emergency exits represented by two bumpers, one on the front and one on the back. As mentioned before, emergency exits do not compete for the server, rather it sends its commands directly to the low-level controller.

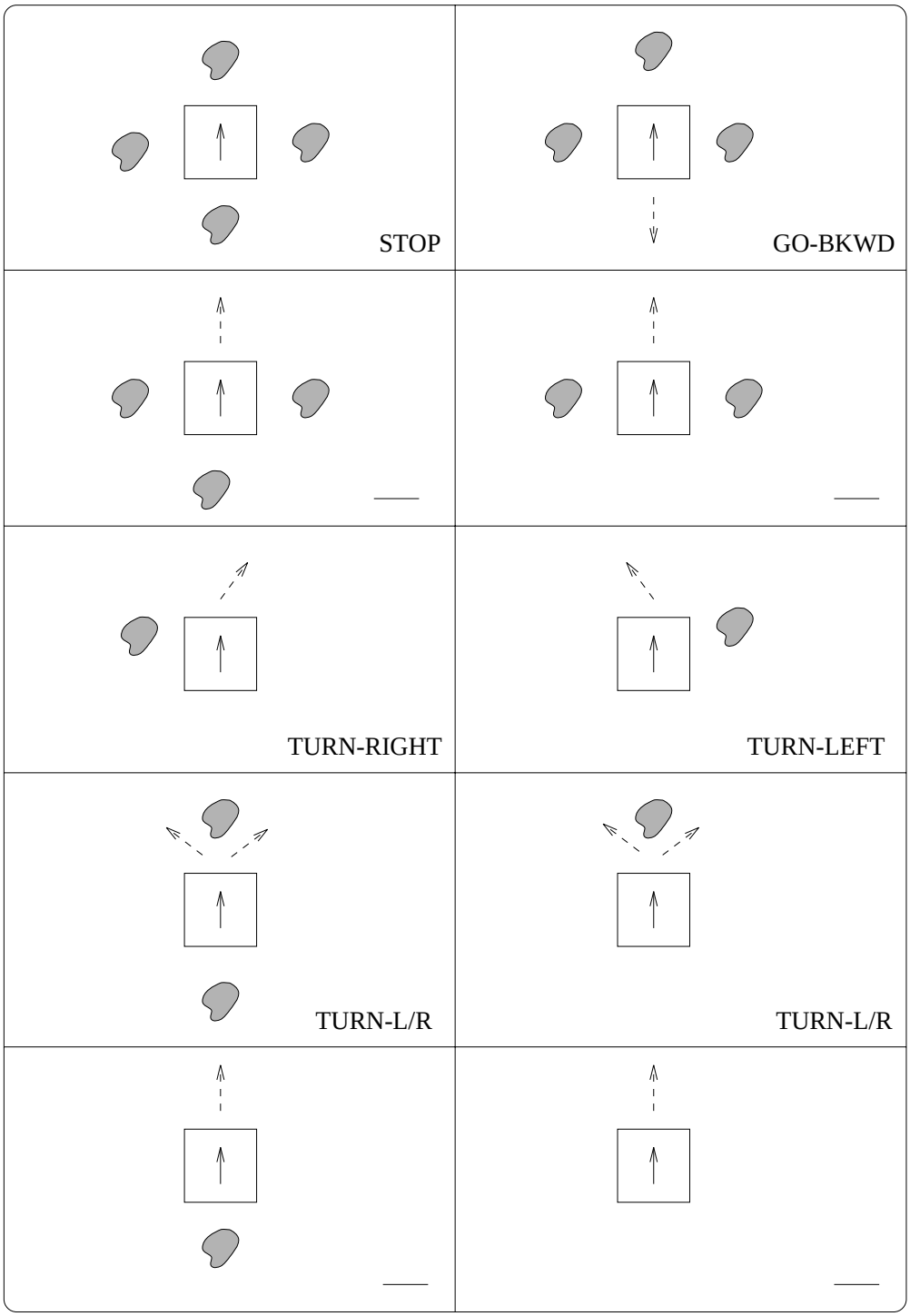

Figure 15: The reaction control when the system state = *FORWARD*.

