2009

Solving the Maze: Robot Localization Using the Monte Carlo Localization Algorithm and Shape Context

Stiliyana Stamenova
Macalester College

Honors Project

Macalester College

Spring 2009

Title: Solving the Maze: Robot Localization Using the Monte Carlo Localization Algorithm and Shape Context

Author: Stiliyana Stamenova

PERMISSION TO DEPOSIT HONORS PROJECTS

Please read this document carefully before signing. If you have questions about any of these permissions, please contact Janet Sietmann (x6545) in the Library.

Title of Honors Project: Solving the Maze: Robot Localization Using the Monte Carlo Localization Algorithm and Shape Context

Author's Name: (Last name, first name) Stanescue, Stiliyana

The library provides access to your Honors Project in several ways:

- The library makes each Honors Project available to members of the Macalester College community and the general public on site during regular library hours.
- Using the latest technology, we make preservation copies of each Honors Project in both digital and microfilm formats.
- Every Honors Project is cataloged and recorded in CLICnet (library consortium OPAC) and in OCLC, the largest bibliographic database in the world.
- **To better serve the scholarly community,** a digital copy of your Honors Project will be made available via the Digital Commons @ Macalester (digitalcommons.macalester.edu.

The DigitalCommons@Macalester is our web based, open access compliant institutional repository for digital content produced by Macalester faculty, students, and staff. It is a **permanent** archive. By placing your projects in the Digital Commons, all materials are searchable via Google Scholar and other search engines. Materials that are located in the Digital Commons are freely accessible to the world; however, your copyright protects against unauthorized use of the content. Although you have certain rights and privileges with your copyright, there are also responsibilities. Please review the following statements and identify that you have read them by signing below. Some departments may choose to protect the work of their Honors students because of continuing research. In these cases the project is still posted on the repository, but content can only be accessed by individuals who are located on campus.

The original signed copy of this form will be bound with the print copy of the Honors Project. The microfilm copy will also include a copy of this form. Notice that this form exists will be included in the Digital Commons version.

I have read the above statement and **agree** to make my Honors Project available to the Macalester College community and to the larger scholarly community in our permanent digital archive the DigitalCommons@Macalester or its successor technology.

Signed Stiliyana Stanescue

OR

I **do not want** my Honors Project available to the larger scholarly community. I want my Honors Project available only in the library, **NOT** for interlibrary loan purposes, and **NOT** through the Macalester College Digital Commons or its successor technology.

Signed_____

NOTICE OF ORIGINAL WORK AND USE OF COPYRIGHT PROTECTED MATERIALS:

If your work includes images that are not original works by you, you must include permissions from original content provider or the images will not be included in the electronic copy. If your work includes discs with music, data sets, or other accompanying material that is not original work by you, the same copyright stipulations apply. If your work includes interviews, you must include a statement that you have the permission from the interviewees to make their interviews public.

BY SIGNING THIS FORM, I ACKNOWLEDGE THAT ALL WORK CONTAINED IN THIS PAPER IS ORIGINAL WORK BY ME OR INCLUDES APPROPRIATE CITATIONS AND/OR PERMISSIONS WHEN CITING OR INCLUDING EXCERPTS OF WORK(S) BY OTHERS.

All students must sign here.

Signature: Stiliyana Stanescue **Date:** 5/4/09

Printed Name: Stiliyana Stanescue

Solving the Maze: Robot Localization Using the Monte Carlo Localization Algorithm and Shape Context

Stiliyana Stamenova

Department of Mathematics and Computer Science
Macalester College

Susan Fox, First Reader
Libby Shoop, Second Reader
Andrew Beveridge, Third Reader

April 29, 2009

Abstract

Automatically navigated cars that drive on their own and robots that can perform any manual work: science fiction authors predicted that it would all be possible in the 20th Century (see Asimov, 1951 "I, Robot"). In the dawn of artificial intelligence, the main obstacle that has to be overcome was creating robots that are able to solve logical problems. Humans, as far as we know, are unique in being able to solve abstract logical problems. Thus, creating a robot to mimic this ability is an extremely complex task.

What researchers initially perceived as a much easier task - having a robot navigate and interact with the physical environment, however, proved equally difficult. Robot vision and localization are just two of the tasks that need to be solved effectively before robots can truly do what Asimov predicted.

Robot localization specifically is at the heart of any artificial intelligence system that needs to navigate in the physical world. It is not possible for a robot to carry out tasks that involve moving in the environment if every time it moves, it gets lost.

In this paper we examine an implementation of the Monte Carlo localization algorithm for the Pioneer 2 robots. Our implementation of the algorithm utilizes the robot motion sensors, sonar sensors and camera to gain as detailed a picture as possible of its environment to allow it to navigate successfully.

Contents

List of Figures

1 Introduction

Consider a robot that has to deliver a piece of mail from one college campus building to another. To be able to deliver the mail, the robot first needs to be able to carry the mail and to move. If the robot can do both, then what it needs to do is find the way between the two buildings. But to be able to find the way between the buildings, it needs to know where it is located at any one point. It may also need to have data about its environment such as a map. *Localization* is the problem of determining the position of an object within the environment. Thus, *robot localization* is the problem of determining where a robot is within the environment.

There are two main problems that fall under robot localization - *tracking* and *global localization* [23]. Given a robot that knows its initial position, as it moves it needs to constantly update its internal knowledge of the position to its new position in the environment. This is the problem of tracking. If a robot was to be kidnapped, however, and placed at an unknown (to it) position in the environment, the robot needs to solve the problem of global localization to determine its position. These two problems of robot localization are very different. In the first case, the robot needs to maintain its data about the position, in the second case, the robot needs to determine its original position.

Returning to the robot that needs to deliver a piece of mail, it will need to be able to track its position. The robot knows its original position, as well as the direction and the speed with which it moves and for how long, so tracking should be a trivial problem. There is something the robot doesn't know, however - how it interacts with the environment as it moves. If the tires of the robot have less air, it will move differently, and hence more or less than it expects to move. Similarly, different surfaces produce different friction with the tires, so it will turn more or less than it expects and so on. The robot needs to have detailed knowledge of its interaction with the environment to be able to correctly determine its new position as it moves. Thus, different robots will be able to estimate their new position with different accuracy.

In this paper we consider the Pioneer 2 robots [18]. A Pioneer 2 robot can have an estimate of its position that is off by as much as 5 meters and an estimate of its orientation that is as much as 180° off. We are trying to solve the tracking problem - we want to design a system that allows the Pioneer 2 robots to successfully track their position as they move in the environment. We also want to be able to use the techniques developed for the solution of the tracking problem to solve the global localization problem.

2 Background

2.1 Robot Sensors

There are two kinds of robot sensors that we differentiate between. *Range sensors* provide data about the distance between the robot and objects around it. *Vision sensors* provide data about what the surroundings of the robot look like.

2.1.1 Range Sensors

The most common types of range sensors are touch sensors, sonar sensors and laser sensors.

Touch sensors give the robot information about whether it is in contact with another object and how much pressure the contact is exerting on the touch sensor. Touch sensors are useful in determining whether the robot has hit an object in the environment, or whether it is holding an object in the environment.

Sonar sensors and laser sensors both provide the robot with information about the distance to other objects. They send out a signal and when the signal bounces off an object and returns, the time it took for the signal to return can be used to calculate the distance. Laser sensors are more precise than sonar sensors because the rays travel faster and they are less dispersive than sonar rays. Thus if a sonar sensor and a laser sensor measure the distance in the same direction, the laser sensor will measure the distance first. If the robot is in the act of turning, or moving further away, the laser sensor will produce a much more accurate result.

The two main problems with sonar sensors are both due to the reflectivity of surfaces. Some surfaces reflect sonar sensors less than others. This means that fewer sonar rays will return to the robot. If none return, the robot will determine that the rays did not return because there is no object for the rays to bounce off, when in fact there is. The second problem with sonar sensors is that when sonar rays hit a surface at an angle, they may be reflected away from the robot. This will also cause no rays to return to the robot and the robot will conclude that there is no object in this direction.

Another consideration for sonar sensors is that they cannot be placed too close to each other on the robot. If they are too close, the rays from one sensors may be detected by another sensors, which will also produce faulty data. This limits the number of sonar sensors that can be placed on a robot. A Pioneer 2 robot, for example, has 16 sonar sensors.

Even though sonar sensors can provide inaccurate data, they still provide data about the environment which can be used for robot localization. It is possible to improve on the data provided by the sonar sensors by, for example, taking multiple measurements over a short period of time and using an average of the measuments. However, the accuracy of the sonar readings will affect the accuracy of the localization.

The Pioneer 2 robots that we consider have touch sensors and 16 sonar sensors, but no laser sensors.

2.1.2 Vision Sensors

The most common vision sensors in robots are cameras. Robot cameras can be used to periodically take still images of the robot's surroundings, or to continuously monitor the surroundings through what is essentially a live video.

A live video will give the robot more information about the surroundings since it will give continuous

information. However, video processing is time consuming since it involves expensive calculations. Thus if a robot needs information quickly, it is often faster to work with still images. The drawback of still images is that they provide discrete information about the environment. In the time between two still images are taken, the environment can change drastically.

The number of cameras that a robot has at its disposal also changes the accuracy with which it can perceive the environment. A single camera provides a 2-dimensional image of the environment that cannot be used to determine the distance between the robot and objects in the image. Two cameras are required for depth perception. Many of the robots used in research only have a single camera since most robot applications only require a single camera. Thus, since the Pioneer 2 robots only have one camera, we cannot rely on the images from the camera to determine the exact position of objects in the environment with respect to the robot.

2.2 Robot Localization Without Vision

In this section we discuss several methods used for robot localization. We discuss robot localization based on its knowledge of its movement (*dead-reckoning*) and two statistical approches to localization - *Kalman filter-based* localization and *Markov* localization. *Monte Carlo* localization, the algorithm which we implement for localization, is described separately in Section 3.

2.2.1 Dead-reckoning

Dead-reckoning is the simplest robot localization technique. As the robot moves through its environment, it has an estimate of the distance and the direction in which it has travelled, as well as any change in its orientation. Dead-reckoning uses the robot's initial position and this data it has about its movement to determine its position at a given time. As previously mentioned, however, the data the robot has about its movement does not always correspond exactly to its actual movement. Thus, dead-reckoning does not take into consideration the interaction between the robot and the environment and it cannot be used for accurate robot localization unless the robot has very precise data about its movement in the environment. The Pioneer 2 robots do not have precise data about their interaction with the environment, so dead-reckoning produces a large error in the position estimate.

Moreover, dead-reckoning relies on the initial position of the robot to update the current position. If the initial position is not known, the robot has no way of determining its exact location in the environment.

2.2.2 Kalman Filter-based Localization

A *filter* in this context is a set of mathematical equations that are used to remove noise from given data in order to predict the current state of the system - in our case, the position of the robot in the environment. The Kalman filter is a recursive filter which uses the measurements for the current

step along with the prediction from the last step to give a prediction about the current state of the system [29].

Kalman filter-based localization algorithms are statistical algorithms for solving the tracking problem. Most such algorithms assume that the uncertainty of the robot's location and sensor data can be modeled as Gaussian-shape distributions [11]. In this case, the sensor measurements are expected to cluster around the real value of the parameter that is being measured. Sensor data, however, cannot always be modeled as a Gaussian-shape distribution. Consider, for example, a robot moving towards (or away from) a wall at an angle. If the sonar rays are mostly bouncing off away from the robot, the sonar measurements will not cluster around the actual distance. In fact, the sonar measurements may cluster around a completely wrong measurement.

Furthermore, algorithms that use Kalman filters assume that the error in the robot's position can be represented by a normal distribution (in fact, a unimodal Gaussian distribution). In the case of global localization, it is possible that more than one location is highly probable for the robot. Thus, the distribution of the uncertainty in the robot location may be multimodal [11]. This means that Kalman filter-based localization algorithms cannot be used effectively to solve the global localization problem.

2.2.3 Markov Localization

A different approach to localization is to represent the robot's environment as a set of discrete positions L. Unlike Kalman filter-based localization, this approach requires that the robot has prior knowledge about its environment, usually a map. Let $l \in L$, $l =< x, y, \theta >$, where x and y are the robot's coordinates in the environment and θ is its orientation. At each iteration, the robot's belief that it is at position l: $Bel(l)$, is updated with the new sensor and motion data.

```
MARKOV LOCALIZATION ALGORITHM
for each location l ∈ L:
    initialize the belief for l
end for
while(true):
    if new sensor data s_T is available:
        for each location l ∈ L:
            update the belief for l using perception model
        end for
    end if
    if new motion data m_T is available:
        for each location l ∈ L:
            update the belief for l using motion model
        end for
    end if
end while
```

Figure 1: Pseudocode for the Markov Localization algorithm [10]

Fig. 1 shows tha pseudocode for the Markov localization algorithm. When the initial position of the

robot is available, the belief is initialized to a normal distribution centered at the initial position. When the initial position is unknown, the belief is initialized to a uniform distribution over the whole environment. The Markov localization algorithm can then be used both for tracking and for global localization.

Markov localization has two key requirements for the environment. The first requirement is that the position at time t only depends on the position at time $t-1$ and the movement of the robot at time $t-1$. The second requirement is that the reading of a sensor at time t is independent of the readings of any other sensor i.e. the reading of any sensor only depends on the environment [10] [11]. An environment that fulfills those requirements is called *Markovian.*

Markov localization is more accurate than Kalman-filter based localization and can be used for global localization. However, Markov localization is difficult to implement. The algorithm also requires that the robot's environment is split into a fine-grid to assure accurate results [11]. Thus, accurate localization requires a large number of points to be updated at every iteration which makes Markov localization time- and memory-expensive.

Since Markov localization can be very successful, however, it is the basis on which the Monte Carlo localization algorithm that we use is built. We describe the Monte Carlo localization algorithm in detail in Section 3.

2.3 Robot Vision

Robot vision, and more specifically, object recognition is one of the hard tasks in artificial intelligence. If a robot could recognize features, or locations in its environment, however, it could use that knowledge for the purpose of localization. There are two main approaches to object recognition - brightness-based recognition and feature-based recognition. Here we discuss both briefly and give a brief justification for our choice of object-recognition technique to use with Monte Carlo localization.

2.3.1 Brightness-based Recognition

Given an image of an object, *brightness-based recognition* techniques assume that the object is described by the entirety of the image. That's why brightness-based recognition is also called "recognition by appearance". The image is then treated as a vector of pixels [17], so that it can be more efficiently compared to other images. Images are compared pixel-by-pixel by grayscale value [7]. Fig. 2 shows two handwritten digit 5s. To a human observer both shapes represent the same digit, however, when compared pixel-by-pixel the two shapes are not the same. Different techniques such as decision trees, neural networks and naïve Bayes models are used to determine the similarity of the images [26].

This method can be very successful in some contexts. Two examples are handwritten digit recognition using LeNet classifiers [16] and automated pedestrian detection [9]. The method can be inefficient, however, since it requires the entire image to be stored for future comparisons. The

Figure 2: Two handwritten digits that are the same to a human observer, but different when compared pixel-by-pixel (Image taken from [7])

main problem with pixel-by-pixel comparison by grayscale value is that depending on the angle from which the image is taken and the focus and lighting of the image, there are slight (and sometimes even drastic) variations in the grayscale value of the pixels that describe the same part of an object. This means that brightness-based recognition is not invariant to angle and lighting. When trying to locate a feature in the environment, however, invariance to angle and lighting can be crucial.

2.3.2 Feature-based Recognition

We consider an image of an object: for example a black cat. The tail of the cat is made-up of multiple pixels, and adjacent pixels have similar coloring. The cat's tail is a *single* feature of the cat, but it is composed of many pixels. If we could store data about the cat's tail without having to store all the pixels that describe it in the image, comparison to other images would be more efficient. We would only need to look for the tail and not all of the pixels of the tail.

Feature-based recognition is based on the idea that it is possible to describe the feature of an object through its contours, or through regions of the object that are most distinct. This yields less data that needs to be stored and compared and thus an increase in speed in object recogition. Since the contours of an object can be detected under most kinds of lighting, feature-based recognition that uses edges is also illumination invariant [23]. Moreover, if the contours of the object are used for its recognition, then geometrical transformation can be applied to two images that are being compared. This means that feature-based recogntion is not only illumination invariant, but also invariant to angle and distance. Fig. 3 gives an example of feature-based recognition done by matching points in the contours of two images.

There is a drawback to using feature-based recognition. Since not the whole image, but only the features of the image are stored for future comparisons, only a small part of all the pixels in the image are stored. This is more space-efficient, but it means that comparison algorithms that use features need to be more effective in determining the similarity between objects. Most feature-based recognition techniques use nearest-neighbour classifiers such as Hausdorff distance [13], Fourier descriptors [20], and Markov Chain Monte Carlo methods [8].

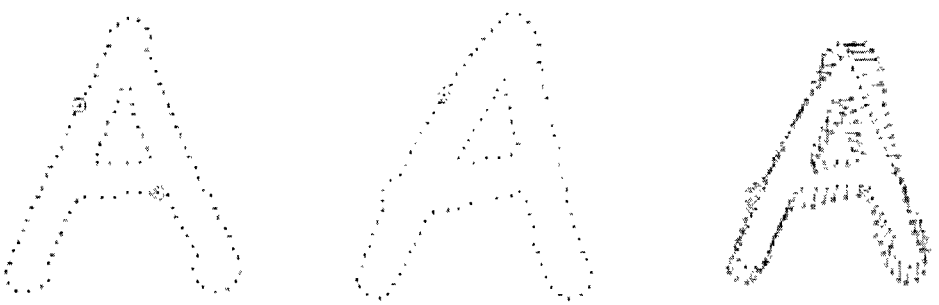

Figure 3: Points in the contours of two shapes and a correspondence between the two contours (Image taken from [7])

2.4 Robot Localization with Vision-based Techniques

Some of the different localization techniques can be very efficient, but they have special requirements (such as a Markov environment) that can be unrealistic. Vision can be used to localize in a more complex environment, vision can be used. In this section we describe two different techniques for localization using vision.

The most common technique for vision-based localization is landmark localization. *Landmark-based* localization operates on the notion that there are recognizable landmarks in the environment that can be used for robot localization. Such landmarks can be artificial (such as bright-colored paper placed at specific places for the purpose of localization), or natural (such as doors in hallways). In both cases, the robot needs to have prior knowledge about its environment and the location of the landmarks [3].

2.4.1 Single Landmark-based Localization

Given a set of unique landmarks in the environment, each unique landmark is associated with a specific location. Thus if a robot detects one of the landmarks it will know its exact location. The problem with such an approach is that in a normal environment there usually aren't enough unique landmarks to continuously localize. This means that the robot will know its exact location at discrete points and as it moves between this points, its estimate of its location will become less accurate.

One way around this problem is to place artificial landmarks in the environment. This means that a sufficient number of unique landmarks have to be placed in the environment. On the one hand, we can use a large number of unique markers. However, this increases the probability of the markers being tampered with. On the other hand, we can use a smaller number of markers placed in strategic locations that will yield best results. In this case, however, if a single landmark is removed, the performance of the localization will suffer greatly. In both cases the system is susceptible to errors when changes in the markers are introduced.

2.4.2 Multiple Landmark-based Localization with a Single Camera

Multiple markers can be used to overcome the errors of single landmark-based localization instead. Typically at least three landmarks need to be visible to the robot at any time it needs to localize, so that it can use triangulation to determine its location. One of the more successful approaches to robot localization based on multiple landmarks was developed by Atiya and Hager[3].

Given a set of three observed landmarks and three landmarks stored by the robot, a spatial transformation can be applied to align the points. This kind of landmark recognition is illumination and distance invariant. Once the two sets of three points are aligned, they can be compared for similarity. Since it is possible for multiple sets of three landmarks to have high similarity, an error threshold is given to the algorithm and all matches that have a smaller error in similarity are returned by the algorithm. This part of the localization technique used by Atiya and Hager requires that all landmarks are known by the robot prior to localization. Moreover, this approach also requires that all necessary data for comparison with the available landmarks is pre-computed to maintain the rum-time efficiency of the algorithm. This means that a large amount of data needs to be stored.

Two problems can occur at this stage of the localization - multiple sets of landmarks can be consistent with the observed data, or some of the detected features may not be actual landmarks [3]. If some of the detected features are not landmarks, the algorithm will produce erroneous matches for the robot's location. If multiple sets of landmarks are consistent, the algorithm will not be able to determine which set is the correct one. In both cases, the algorithm will produce multiple possible locations. But since the only available data is the original image of the environment, the robot will still have no way of determining which of these locations is correct.

We can limit the number of probable locations, but not determine the precise global location. The implication is that this method cannot be used for exact global localization. Since this method can only produce results when three landmarks are in view of the robot, it also cannot be used for tracking, which is the main problem we are trying to solve. Moreover, it requires a *stereo* image of the environment, and we only have a single camera and hence a 2D image of the environment instead of a 3D image.

2.4.3 Multiple Landmark-based Localization with Multiple Cameras

In the case when the robot is not equipped with a camera able to take stereo images, multiple cameras can be used for vision-based localization. One such approach was developed by Se, Lowe and Little [24].

This method compensates for the fact that each image is two-dimensional by using the images from all the cameras concurrently. The effect is similar to that of having a single stereo camera and localization can be very successful [24]. Unfortunately, the Pioneer 2 robots are equipped with a single camera, so we cannot use a similar approach.

3 Monte Carlo Localization

3.1 Overview of MCL

Monte Carlo localization (MCL) is a statistical approach to robot localization based on Markov localization. Most of the work on MCL was done by Fox, Burgard, Dellaert and Thrun [11]. Like Markov localization, MCL requires a map of the robot's environment. It is a *particle filter* [11], so the belief $Bel(l)$ of the system is represented by n particles, or samples. Each sample is of the form $< l, p >$ where $l =< x, y, \theta >$ as in the Markov localization approach and p is the normalized probability that the robot is at position l.

The first step of MCL is initialization of the sample set. When the initial position of the robot is known, the sample set is drawn from a normal distribution centered at the robot's location. When the initial position is unknown, the sample set is drawn from a uniform distribution over the entire environment of the robot.