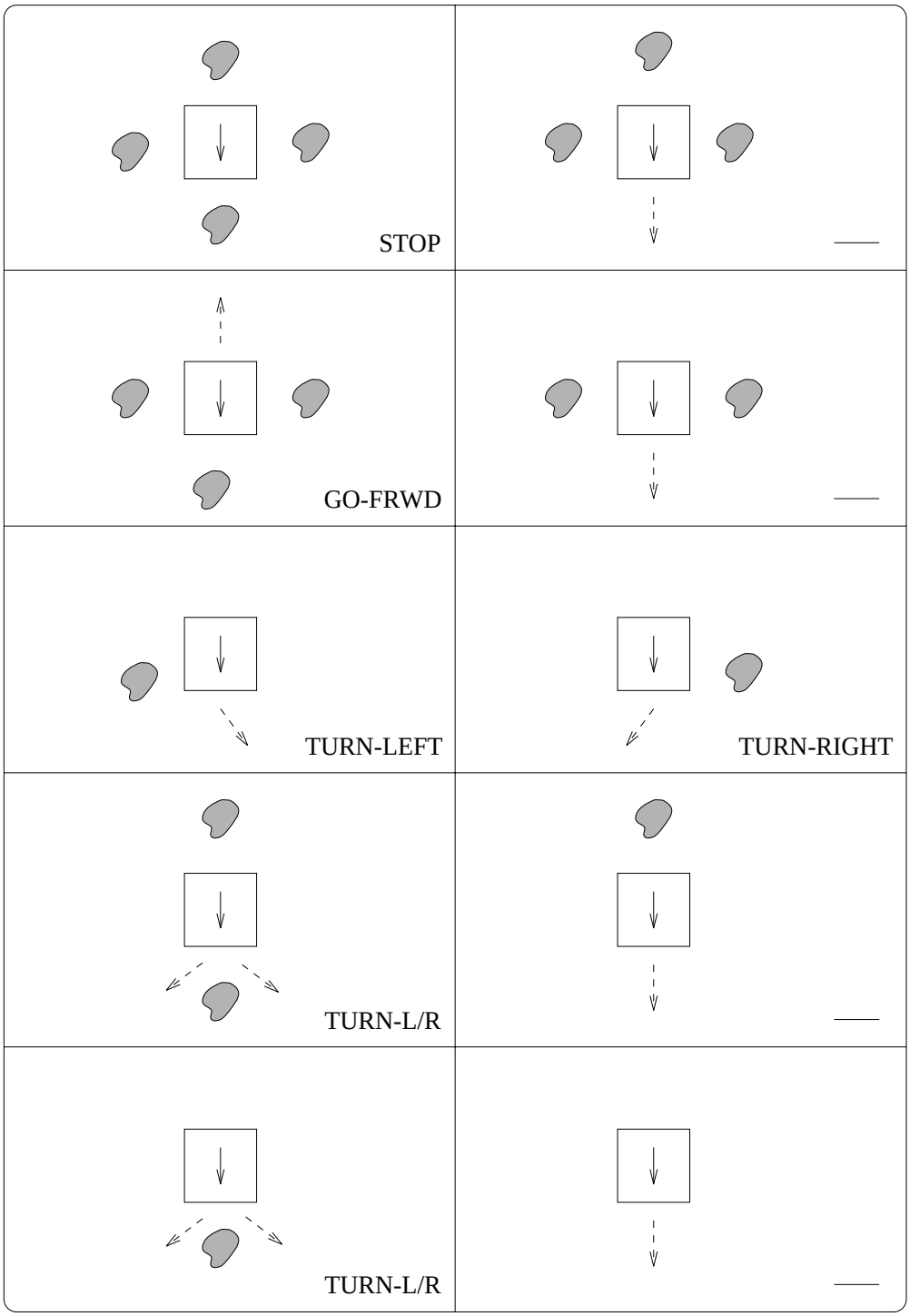

Figure 16: The reaction control when the system state = *BACKWARD*.

The priority scheme in our application is set by each client as a number from 1 to 10, with 1 as the highest priority. Normally, 1 is reserved for the collision avoidance client. The server checks for the priority associated with each command, and executes the command with the highest priority while notifying the "losers" which command was executed. If two commands with the same priority arrive at the same time, the server arbitrarily selects one of them and ignores the other.

Commands that were not selected are cleared since the state of the robot has been changed after executing the command with the highest priority.

5.4 Simulation Results

Several experiments were performed on the simulator to check the applicability and validity of the proposed control scheme, and the results were very encouraging. The following is a description of four of these experiments along with the output of the simulation showing the portion of the commands that were selected and the trajectory of the robot during each experiment.

Experiment (1)

This was the first experiment performed to demonstrated the applicability of this control scheme. In this experiment, two clients were running simultaneously; the collision avoidance client, and a simple navigator which always sends the command GO-FRWD. The collision avoidance has priority 1, which is the highest priority, and the navigation client has priority 9. The following shows part of the output printed during this experiment which shows the commands that has been executed by the server and some other information about the server activities.

```
Collision Avoidance: client #1.
Simple Navigation: client #2.
Server Starts as process #0.

* Accepted RESET from 1 *
- Rejected RESET from 1 *
* Accepted GO-FRWD from 2 *
* Accepted GO-FRWD from 2 *
* Accepted GO-FRWD from 2 *
* Accepted GO-FRWD from 2 *
* Accepted GO-FRWD from 2 *
```

```
* Accepted GO-FRWD from 2 *
* Accepted TURN-LEFT from 1 *
- Rejected GO-FRWD from 2 -
* Accepted TURN-LEFT from 1 *
- Rejected GO-FRWD from 2 -
* Accepted GO-FRWD from 2 *
* Accepted GO-FRWD from 2 *

         . . .
```

Two indoor configurations where used for these experiments; one representing a lab with tables and chairs, while the other represents long halls with doors and some obstacles. Figure 17 shows the trajectory of the robot in the lab environment, and Figure 18 shows the trajectory of the robot under the same experiment in the hallway environment.

Experiment (2)

In the second experiment, we added another goal-directed client which tries to move the robot to a certain goal location. This client has priority 5 which is higher than the simple navigator process. This new client sends commands to the server to update the direction of the robot such that it moves towards the goal location. In this experiment, the initial and the final points were chosen such that there are some obstacles between them. Figure 19 shows the robot trajectory for this experiment from the initial location to the goal location. Notice that at several points, the collision avoidance client took over and moved the robot away from the obstacles, then the new client updates the direction towards the goal point.