There are two main steps to the algorithm analogous to the two steps in Markov localization:

Step 1. Motion model update

As the robot moves through the environment, it keeps track of how far it thinks it has moved and in which direction. It also keeps track of how much and in which direction it has turned. As previously discussed, however, this information may not be accurate. When the robot sends the information about its movement to the algorithm, the samples are updated. However, the new position of the sample is determined by randomly sampling a normal distribution of possible new locations based on the old location and the robot's movement [11]. This normal distribution is centered at the value received from the robot. Each sample is, thus, randomly moved as opposed to moving all samples in the same way.

Step 2. Sensor model update

As data about the robot's environment is measured by the robot's sensors, the data is passed through MCL. The samples are then updated by re-weighing their probability [11]. Given the sensor measurements s received from the robot and a sample $< l, p >$, the probability $P(s, l)$ of observing these measurements at location l is calculated from the map [11]. The probabilities for all samples are then normalized.

Before the motion model update (or as part of the motion model update), a new set of samples is generated. To create the new set, samples are drawn randomly from the old set using the probabilities p_i assigned to each sample i [11]. That is, samples that have a higher probability p_i assigned to them are more likely to be included in the new set of samples.

Since MCL is based on Markov localization, it can be used for both global localization and tracking. MCL is particularly well-adapted for tracking since it provides a framework in which the robot's belief of its position is continually tested and adjusted.

MCL, however, also works best in a Markovian environment and with accurate sensor measurements.

3.2 The Monte Carlo Localization Algorithm

In this section we present the Monte Carlo localization algorithm as implemented for use with the Pioneer 2 robots with the use of sonars for the sensors. There are several minor differences between the original algorithm of Fox et al. [11] and our algorithm. Most importantly we split the robot motion model update into two steps: drawing random samples from the old sample distribution and then updating the new samples using the motion model. We describe the algorithm in its use for tracking.

```
MONTE CARLO LOCALIZATION ALGORITHM
generate the set S of samples s =< l, p > (S has size n)
while(true):
    get new sensor data s_T and new motion data m_T
    for each sample s =< l, p >∈ S:
        calculate l' using l and m_T
        set s =< l', 1 >
        calculate p' using l' and s_T
        set s =< l', p' >
    end for
    for i = 0 to i = n − 1
        set s_i to a randomly drawn sample from S
    end for
    calculate the standard deviation and mean position of the sample S
    if s_T differs from the expected sensor data for the current position
        update the position the robot believes itself to be
    end if
end while
```

Figure 4: Pseudocode for the Monte Carlo Localization algorithm

Since in this case we are using the algorithm for tracking, in the initialization phase we generate the sample set by drawing the samples from a normal distribution centered at the robot's initial location. All the samples that are drawn are within at most 10 meters of the robot to ensure that they will converge faster. If the samples were drawn uniformly over the entire environment, the convergence would be slower and the algorithm could not be used effectively for tracking.

When new sensor and motion data is received, we first update the samples using the motion data. As in the original Monte Carlo algorithm, the new position of each sample is determined by drawing from a normal distribution centered at the expected new position given the robot's movement data. The new probability of the sample is set to 1 which means that all new samples are equally probable. We set the probability to 1 as opposed to $\frac{1}{n}$ in order to simplify the calculations.

After all the samples are updated using the motion data, the expected sonar data for each sample is calculated from the position of the sample and a map of the environment. The expected sonar data

is then compared to the observed sonar data and the new probability of the sample is computed. Once the new probability of all samples is computed, the probabilities are normalized so that their sum is n.

The last step of the original algorithm is creating a new sample set. Given the original sample set with updated values for location and probability, m new samples are drawn from the old sample set at random with the probability attached to each sample. We draw m samples, where $m < n$ and n is the original set's size, and then add $n - m$ randomly generated samples as suggested in [11]. The purpose of adding random samples is that in the event of the robot losing its localization, it will be able to re-localize successfully. We draw m random samples where m is 5% of n.

The additional step which has been added to our algorithm can be performed at any time, but we find it more successful to perform at the end of the update phase. We use the location the robot currently believes itself to be, we calculate the the expected sonar data. The expected sonar data is then compared to the observed sonar data and if a large discrepancy is found, the location the the robot believes itself to be is updated to the mean position of the sample set S. We are using the mean of S for the new position, since most of the samples in S are supposed to be converging close to the robot's actual position since they will have the highest likelihood.

Since MCL is an online algorithm, it can be implemented to generate an answer at any time [11]. As the algorithm gets more data its answer becomes more accurate, so we have found it more effective to only use the answer when it is needed i.e. when the discrepancy between the observed and expected sonar data is large.

3.3 Motion and Sensor Models

3.3.1 Motion Model

The motion model, or the odometry model, uses the observed data for the robot's movement m_T to update each sample. Since the movement m_T is the robot's belief of how it moved, the actual movement may not correspond exactly to m_T. To account for the possible error, a probabilistic component is added when each sample is translated [28].

Given a sample $s =< l, p >$, the probability that a location l' is the new location of the robot is $P(l'|l, m_T)$. That is, the probability of l' depends both on l and m_T, but not on any other data. This is called a *Markov* assumption [28]. What is important about this assumption is that we don't need to keep a history of all of the robot's previous positions and movements - we only need the last known position and motion data to compute the expected new position.

Following [28] instead of implementing a closed form computation for $P(l'|l, m_T)$, we use a *sampling model*. That is, we have a routine *moveSamples(dX,dY,dTh)* which takes the components dX (movement in the x direction), dY (movement in the y direction) and dTh (rotation) of m_T and then updates each sample based on m_T. For each sample, we draw a random number r from a Gaussian distribution centered at 0 and we use r to translate each of the components of m_T by r. For example, the new x coordinate of the sample x' is computed as $x' = r + x_{old}$, where x_{old} is the

sample's old x coordinate.

We have added an additional step to our routine not described by [28] to guarantee that the coordinates of each sample remains within the map (we don't want the robot to think it fell off the edge of the world). This is the only difference between our model and the model proposed by [28].

After the update to the location of the sample has been executed, the probability of the sample is set to 1 guaranteeing that all samples have the same probability.

Figure 5 show the effect of the motion model on a set of samples. The straight line represents the robot's actual movement. We can see that as the robot moves, the samples start to scatter due to the probabilistic component. Note that the figure shows only the effect of the motion model in the absence of the sensor model.

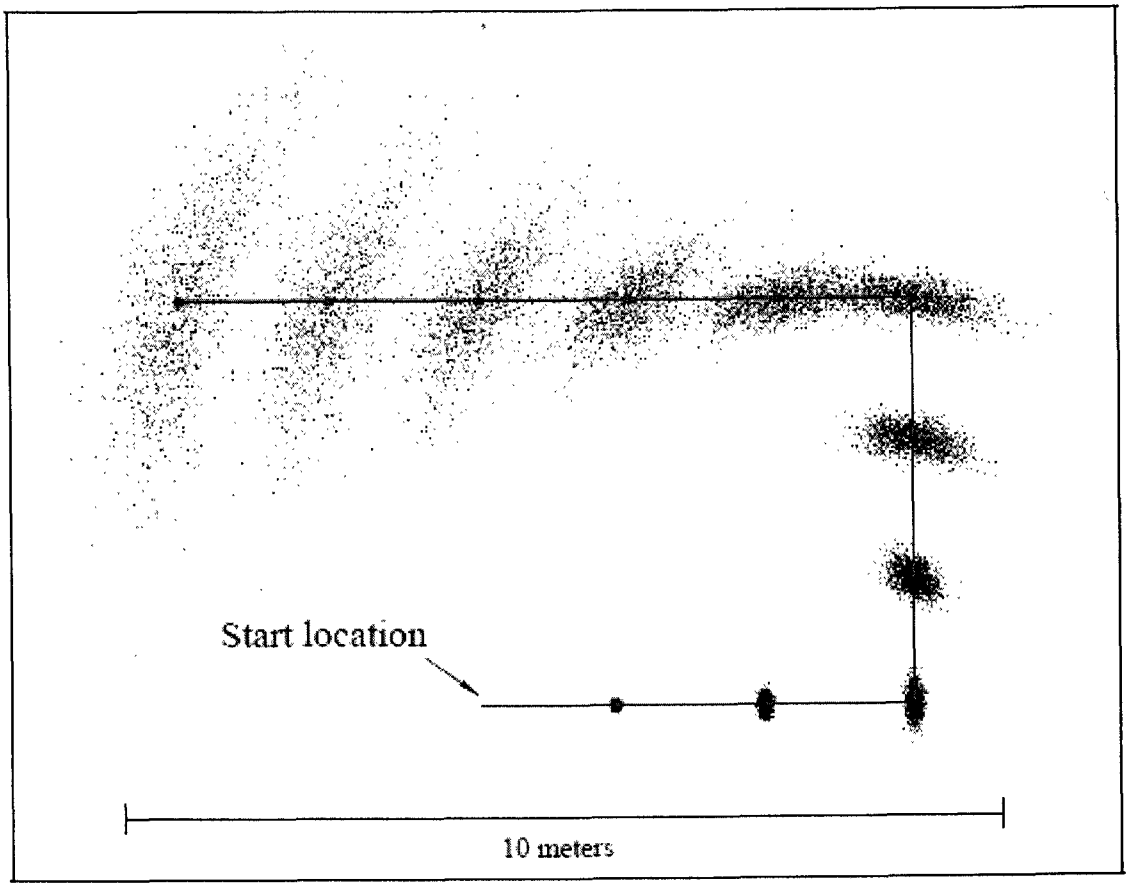

Figure 5: Approximation of the position of the robot using probabilistic sampling (Image taken from [28])

3.3.2 Sensor Model

The sensor model, or the perceptual model, relies on the fact the the robot is equipped with a map of its environment. It uses the observed data from the robot's sensors s_T to update the probability (or weight) of each sample.

Given a sample $s =< l, p >$ and observed sensor data $s_T = \{o_i\}$, the new weight of the sample is computed in several steps. Here o_i is the value measured by the i^{th} sonar.

Step 1

Using l and the map of the environment, the expected sonar value e_i is computed for each sonar i. We can do this easily by computing the straight-line distance between l and the closest obstacle on the map in the direction of the sonar ray i since we know the direction in which each sonar ray is sent.

Step 2

Given the expected value e_i of each sonar reading, we calculate the probability $P(o_i|e_i)$ that the value o_i is measured by the i^{th} sonar. We assume that the distribution of the probable readings at l is a normal distribution centered at e_i. We can then approximate the probability that o_i is measured at l as $P(o_i|l) = P(o_i|e_i)$. By using the value $P(o_i|e_i)$ for the value of $P(o_i|l)$ we allow for noise in the sensor data.

Step 3

Once the probability of each o_i has been calculated, we need to calculate the probability of s_T at l. We let $P(s_T|l) = \prod_{i=1}^{n} P(o_i|e_i)$. This assumes that $P(o_i|e_i)$ is independent of $P(o_j|e_j)$ for $i \neq j$. Given a single measurement s_T, however, $P(o_i|e_i)$ is not always independent of $P(o_j|e_j)$ [28]. For example, in the presence of people or objects that block both rays, the two probabilities will not be indepedent. In our case, however, we use an average of several sonar measurements for s_T. Thus in most cases $P(o_i|e_i)$ will be independent of $P(o_j|e_j)$, so we can make the assumption that this is always the case.

Step 4

After the probability $P(s_T|l_i)$ is calculated for each sample i, we sum all the probabilities to get $t = \sum_{i=1}^{n} P(s_T|l_i)$. We then set the weight p_i of the sample i to $p_i = \frac{1}{t} P(s_T|l_i)$ to normalize the weights. This guarantees that $\sum_{i=1}^{n} p_i = 1$. We can now use the new sample set S for re-sampling.

3.4 Implementation

3.4.1 Localization Device

The brain for the Pioneer 2 robot that we use is implemented in Python and works through Pyro [21]. The brain is part of the RUPART hybrid robot control implemented by Professor Susan Fox. The robot's movement and sensor data is retrieved from the robot using Python. Heavy mathematical computations, however, such as the ones required for MCL are not efficiently implemented in Python, so our MCL implementation is done in C++. This means that our first task was to implement an interface for communicating data between the robot's brain and MCL. This was done by implementing a localization device for RUPART in Python, which then opened a pipe to the MCL implementation in C++. Data between the two was transmitted through the pipe. Fig. 6 shows a screenshot of the data provided by the localization device to the robot brain.

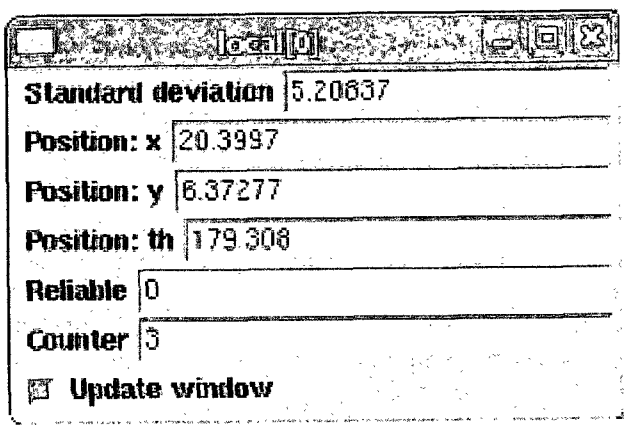

Figure 6: The data provided by the localization device to the brain.

3.4.2 The Map

The map of the environment is an essential part of MCL since it is used for calculating the expected values for the sonar measurements. The map was drawn using the ArMap interface from Aria [1]. Aria, however, does not provide code for map manipulation which can be easily called from C++. Thus we implemented a set of classes to read the map file from C++ and be able to store and manipulate the data easily.

The map that we used was of the science building at Macalester College, Olin-Rice. This is the building that RUPART is designed to navigate in. The map was created after manually measuring the walls in Olin-Rice.

3.4.3 Sonar Model

The next part of our implementation is a sonar model class in C++ which contains data about the locations of the sonars on the robot as well as the angles at which the sonar rays eminate from the robot. The sonar model data is used for the calculation of the expected sonar measurements from the map.

3.4.4 Samples

We implemented several classes for sample manipulation. The sample classes are used for sample intialization. They also contain the implementations of the motion and sensor models.

These classes also contain the implementation of the routines used for approximating the robot's actual position. That is, the routine *getExpectedPosition*, *getExpectedPositionValue* and *getRobotPositionValue* which provide the position where the robot is expected to be (as approximated by MCL), and values for evaluating how good the approximation is and how accurate the robot's belief of its position is expected to be. Both values are calculated by comparing the expected sonar data at these positions with the actual sonar data.

The MCL algorithm, as described in [28], does not include a calculation for the expected position of the robot. The set of samples S represents $Bel(l)$ which is the belief of the robot's location produced by MCL, but this set does not provide a single value for l. We experimented with several different possible calculations and determined that the mean of the samples S can be used to approximate l most successfully.

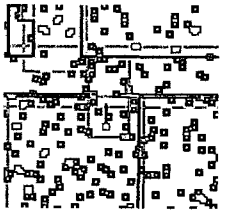

Figure 7: A small set of samples near the robot as generated by our implementation of MCL.

3.4.5 Monte Carlo

The last part of our implementation of MCL is a set of Monte Carlo classes that include the necessary interface from C++ for communicating with the localization device in Python as well as the implementation of the Monte Carlo algorithm. Since most of the routines that calculate the probabilities and evaluate the functions are implemented in the previously discussed classes, the Monte Carlo classes contain only a very high-level implementation of MCL.

3.4.6 Additional Tools

As we worked with our implementation of MCL, it became apparent that it is impossible to judge the performance of the algorithm (or discover errors) without being able to visualize the sample set. Since C++ does not provide good tools for implementing a graphical user interface, we added an implementation of a visualization tool in Java. The visualization tool displays the robot's belief of its location, along with the samples and the sonar readings on the map. The visualization tool was extremely useful in detecting errors in the algorithm as well as monitoring its performance. Fig. 8 shows the visualization tool.

3.5 Results

Our first implementation of Monte Carlo localization described above did not use the robot's camera - only the robot's sonars. Figure 9 shows the map of the science building Olin-Rice where we tested the implementation of MCL.

During our testing of MCL we found that the walls of Olin-Rice are very reflective and the robot can actually hit a wall at an angle while the sonar sensors are still detecting a distance of more than 8 m between the robot and the wall. Some of the walls are also made of glass panes with gaps in-between which produce a similar error. Thus, the sonar measurements that we got were not always accurate. Moreover, since the Pioneer 2 robots are equipped with only 16 sonar sensors, we had discrete measurements about the robot's surroundings, as opposed to continuous measurements that could be provided by a different sensor such as a laser.

Due to the fact that the sonar readings in the building where we tested MCL were not very accurate, we introduced an additional calculation for a sample's weight. After the weight for the sample is calculated using the vanilla MCL approach, samples that are located in the same area of the building as the robot's believed location are given higher weight. We found that with continuous update from MCL the robot estimates correctly which area of the building it is located in, but not the exact position, so weighing the samples using an area variable caused the samples to converge more quickly to the robot's actual position. Figure 10 and 11 show the MCL samples right after they are initialized when they have started to converge to the robot's location.

After testing the implementation of MCL we found that MCL is usually more accurate than the robot's estimation of its location through dead-reckoning. The results that we got, however, were still not very accurate: after a while the samples would converge, but sometimes to the wrong location. This resulted in erroneous update of the robot's position, which even though more accurate than the robot's belief of its location, was still not ideal. We could not reproduce the successful results of Fox et al. [11] using only the 16 sonar sensors as opposed to a laser sensor such as the one used by Fox et al.

Since the Pioneer 2 robots that we have do not have laser sensors, any improvement via better heuristical methods for evaluating samples would have produced only marginally better results. In order to improve the performance of the localization algorithm, we needed a new sensor to get more data from the environment. The other sensor that the Pioneer 2 robots have is a camera, so vision

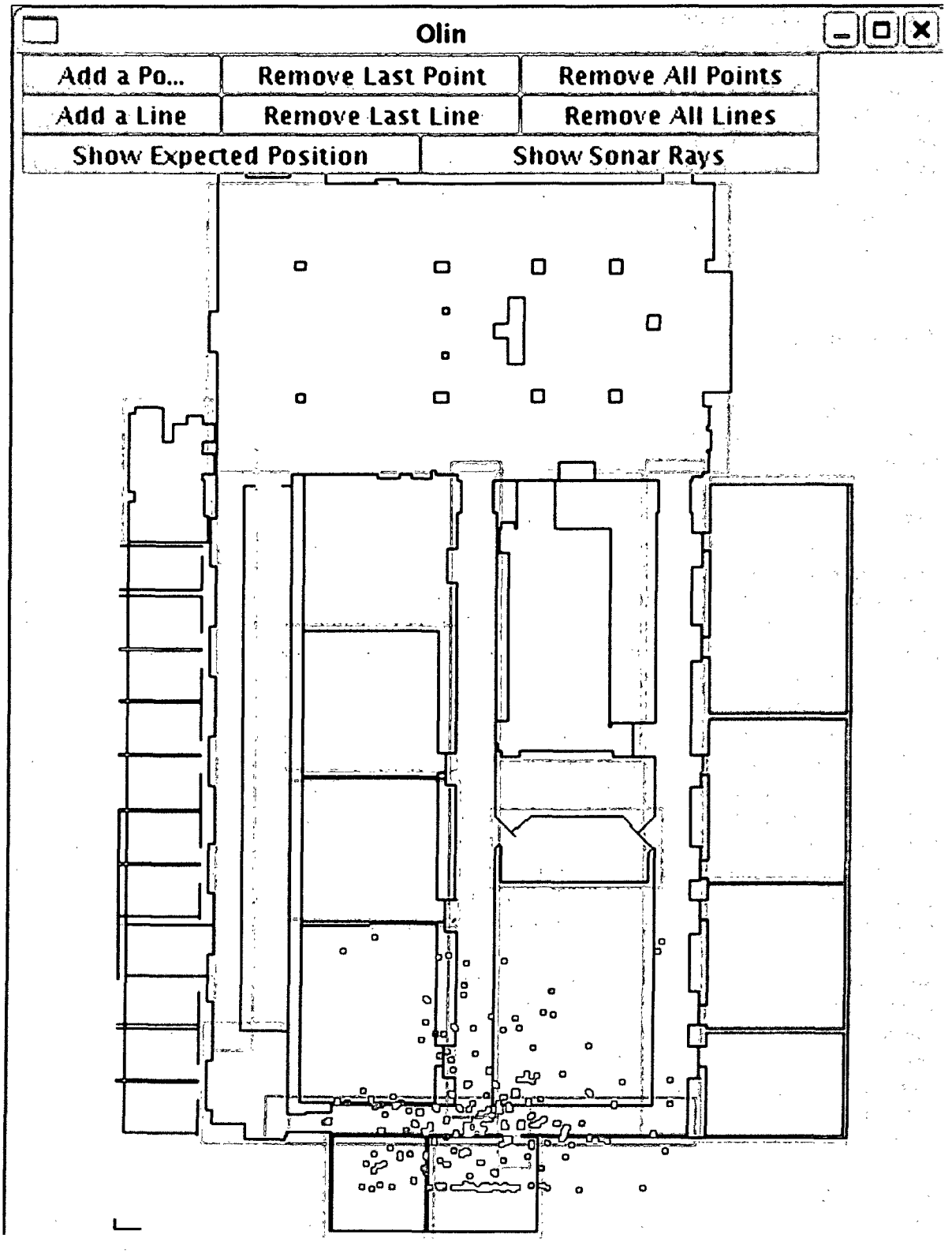

Figure 8: The visualization tool developed for use with our implementation of MCL.