Experiment (3)

In the third experiment, we replaced the goal-directed client with a door-finding client. This new client tries to find open doors and direct the robot to go through these doors. Finding doors using sonar sensor is very hard and problematic, and there is a lot of research in this area. For this experiment we used a very crude algorithm and a simple hallway structures just to demonstrate the capabilities of the proposed control scheme. Figure 20 shows the robot trajectory while moving in a hallway environment with two open doors at different places.

Experiment (4)

In this last experiment, We demonstrate the use of the tolerance measures discussed in Section 4. this experiment also illustrates the use of the logical sensors concept to implement high-level requests which incorporate tolerance measures and time calculations.

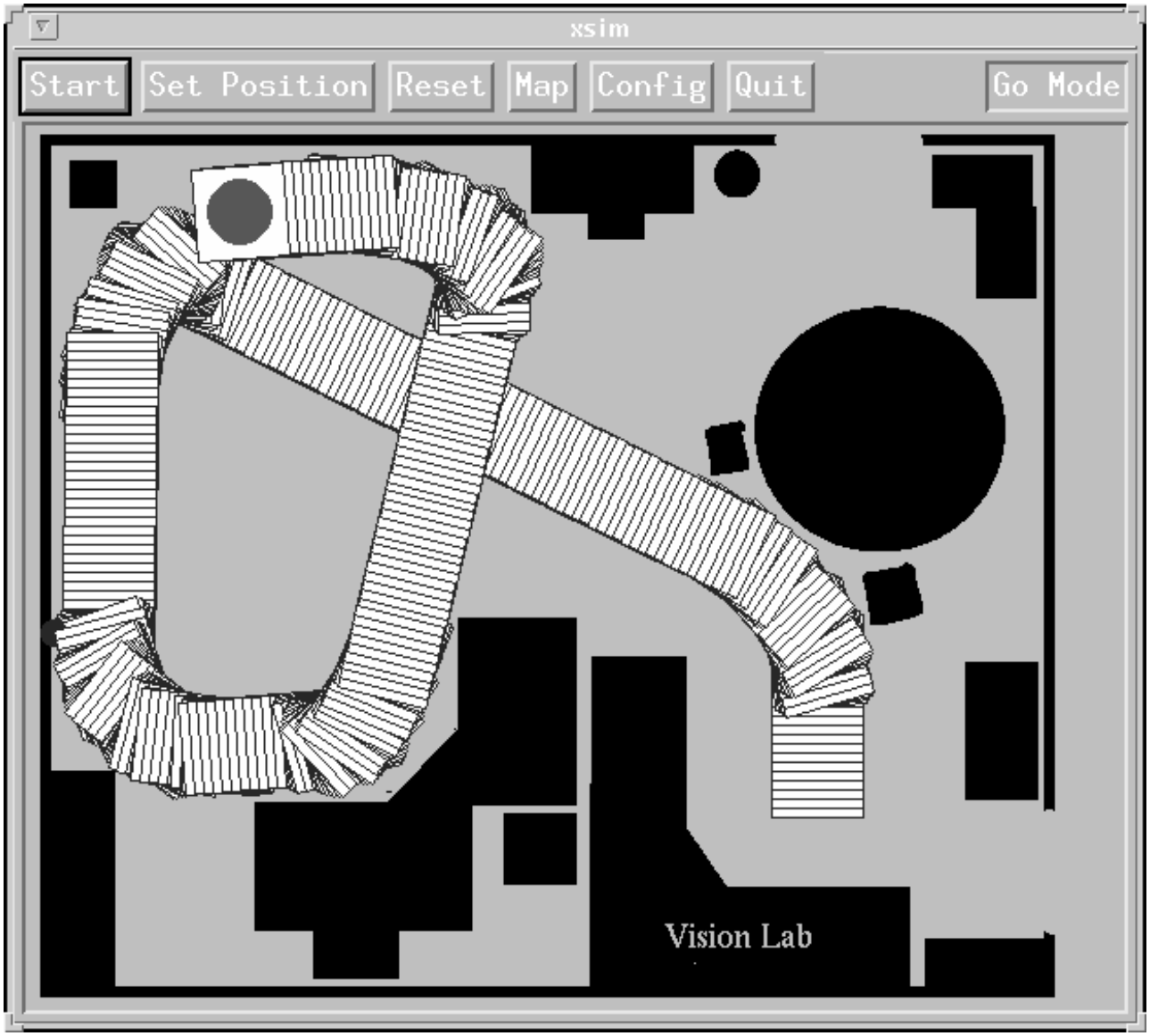

Figure 17: The trajectory of the robot in the lab environment.

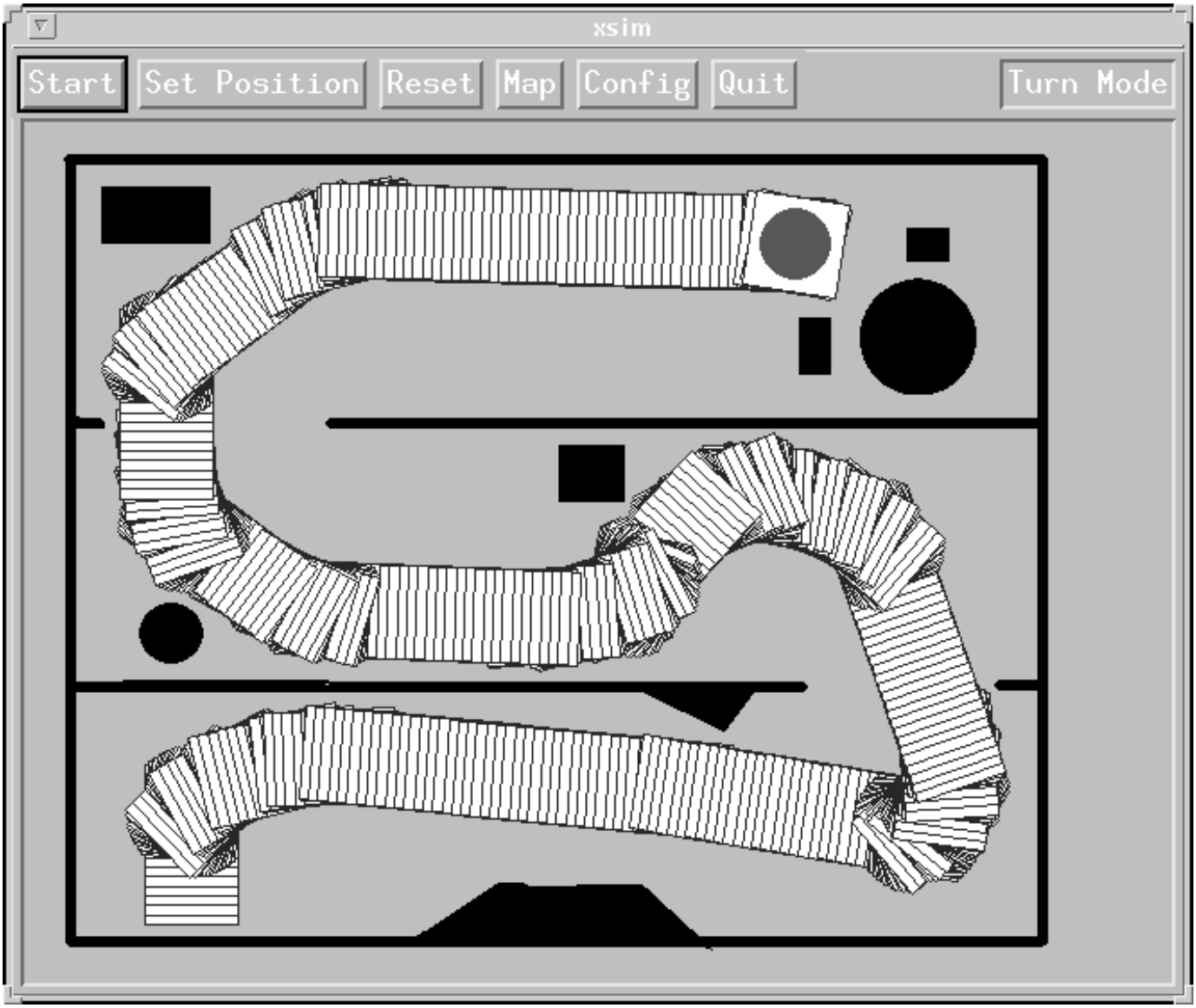

Figure 18: The trajectory of the robot in the hallway environment.