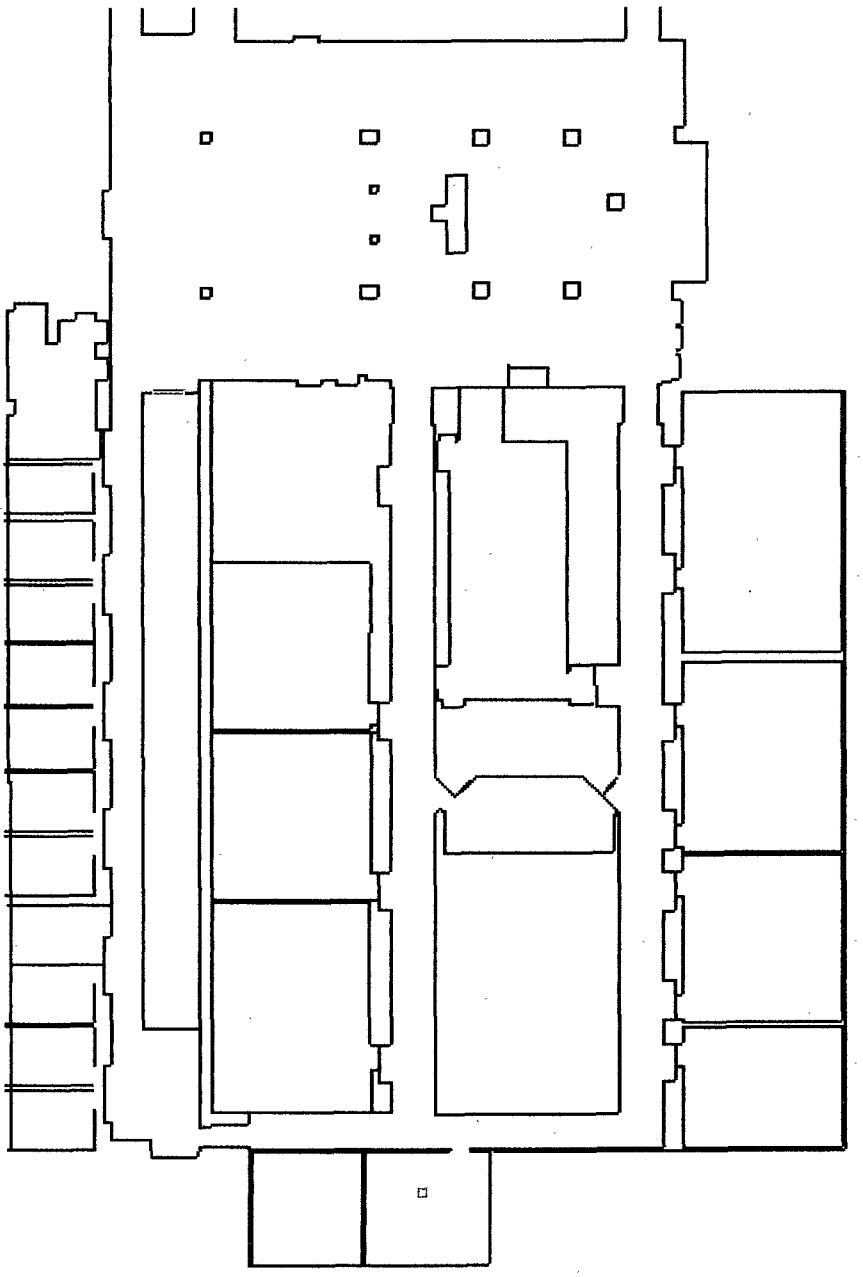

Figure 9: The map of Olin-Rice used for MCL.

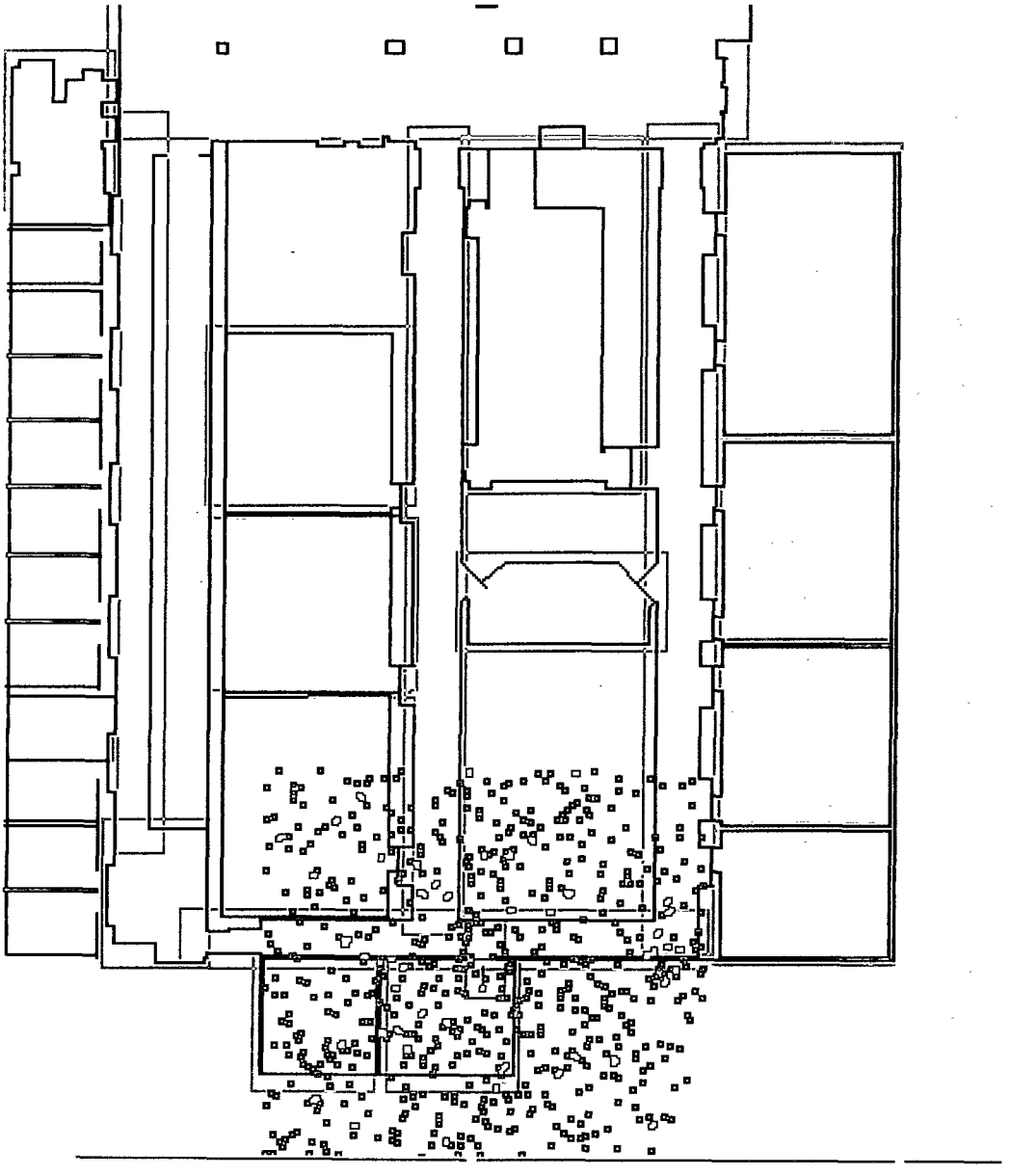

Figure 10: The initial sample distribution: Yellow rectangles represent areas. The areas that the robot is in are in cyan. The red circle shows the robot location.

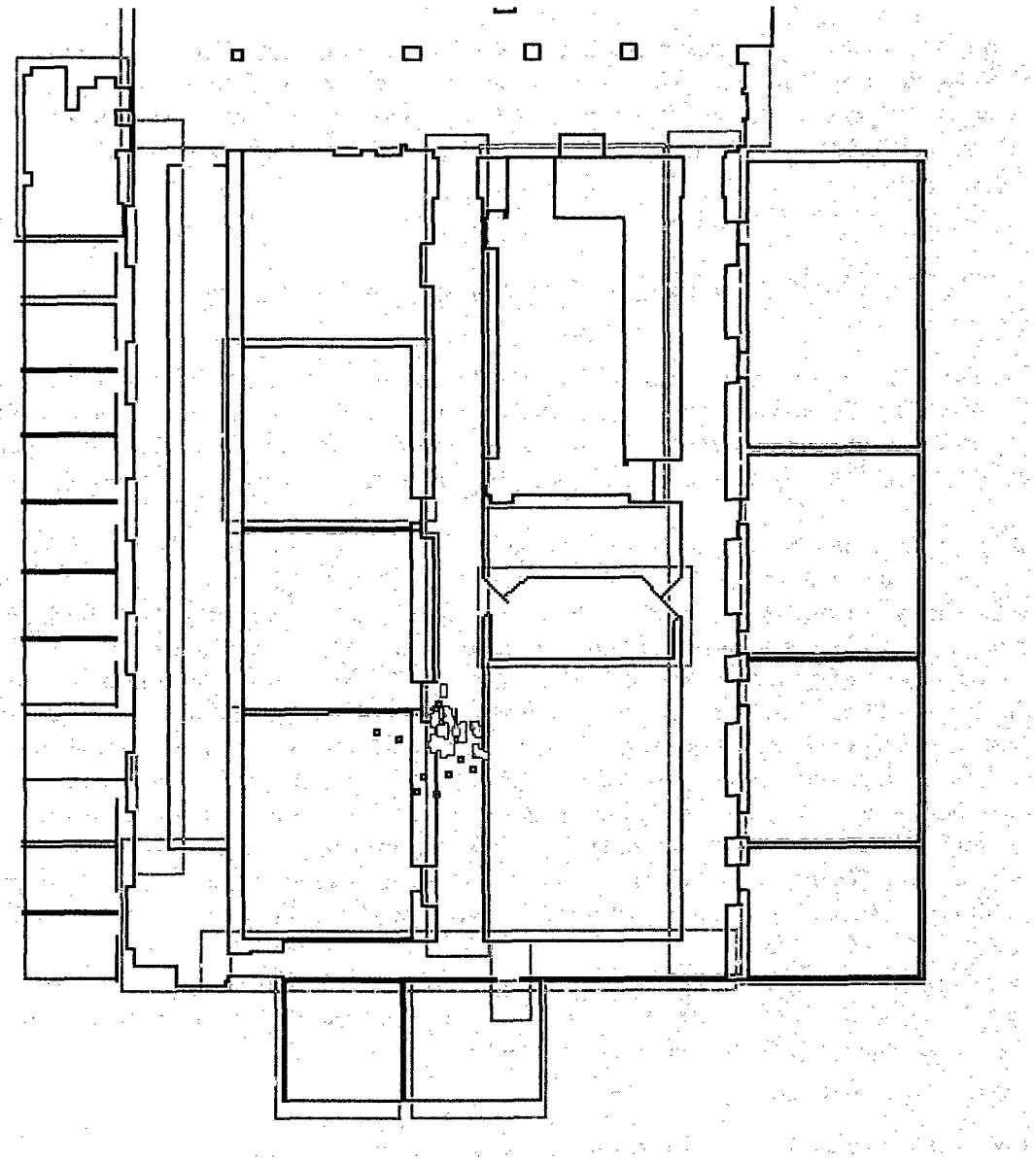

Figure 11: After several updates the samples are starting to converge where the robot is.

is crucial for better localization of the Pioneer 2 robots.

4 Object Recognition with Shape Context

In order to improve the precision of Monte Carlo localization, we decided to include vision as one of the sensors for localization. If the robot can recognize objects, or locations in its environment using vision, then it can use this additional information to perform more accurate localization. Before we add vision to the localization, however, we present an implementation of object recognition using shape context.

4.1 Overview of Matching with Shape Context

We know that the contours of an object are illumination invariant. Therefore a recognition algorithm, such as the shape context algorithm, that uses the contours as the main features of the object would also be illumination invariant. This means that the algorithm does not require any image manipulation to ensure that a change in illumination will not lead erroneous results.

Before the shape context algorithm is applied, the object contours are extracted from the given image. To compare the contours between two objects pixel by pixel, however, is also not efficient, so the shape context matching agorithm extracts a random sample of points from the internal and external contours of the object. These points are then used to create a histogram that represents the distribution of the other points in the contour with respect to the chosen points [4]. To make the shape context matching algorithm scale invariant, the histograms can be normalized and then used to compare points in the shape. The histograms represent the distance between a point and all other points and hence the set of vectors from the point to all the other points. This means that the histogram is a rich descriptor of the shape [4]. It is not necessary to compare all points in the contours between shapes, but only the points that were sampled from the inside and outside of the contours. Therefore, the shape context matching algorithm compares fewer data points than traditional object recognition algorithms and usually performs faster.

We can use the shape context algorithm to compare an image against a database of already available images to find the best match. The image that produces the best match is then an image of the object that the robot would assume it was "seeing".

4.2 The Shape Context Matching Algorithm and Implementation

The original shape context matching algorithm presented by Belongie and Malik [4] includes provisions to make the algorithm invariant to most common deformations such as translation. Since the computations required to make the algorithm deformation invariant are expensive, we chose not to include such computations. This means that the performance of the algorithm will suffer in terms of matching objects, but its speed will increase.

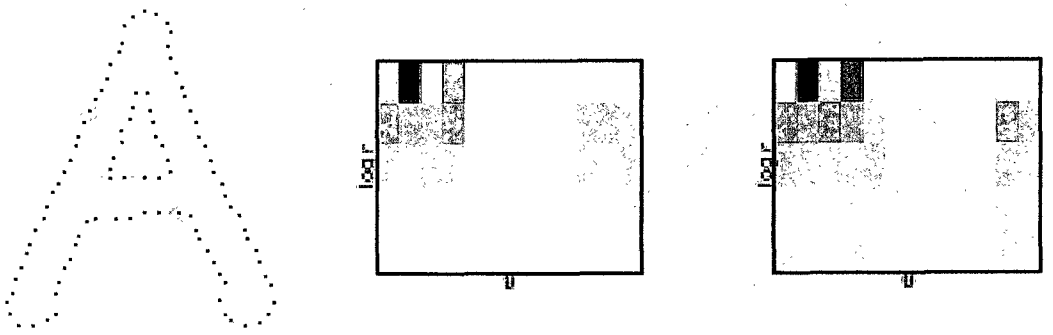

Figure 12: An example of a sample set of points extracted from a contour and the histograms for the two points markers as ○ and ◁ (Image taken from [6]).

SHAPE CONTEXT MATCHING ALGORITHM
get the contour of the object
normalize the contour
for $i = 1$ to $i = n$
  pick a sample point s_i from the contour
end for
for $i = 1$ to $i = sp$
  calculate the shape context (histogram) of s_i
end for
for $j = 1$ to $j = m$
  let c_m = cost of matching current object with object m
end for
return the object with the lowest cost c_m

Figure 13: Pseudocode for the Shape Context Matching Algorithm

4.2.1 Getting and Normalizing the Contour

To efficiently get the contours from the image, we use the built-in features from the Pioneer 2 robots. Pyro provides two filters for the robot's camera - the Sobel filter and the mean blur filter. The Sobel filter is used for edge detection. It uses the Sobel algorithm to find edges in the image and then discards all pixels that are not part of the edges. The edges are then converted to grayscale by setting the output "channels" (Red, Green and Blue) to the same value. Applying the Sobel filter detects the outer borders of the image as edges and adds a contour at the image borders. Since this contour is not actually a contour related to the objects inside the image, we ignore the outer-most 5% of each image in order to avoid using the wrong contour. Figure 14 shows an example of an image and the contour extracted from the image using the described method.

We used the mean blur filter to improve the speed of the algorithm. The portion of the algorithm that calculates the cost of matching 2 images uses the Hungarian method which executes in $O(n^3)$

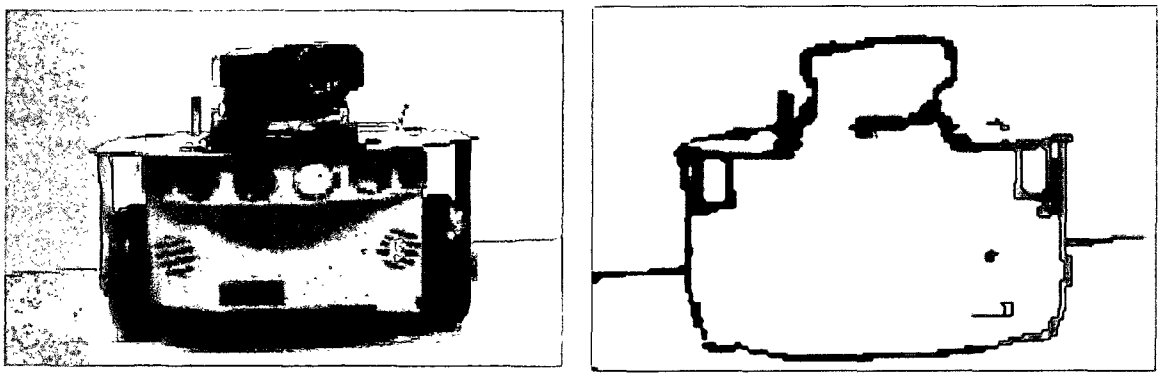

Figure 14: On the left an image of Lauren, one of the Pioneer 2 robots. On the right, the contour of the same image.

where n is the number of sample points selected from each contour, so a small change in n can significantly improve the performance of the algorithm. We use the Hungarian method to find which two points in the contour need to be matched to produce the lowest possible cost. It is a graph algorithm for solving the assignment problem.

The mean blur filter selects a pixel and sets its value to the mean value of the pixels in the square that contains it. We used a 3x3 square for the filter and then selected only the center of the 3x3 square for the contour. This decreased the number of pixels in the contour by a factor of 9, which reduced the number of calculations for the Hungarian method by a factor of 9^3. Fig. 15 shows the effect of the mean blur filter on a 3x3 square.

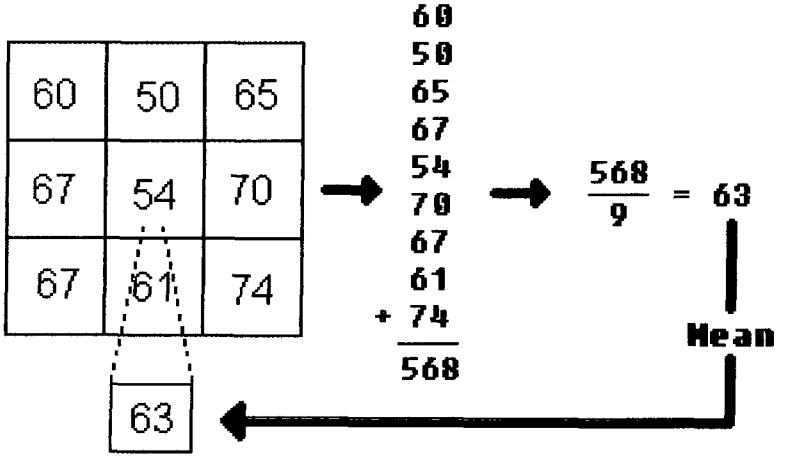

Figure 15: The mean blur function for a 3x3 square (Image from "Pyro, Python Robotics", 2006).

After the contours from the image are extracted, we normalize the contours. Thus, our implemen-

tation of the shape context matching algorithm is scale invariant. To normalize the contour, we sum the distances between all pairs of points and then average the sum over the number of points. The value obtained is λ. We can then use λ to scale the contour.

4.2.2 Selecting Sample points

The next step in the algorithm is selecting the sample points. Before the sample points are selected we need to determine how many sample points should be selected. Belongie et al. [7] use 300 points per object. However, we found that we can't always extract 300 points from the contour since we are reducing the number of points in the contour by a factor of 9. Instead we select 50 points from each contour. This guarantees that we can find enough points from the contours and reduces the time for the computations.

The sample points are not selected according to any special criterion - it is not necessary that the points belong to key elements of the contour such as maxima of curvature [6].

In the rare event when there aren't 50 points in the contour, we follow [6] and add "dummy" points. We find the smallest matching cost c for every particular set of points and then add dummy points of cost close to c. If all the "dummy" points that we add have the same cost, the performance of the Hungarian method slows down, so instead we pick a randomly generated value close to c, so that the "dummy" points don't have all the same cost. This is a divergence from the original algorithm presented by Belongie et al. in [6] (they use a small constant cost for the "dummy" points), but we don't expect a change in the effiency of the algorithm [26].

To select 50 points from the contour, we use a random number generator to produce a value within the number of points of the contour [26] and then pick the corresponding point in the contour.

4.2.3 Calculating the Shape Context

The shape context of each sample point is represented by a histogram which contains information about the distance from the sample point to all other points in the contour. The histogram can be viewed as a set of vectors from the sample point to all the points in the contour. The orientation (absolute or relative) of the coordinate system to an axis does not affect the results produced when calculating the shape context [6] [26]. Thus, when calculating the shape context for each point, the only thing which is relevant is the distance between the point and the rest of the points. Since points that are closer to the sample point have higher weight, the relative distance is captured through a log-polar coordinate system [26].

The histogram for the shape context has 12 equally spaced angle bins and 5 equally spaced log-radius bins following [6]. The distance and location of each point to the sample point are calculated and the point is then counted in the appropriate bin. Once the distance and location of all points in the contour relative to the sample point are calculated, the histogram is normalized to make it invariant to the number of points available in the contour. Fig 16 shows the bins of a histogram and an example of a histogram.

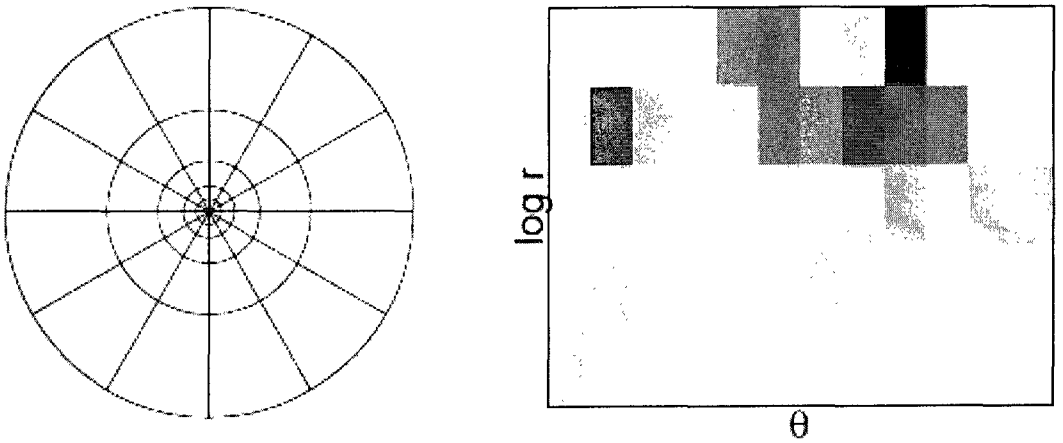

Figure 16: The bins of a histogram on the left and an example of a shape context histogram on the right (Image from [5]).