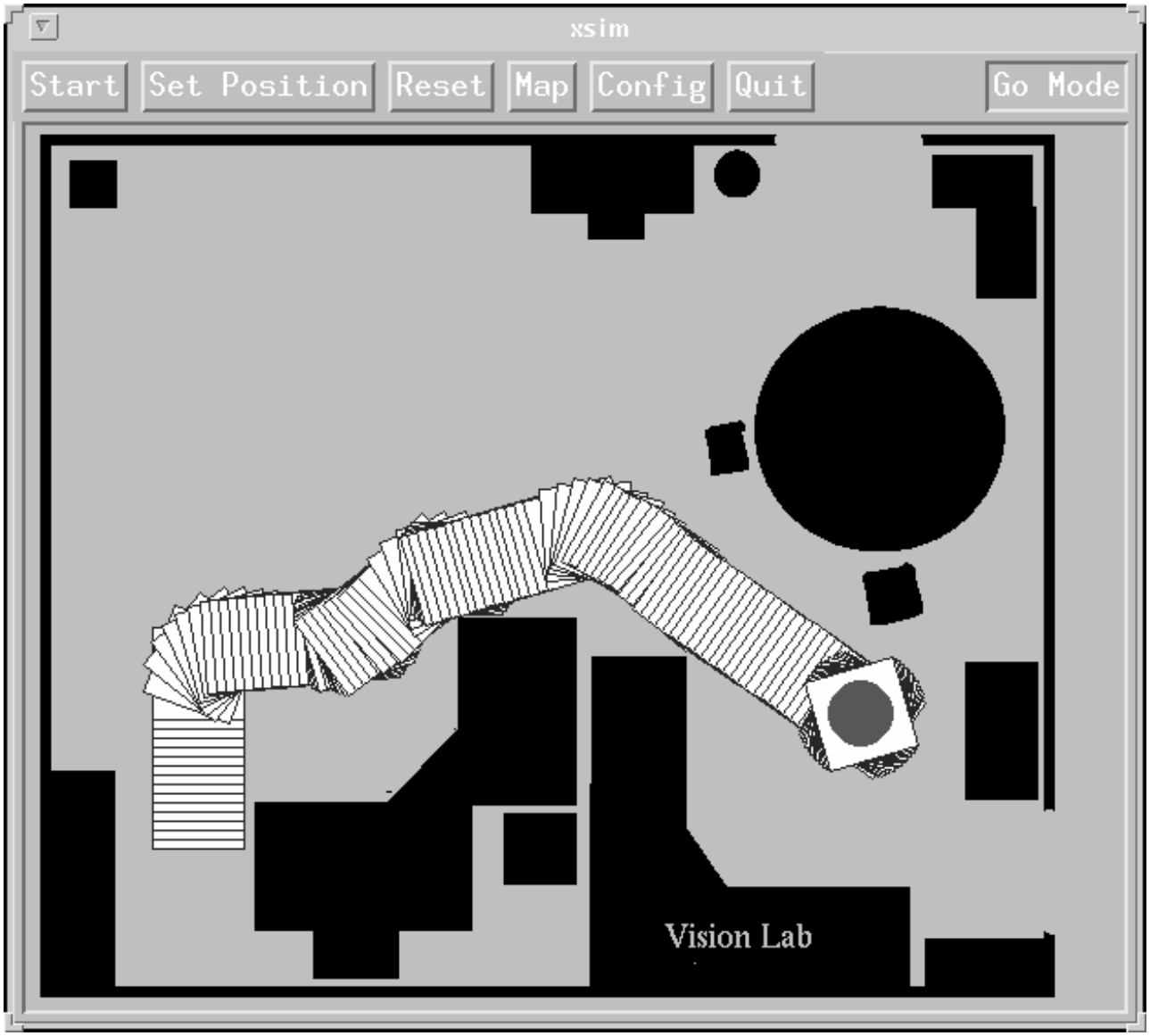

Figure 19: The trajectory of the robot from the initial to the goal point.