4.3 Hungarian Method

Before we go into detail of how we find the best match, we provide a brief overview of the Hungarian method. The Hungarian method provides a solution to the assignment problem. Let G be a weighted bipartite graph with partite sets U and V. Then the Hungarian method finds a correspondence between the vertices in U and the vertices in V with maximum weight.

For our original implementation of the shape context matching algorithm, we decided that since matrix computations are expensive in Python, an implementation in C would be more efficient. We used a C implementation of the Hungarian method done by Brian Gerkey from the SRQ Artificial Intelligence Center [12].

4.3.1 Finding the Best Match

After the shape context has been calculated for each sample point from the contour, the whole image has to be compared against the available images. Belongie et al. choose the images to store of an object according to which images show specific characteristics of the object [7]. However, we wanted to be able to reduce the number of images that are stored, so instead we captured the objects as we found them in the real world [26]. While Belongie et al. [7] use 72 views per object for 3D objects (that is one view every 5 degrees), we chose instead to store at most 2 views per object [26]. This decreases the accuracy of the object recognition algorithm. However, since our implementation is scale invariant and illumination invariant, we don't expect a significant decrease in accuracy.

Finally, the natural approach to finding the closeness between the images is to use a statistical approach. Belongie et al. [7] use the χ^2 statistic to calculate the difference between each of the bins of the histogram [26].

Given two images I_1 and I_2 and point i from the sample countour points of I_1 and point j from the sample contour points of I_2, we let $g(k)$ be the k-bin histogram of i and $h(k)$ be the k-bin histogram of j. Then distance, or the cost of matching i to j is given by [6]:

$$C_{i,j} = \frac{1}{2}\sum_{k=1}^{K}\frac{|g(k) - h(k)|^2}{g(k) + h(k)}$$

We can then create a matrix M where the entry in the i^{th} column and j^{th} row is $C_{i,j}$. We require M to be square in order to be able to execute the Hungarian method on M, so if one of the two images has fewer sample points, we add "dummy" points of small cost to the matrix as described previously.

Now the matrix M can be used to find the minimum cost of matching I_1 to I_2 by finding the minimum of $\sum_i C_{i,\pi(i)}$ where $\pi(i)$ is the unique point in I_2 assigned to i [6]. Note that the matching $\pi(i)$ is one-to-one. The minimum cost of matching I_1 to I_2 is $\min\sum_i C_{i,\pi(i)}$. This is, in fact, a weighted bipartite matching problem, so we can apply the Hungarian algorithm to find the minimum cost C_S of matching I_1 to I_2.

Let I be the latest image captured by the camera. Then we can use the shape context algorithm to find the object that is most likely in image I based on the images in the database. For each image J in the database, we calculate the cost C_S of matching image J to image I. One of the images in the database will have minimum cost. Then the object with minimum cost is assumed to be the match of the object in image I.

In our implementation, we keep three kinds of data in the database: an inventory of all available images, an inventory of the objects associated with each image and image files. In order to improve the performance of the shape context matching algorithm, we don't store raw data for each image. Instead, we store a pre-calculated set of histograms for a set of sample points from the contour of the original image. Thus we only need to find the shape context of the new image that we want to use for recognition and not for all the images in the database. In most cases, this means that we are not only executing fewer calculations, but also storing less data, so our algorithm is both more time- and more space-efficient.

4.4 Results

We tested the performance of our implementation of the shape context matching algorithm using images of 11 different everyday objects and 10 random images [26]. We added the 10 random images to our database in order to be able to better observe the behavior of the algorithm. The objects that we used are given in Figure 17.

Image number	*Image description*
1 - 10	random
11	doorstop
12	floor box
13	awesomebot
14	glasses
15	pencil
16	mouse (on a dark background)
17	book
18	straight A
19	curved A
20	suction cup
21	remote control

Figure 17: A list of the images in the database used for our experiment [26]

After we took the images for the database, we asked the robot to recognize each object. For each object we had 5 tests - twice in the same orientation as the image stored in the database, once zoomed out and twice with a rotation. We only stored one image per object in the database.

The robot had a 70.91% overall success rate of recognizing the objects in the database, and when the object was kept in the original orientation the success rate was more than 90%. We found that the algorithm performed more successfully when there was high contrast between the object and its surroundings. For example, a pair of glasses on a dark coloured floor were not easily recognized. On the other hand, objects with clear contours and with internal contours had a high success rate. Figure 18 has the results of our experiment.

Object	Trial 1	Trial 2	Trial 3	Trial 4	Trial 5
Actual	*Same orient.*	*Same orient.*	*Zoomed out*	*Turned*	*Turned*
Doorstep	Doorstep	Doorstep	Doorstep	Mouse	Random7
Floor Box	Floor Box	Floor Box	Floor Box	Floor Box	Floor Box
AwesomeBot	AwesomeBot	AwesomeBot	AwesomeBot	AwesomeBot	AwesomeBot
Glasses	Book	Glasses	Random5	Glasses	Book
Pencil	Pencil	Pencil	Pencil	Pencil	Pencil
Mouse	Mouse	Remote	Mouse	Remote	Mouse
Book	Book	Book	Book	Glasses	Book
Straight A	Curved A	Straight A	Straight A	Suction Cup	Straight A
Curved A	Curved A	Curved A	Random 6	Curved A	Straight A
Suction Cup	Suction Cup	Suction Cup	Suction Cup	Suction Cup	Suction Cup
Remote	Pencil	Pencil	Pencil	Random6	Floor Box

Figure 18: The results from our experiment[26].

4.5 Shape Context for Localization

Since we plan to use the shape context matching algorithm for recognizing features in the environment (such as walls), we can expect that the images that we will be working with will have clear contours (such as between a wall and the floor). To improve the performance of the algorithm we also plan to increase the size of the database to include multiple images for each robot location. This, however, limits the number of locations we can store in the database, since too large a database would slow down the recognition considerably.

5 MCL with Shape Context

5.1 The New Algorithm

To improve the performance of our implementation of MCL we use the shape context algorithm to better evaluate the probability of each sample. The changed MCL algorithm that we use is shown in Fig. 19.

There are two main changes to our original implementation of MCL. The first change is that now we rely on data from the robot's camera as well as sonars. We maintain a database of shape contexts for images that correspond to specific locations in the environment and when we get the data from the robot's camera, we determine which locations most likely correspond to the new data. This adds an extra step to the algorithm.

Once we have determined the k most likely locations using the camera data, we proceed to execute MCL as before. After we calculate the probability of a sample based on the sonar data, we use the data from the first step of the algorithm to determine if the sample's location is in one of the most likely areas. If the sample is in one of the likely areas, we set its probability to the one calculated based on the sonar data. If the sample is not in one of the likely areas, however, we scale its probability by a scale factor sf, where $0 < sf < 1$. Then samples that are not in the most likely areas will have a lower probability.

The extra data that the robot gets through its camera allows for a more precise algorithm since it can further narrow down the possible locations for the robot. The algorithm is thus better adapted for global localization than the original algorithm.

5.2 Implementation

There were several steps to the implementation of the new algorithm. We must get data from the robot's camera, split the map into areas and maintain a database of images. We also adapted our implementation of the shape context algorithm: we created two different utilities based on the shape context algorithm. The first runs as part of the localization algorithm. The second utility is used only to add images to the database and is executed separately. Finally, we had to pass the

```
MONTE CARLO LOCALIZATION ALGORITHM WITH SHAPE CONTEXT
generate the set S of samples s =< l, p > (S has size n)
let sf be a constant scaling factor, where 0 < sf < 1
while(true):
    get new sensor data s_T, new motion data m_T and new camera data c_T
    for each image im in database:
        let c_i be the cost of matching im to c_T
    let images be the set of k most likely images from the database
    let loc be the set of most likely location
    for each image im in images
        determine loc_i where loc_i is the corresponding location
        add loc_i to loc
    for each sample s =< l, p >∈ S:
        calculate l' using l and m_T
        set s =< l', 1 >
        calculate p' using l' and s_T
        if l' not in loc
            let p' = sf * p'
        set s =< l', p' >
    end for
    for i = 0 to i = n − 1
        set s_i to a randomly drawn sample from S
    end for
    calculate the standard deviation and mean position of the sample S
    if s_T differs from the expected sensor data for the current position
        update the robot's belief of its position
    end if
end while
```

Figure 19: Pseudocode for the Monte Carlo Localization algorithm with Shape Context

data to the localization algorithm. We also made a small change to the localization device - now when new data is provided to the device, the latest list of most likely areas is included with that data.

5.2.1 The Camera Device

The original implementation of the shape context algorithm ran directly on the Pioneer 2 robot, so we could easily use the built in drivers for the robot's camera in Pyrobot. However, since the MCL algorithm is expensive, we ran it remotely on a machine with a better processor than the Pioneer 2. This presented a problem since the camera drivers provided with Pyrobot do not support remote connections. Thus the first step in our implementation was creating a fake camera device which can get images from the camera on the robot and allow the localization code to access them on the remote machine.

We used Prof. Fox's implementation of a server-client pair of scripts. A server script which takes

images and stores them runs on the robot. On the remote machine, a different script queries for new images and transfers them. This transferred the images to the remote machine. However, we rely on the filters provided for cameras in Pyrobot for the shape context algorithm. Thus we needed to be able to load the images to a camera device on the robot. We modified the original code to a device (similar to the localization device), which queries the server for an image and then adds to a fake camera (provided with Pyrobot as the FakeCamera class [22]) on the robot. The fake camera can then be treated as a regular camera by the shape context code. The code for getting the data to the fake camera and then manipulating it runs in a separate thread to guarantee better performance.

5.2.2 Learning Brain and Database

To maintain a database of shape contexts for the different locations on the map, we first had to be able to add the shape contexts to the database. We modified the original shape context code to only take images, calculate the shape context and add it to the database. This code uses only the simple camera device and part of the original shape context code and it constitutes our learning brain.

We split the map of Olin-Rice into 65 distinct areas (see Fig. 20) and then took between 2 and 4 images in each area that would best represent them (such as corners, doors or other distinct features).

5.2.3 Vision Localization Device

Once the database was set up we added a new device to the robot - a vision localization device. The purpose of the vision localization device is to use the camera device and take images from the camera. It then compares to the shape contexts in the database using the shape context algorithm. The shape context algorithm also runs in a separate thread. The vision localization device then keeps a list of the k most likely areas. We decided to use 10% of the database's areas for k. Thus $k = 7$ in our implementation. The main challenge with the vision localization device was ensuring that when queried it would always have a list of areas available, so that it does not keep the querying method waiting. However, if the list was too old, the localization algorithm would skew the sample to ones closer to an old location. Thus after the list of locations is returned, we immediately reset it to an empty list of areas until new data is available. This makes sure that the list of areas returned is always the newest list available or an empty list.

5.2.4 List of Known Areas and MCL

Finally, our last change was to add a list of all known areas to MCL. Then after the evaluation step of the samples based on the sonar data in the original implementation of MCL, we add a new step. We determine if a sample is in one of the most likely areas, and if it is not, we scale its probability down. We chose a scale factor of 0.7 since the samples not in one of the most likely locations still

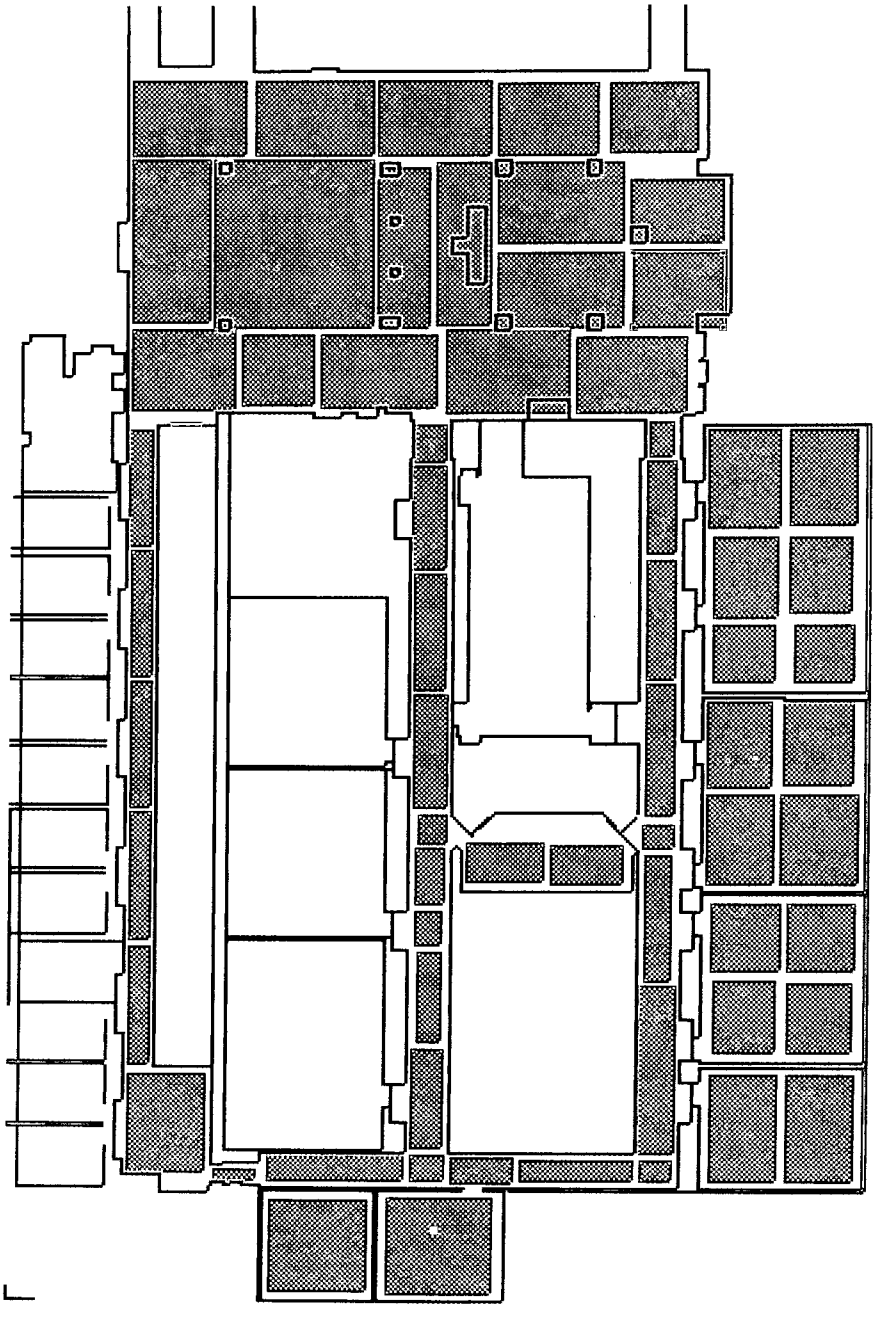

Figure 20: The map of Olin-Rice split into 65 areas.

have a high probability even after being scaled down. Since it is possible that none of the most likely locations is accurate, it was important that the samples that are not in one of them still have relatively high probability. This way during resampling the samples don't automatically converge entirely in the likely areas.

5.3 Results

We tested two scenarios for the robot localization system - tracking and global localization. When tracking we found that the performance of the localization did not improve much beyond the performance of the original implementation of MCL that did not include vision. The reason for this lack of improvement was processor performance - transfering the images from the robot to the client machine and then executing the shape context was slow due to the hardware that we had at our disposal. Therefore, the updates from the shape context algorithm were slow and several updates in the MCL algorithm would happen before any data was available from the vision part of the system. Thus the data from the shape context algorithm did not have a large impact on the results produced by the combined algorithm. The performance of the MCL with shape context was thus very similar to the performance of the MCL without shape context and therefore, tracking was not very efficient.

The performance of the algorithm under global localization was also similar to the performance of plain MCL. Updates from the shape context algorithm were still slow and did not have a big impact on the results. We did observe one difference: as the robot was moving and the updates from the vision system lagged behind, the sample points would start to converge to an area that the robot had recently occupied but had left since. Once the samples converged, the global localization problem was reduced to a tracking problem which, as we pointed out before, we did not solve exactly.

6 Future Work

The performance of the MCL algorithm with shape context was not very good. There are many improvements that can be done to the system. Hardware changes would produce immediate increase in efficiency: either better hardware for the Pioneer 2 robots, or better hardware for the client machine. One of the main problems with the current implementation of the MCL algorithm is that it relies on the sonars which are not very accurate. Using a laser sensor would make this porition of the MCL algorithm much more accurate. Better hardware for the client machine would allow storing more images in the database, as well as using more points in each image, thus producing more accurate results from the vision portion of the algorithm.

Another improvement to the MCL algorithm with vision would be adding a second camera to the robot and then using the two cameras to get 3D images of the environment to use with localization. This would result in more accurate localization by allowing for more speciffic landmarks to be used by the algorithm as well as extracting more sophisticated features (such as intersections of corridors, corners, etc).

References

[1] *Aria: ArMap Class Reference* Retrieved April, 2009 from www.ai.rug.nl/vakinformatie/pas/content/Aria-manual/classArMap.html

[2] Asimov, Isaac. 1951. *I, Robot.* New York: Gnome Press

[3] Atiya, S. and Hager, G. D. 1993. *Real-time Vision-based Robot Localization* Robotics and Automation, IEEE Transactions on, 9(6):785800

[4] Belongie, S. and Malik, J. 2000. *Matching with Shape Contexts* Proc. IEEE Workshop on Content-based Access of Image and Video Libraries

[5] Belongie, S., Malik, J., and Puzicha, J. 2001. *Matching Shapes* Proc. Eighth International Conference Computer Vision, July

[6] Belongie, S., Malik, J., and Puzicha, J. 2001. *Shape Context: A New Descriptor for Shape Matching and Object Recognition* Advances in Neural Information Processing Systemts: Proc. 2000 Conf

[7] Belongie, S., Malik, J., and Puzicha, J. 2002. *Shape Matching and Object Recognition Using Shape Contexts* IEEE Transactions On Pattern Analysis and Machine Intelligence, Vol. 24, No. 24, April

[8] Dellaert, F., Seitz, S., Thrun, S., and Thorpe, C. 2000. *Feature Correspondence: A Markov Chain Monte Carlo Approach* NIPS-00

[9] Fang, Y., Yamada, K., Ninomiya, Y., Horn, B., and Masaki, I. 2004. *Shape-Independent Method for Pedestrian Detection with Far-Infrared Images* IEEE Transactions On Vehicular Technology, Vol. 53, No. 5, Sep.

[10] Fox, D., Burgard, W. and Thrun, S. 1999. *Markov localization for mobile robots in dynamic environments* Journal of Artificial Intelligence Research, vol. 11, p.391-427

[11] Fox, Dieter, Burgard, W., Dellaert, F., and Thrun, Sebastian. 1999. *Monte Carlo Localization: Efficient Position Estimation for Mobile Robots.* In AAAI 99/IAAI 99: Proc. of the 16th National Conf. on Artificial Intelligence and the 11th Innovative Applications of Artificial Intelligence Conf., p.343-349, Menlo Park, CA

[12] Gerkey, B. *A C Implementation of the Hungarian Method* Retrieved December, 2006 from http://ai.stanford.edu/ gerkey/tools/hungarian

[13] Huttenlocher, D., Klanderman, G., and Rucklidge, W. 1993. *Comparing Images Using the Hausdorff Distance* IEEE Transactions On Pattern Analysis and Machine Intelligence, Vol. 15, No. 9, Sep.

[14] Jonker, R. and Volgenant, A. 1987. *A Shortest Augmenting Path Algorithm for Dense and Sparse Linear Assignment Problems* Computing, Vol. 38

[15] Latecki, L., Lakamper, R., and Eckhardt, U. 2000. *Shape Descriptors for Non-Rigid Shapes with a Single Closed Contour* Proc. IEEE Conf. Computer Vision and Pattern Recognition.

[16] LeCun, Y., Bottou, L., Bengio, Y.m and Haffner, P. 1998. *Gradient-Based Learning Applied to Document Recognition* Proc. IEEE, Vol. 86, No. 11, Nov.

[17] Maxwell, B. 2006. *Notes from E27/CS27 Computer Vision, F03* Retrieved December 1, 2006 from http://palantir.swarthmore.edu/maxwell/classes/e27/F03/E27_F03_Lectures.pdf

[18] *MobileRobots Research and University Robots* Retrieved April, 2009 from www.activrobots.com

[19] Papadimitriou, C. and Stieglitz, K. 1982. *Combinatorial Optimization: Algorithms and Complexity* Prentice Hall

[20] Persoon, E. and Fu, K. S. 1986. *Shape Discrimination Using Fourier Descriptors* IEEE Trans. Pattern Anal. Mach. Intell, Vol. 8, No. 3, May

[21] *Pyro, Python Robotics* Retrieved April, 2009 from www.pyrorobotics.org

[22] *Pyrobot: Fake Camera Class* Retrieved April, 2009 from http://pyrorobotics.org/usermanual/pyrobot.camera.fake.FakeCamera-class.html

[23] Russell, S. and Norvig, P. 2003. *Artificial Intelligence: A Modern Approach* Prentice Hall, 2nd edition

[24] Se, S., Lowe, D. and Little, J. 2001. *Vision-based Mobile Robot Localization and Mapping Using Scale-invariant Features.* Vol.2, p.2051-2058

[25] Shapiro, L. S. and Brady, J. M. 1992. *Feature-based Correspondence: an Eigenvector Approach* Image Vision Comput., Vol. 10, No. 5, Jun.

[26] Stamenova, S. and Handler, S. 2006. *Object Recognition Using Shape Contexts* Paper for CS 484, Macalester College

[27] Thrun, S., Fox, D., and Burgard, W. 2000. *Monte Carlo Localization with Mixture Proposal Distribution* In Proc. of the 17th National Conference on Artificial Intelligence and 12th Conference on Innovative Applications of Artificial Intelligence, pages 859-865. AAAI Press / The MIT Press

[28] Thrun, S., Fox, D., Burgard. W, and Dallaert, F. 2001. *Robust Monte Carlo Localization for Mobile Robots* Artif.Intell., 128(1-2):99-141.

[29] Welch, Greg and Bishop, Gary. 1995. *An Introduction to the Kalman Filter* University of North Carolina at Chapel Hill, Chapel Hill, NC

[30] Yuen, D. and MacDonald, B. 2005. *Vision-based Localization Algorithm Based on Landmark Matching, Triangulation, Reconstruction, and Comparison.* Robotics, IEEE Transactions on, 21(2):217-226.