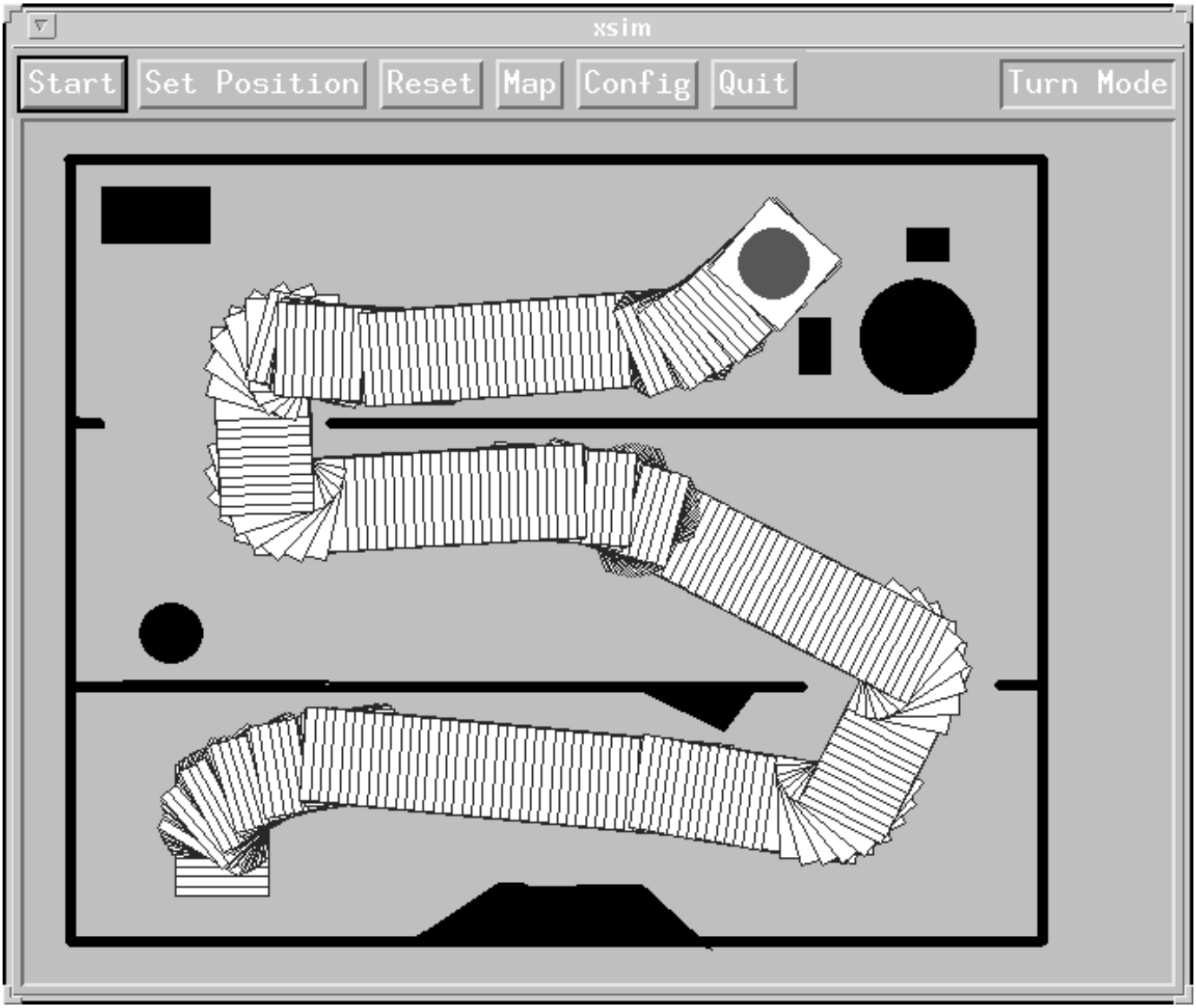

Figure 20: The trajectory of the robot while moving through open doors.

The request which was implemented for this experiment is *measure* which has the following syntax:

$$measure(tolerance, time, preference)$$

where $tolerance$ is the required tolerance with 0 meaning get the best tolerance, and -1 means tolerance is not important. $time$ is the required response time, and again 0 means as fast as possible, and -1 means time is not important. When both, $time$ and $tolerance$ are specified, the logical sensor may not be able to satisfy both criteria, and this is when $preference$ is used to specify which criteria should be preferred. This request returns the resulting tolerance and the time consumed into the same parameters that were sent.

The following is the output of a program which uses this request to measure a point in front of the robot. First it sends a request to get the measure as fast as possible ignoring the tolerance.

```
Fast response required ...
!!! minimum time ...
!!!  current reading is 2071 mm, with tolerance 402.4 in time 0.9 sec.

Result: distance = 2071 mm, tolerance = 402.4, and time = 0.9 sec.
```

Second, the program sends a request to get the best accuracy (minimum tolerance), and the time is irrelevant.

```
Best tolerance required ...
!!! minimize the tolerance ...
!!!  current reading is 1536 mm, with tolerance 298.4 mm. in time 0.6 msec.
!!!  current reading is 1412 mm, with tolerance 274.3 mm. in time 2.5 msec.
!!!  current reading is 1383 mm, with tolerance 268.7 mm. in time 3.4 msec.
!!!  current reading is 1350 mm, with tolerance 262.3 mm. in time 4.4 msec.
!!!  current reading is 1291 mm, with tolerance 250.8 mm. in time 5.6 msec.
!!!  current reading is 1259 mm, with tolerance 244.6 mm. in time 6.5 msec.
!!!  current reading is 1215 mm, with tolerance 236.0 mm. in time 7.6 msec.
!!!  current reading is 1168 mm, with tolerance 226.9 mm. in time 8.7 msec.
!!!  current reading is 1139 mm, with tolerance 221.3 mm. in time 9.6 msec.
!!!  current reading is 1091 mm, with tolerance 212.0 mm. in time 10.4 msec.
!!!  current reading is 1062 mm, with tolerance 206.3 mm. in time 11.0 msec.
!!!  current reading is 1015 mm, with tolerance 197.2 mm. in time 11.8 msec.
!!!  current reading is 971 mm, with tolerance 188.6 mm. in time 12.5 msec.
!!!  current reading is 938 mm, with tolerance 182.2 mm. in time 13.2 msec.
!!!  current reading is 894 mm, with tolerance 173.7 mm. in time 13.9 msec.
!!!  current reading is 847 mm, with tolerance 164.6 mm. in time 14.7 msec.
!!!  current reading is 818 mm, with tolerance 158.9 mm. in time 15.3 msec.
!!!  current reading is 756 mm, with tolerance 146.9 mm. in time 16.3 msec.
!!!  current reading is 724 mm, with tolerance 140.7 mm. in time 16.9 msec.
!!!  current reading is 694 mm, with tolerance 134.8 mm. in time 17.5 msec.
!!!  current reading is 633 mm, with tolerance 123.0 mm. in time 18.4 msec.
!!!  current reading is 603 mm, with tolerance 117.2 mm. in time 19.0 msec.

  distance = 1536 mm,  tolerance = 117.2, and time = 19.0 msec.
```

Finally, the program specifies both time and tolerance to be met, preferring the time.

```
Tolerance required = 150.0, time required = 6.0 msec.
!!! both criteria are specified ...
!!!  current reading is 2190 mm, with tolerance 425.5 mm. and time 0.9 msec.
!!!  current reading is 2101 mm, with tolerance 408.2 mm. and time 2.7 msec.
!!!  current reading is 2068 mm, with tolerance 401.8 mm. and time 4.1 msec.
!!!  current reading is 2026 mm, with tolerance 393.6 mm. and time 5.6 msec.

Result: distance = 2190 mm, tolerance = 393.6, and time = 5.6 msec.
```

Figure 21 shows the movement of the robot while taking these measurements. The first request did not cause any movement since it required minimum time. The second request cased the robot to move forward to minimize the tolerance region. During this movement, the speed of the robot decreases to get better accuracy. Finally, the last request also caused the robot to move forward, but it stopped before reaching the required tolerance since the time was preferred.

In this experiment we used only the translation in the y direction to minimize the tolerance. However, the other two approaches mentioned in Section 4 could be used to get better results with probability measures as well.

6 Conclusion and Future Work

In this paper, a distributed sensor-based control scheme was proposed. In this scheme, each sensor can be viewed with three different levels of abstraction; *dumb sensors* which provide raw data, *intelligent sensors* which provides high level information in a form of events, and finally, *commanding sensors* which can issue commands representing a reaction behavior for the system. Commands can be issued by different processes called *clients*. Each client may issue commands at any time, and a multiplexer (the server) selects the command to be executed. A priority scheme has to be defined as a bases for selection. Tolerance measures for sonar sensors were proposed and different strategies to increase position accuracy were investigated. An example for applying this control scheme to a mobile robot was described along with the simulation results. We believe that this control scheme provides more flexible and robust control systems, and allows more modular design for the whole control system. It also provides fast response for reaction behavior which is an essential requirement in real-time systems.

The next step to this work is to implement a distributed controller for the "real" LABMATE using the proposed control scheme. A more detailed decision function for the logical sensors may be defined and the communication protocols among the sonar sensors needs to be explicitly defined. Higher level functions for increasing the accuracy of the measured point locations is to be defined utilizing the different approaches discussed in the paper. Finally, the data noise should

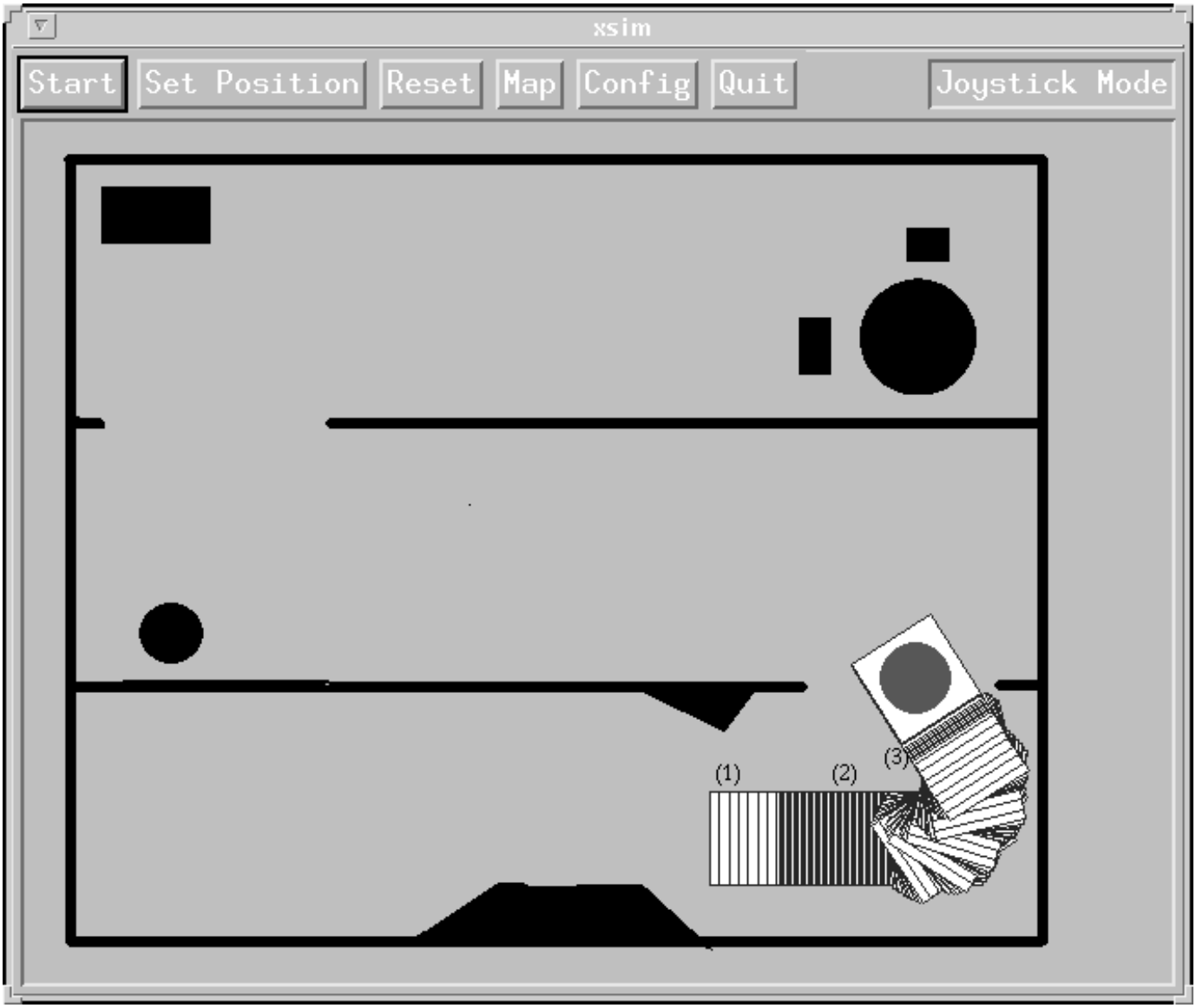

Figure 21: The trajectory of the robot while performing the requests.

be considered and modeled, and some filtering technique such as Kalmann filtering can be used to reduce this noise.

References

[1] In *MPI: a message-passing interface standard.* University of Tennessee, Knoxville., May 1994.

[2] AHLUWALIA, R. S., AND HSU, E. Y. Sensor-based obstruction avoidance technique for a mobile robot. *Journal of Robotic Systems 1*, 4 (Winter 1984), pp. 331–350.

[3] BUDENSKE, J., AND GINI, M. Why is it difficult for a robot to pass through a doorway using altrasonic sensors? In *IEEE Int. Conf. Robotics and Automation* (May 1994), pp. 3124–3129.

[4] DEKHIL, M., GOPALAKRISHNAN, G., AND HENDERSON, T. C. Modeling and verification of distributed control scheme for mobile robots. Tech. Rep. UUCS-95-004, University of Utah, April 1995.

[5] GOURLEY, C., AND TRIVEDI, M. Sensor-based obstacle avoidance and mapping for fast mobile robots. In *IEEE Int. Conf. Robotics and Automation* (1994).

[6] HAGER, G. D. Task-directed computation of qualitative decisions from sensor data. *IEEE Trans. Robotics and Automation 10*, 4 (August 1994), pp. 415–429.

[7] HENDERSON, T. C., AND SHILCRAT, E. Logical sensor systems. *Journal of Robotic Systems* (Mar. 1984), pp. 169–193.

[8] KLEEMAN, L., AND KUC, R. An optimal sonar array for target localization and classification. In *IEEE Int. Conf. Robotics and Automation* (May 1994), pp. pp. 3130–3135.

[9] KORBA, L. Variable aperture sonar for mobile robots. In *IEEE Int. Conf. Robotics and Automation* (May 1994), pp. 3142–3147.

[10] KOŠECKÁ, J., AND BOGONI, L. Application of discrete event systems for modeling and controlling robotic agents. In *IEEE Int. Conf. Robotics and Automation* (May 1994), pp. 2557–2562.

[11] LEE, C. S. G. *Sensor-based robots: algorithms and architecture*. Springer-Verlag, 1991.

[12] LIN, C. C., AND TUMMALA, R. L. Adaptive sensor integration for mobile robot navigation. In *IEEE International Conference on Multisensor Fusion and Integration* (Oct. 1994).

[13] LUO, R. C., AND KAY, M. G. *Multisensor integration and fusion for intelligent machines and systems*. Ablex Publishing Corporation, 1995.

[14] MILLER, W. T. Sensor-based control of robotic manipulators using a general learing algorithm. *IEEE Journal of Robotics and Automation* (Nov. 1987), pp. 157–165.

[15] MUTAMBARA, A. G. O., AND DURRANT-WHYTE, H. F. Modular scalable robot control. In *IEEE International Conference on Multisensor Fusion and Integration* (Oct. 1994).

[16] REMBOLD, U., AND HORMANN, K. *Languages for Sensor-Based Control in Robotics.* Springer-Verlag, 1987.

[17] SABATINI, A. M., AND BENEDETTO, O. D. Towards a robust methodology for mobile robot localization using sonar. In *IEEE Int. Conf. Robotics and Automation* (May 1994), pp. 3136–3141.

[18] SCHENKAT, L., VEIGEL, L., AND HENDERSON, T. C. Egor: Design, development, implementation – an entry in the 1994 AAAI robot competition. Tech. Rep. UUCS-94-034, University of Utah, Dec. 1994.

[19] SHILCRAT, E. D. Logical sensor systems. Master's thesis, University of Utah, August 1984.

[20] TRC TRANSITION RESEARCH CORPORATION. *LABMATE user manual, version 5.21L - f.*, 1991.

[21] YAKOVLEFF, A., NGUYEN, X. T., BOUZERDOUM, A., MOINI, A., BOGNER, R. E., AND ESHRAGHIAN, K. Dual-purpose interpretation of sensory information. In *IEEE Int. Conf. Robotics and Automation* (1994).

d_right	d_left	d_frwd	d_bkwd	*FORWARD*	*BACKWARD*
c	c	c	c	*STOP*	*STOP*
c	c	c	f	*GO-BKWD*	—
c	c	f	c	—	*GO-FRWD*
c	c	f	f	—	—
c	f	c	c	*TURN-RIGHT*	*TURN-LEFT*
c	f	c	f	*TURN-RIGHT*	*TURN-LEFT*
c	f	f	c	*TURN-RIGHT*	*TURN-LEFT*
c	f	f	f	*TURN-RIGHT*	*TURN-LEFT*
f	c	c	c	*TURN-LEFT*	*TURN-RIGHT*
f	c	c	f	*TURN-LEFT*	*TURN-RIGHT*
f	c	f	c	*TURN-LEFT*	*TURN-RIGHT*
f	c	f	f	*TURN-LEFT*	*TURN-RIGHT*
f	f	c	c	*TURN-L/R*	*TURN-L/R*
f	f	c	f	*TURN-L/R*	—
f	f	f	c	—	*TURN-L/R*
f	f	f	f	—	—

Table 1: An example of a decision function for reaction control.