

Geometric Constraint Identification and Mapping for Mobile Robots

David J. Austin

Brenan J. McCarragher *

d.austin@computer.org

brenan@faceng.anu.edu.au

Center for Autonomous Systems[†]
Royal Institute of Technology,
SE 100-44 Stockholm, Sweden.

Department of Engineering, Faculties,
Australian National University,
Canberra, ACT 0200, Australia.

October 19, 2001

Keywords: mapping, geometric model, constraint identification.

Abstract

A new method of map building for mobile robots is presented. Recent developments have focused on grid-based mapping methods which suffer from the drawback of their size, requiring a great deal of memory and prohibiting the use of many path-planning algorithms. By contrast, geometric maps provide a compact alternative which facilitates path-planning. We propose a new method which identifies geometric models of the constraints imposed upon the robot by the environment. A rigorous approach is taken to the process of constraint identification, which is cast as a minimisation problem. A number of primitive geometric objects are used for constraint modeling including line segments, arc segments, cubic segments and, for three degree of freedom systems, polygonal planar patches. A number of operations are also defined which integrate new sensor readings into the existing model. Simulation results are presented for two and three degree of freedom systems, demonstrating the effectiveness of the constraint identification process. A comparative study is also presented which gives guidelines for the proper selection of primitives and operations.

*Corresponding author

[†]The results presented in this paper were obtained while the first author was also at the Australian National University.

Nomenclature

- \mathbf{x} : position (state) vector of the robot
- \mathbf{x}_i : *constrained points*, where a constraint on the robot's motion has been observed
- x, y, θ : co-ordinate axes used to describe the state of the robot
- ϵ : the positioning uncertainty of the robot
- n : number of constrained points observed so far
- m : the number of primitives in the model
- $G(n)$: the model of the environment after n constrained points have been observed
- $prim_j$: the j^{th} geometric object or primitive used to model the constrained points
- $dist(prim_j, \mathbf{x})$: the distance from the point \mathbf{x} to the j^{th} primitive
- err_j : error function of the j^{th} primitive
- α : model complexity penalty, a parameter of the model identification process
- β : primitive length penalty, a parameter of the model identification process
- $qual_{num}$: quality measure; number of primitives in the model
- $qual_{cov}$: quality measure; percentage of the actual constraints covered by the model

1 Introduction

Mobile robotics is one of the most challenging and exciting areas of robotics today, largely due to the difficulties in developing and using maps of unstructured environments. Many researchers have studied the problem of map building for mobile robots, particularly for indoor or office environments. When developing a map which represents the physical environment, there are two basic approaches, *grid-based* and *geometric* maps. Grid-based maps typically represent the environment using a uniform grid. Each grid cell has a value which indicates probability that there is an obstacle at the corresponding position in the environment. Many researchers have used grid-based maps, including Elfes [10], Thrun *et al.* [18, 17], Borenstein [3, 15], Yamauchi *et al.* [19], and Murray and Jennings [14]. Grid-based maps are easy to

construct but it is difficult to extract information from them because of their size. The size of grid-based maps leads to high computational effort when determining control commands and to very high storage requirements. Furthermore, grid-based maps become impractical in higher dimensional systems. For example a rectangular robot, whose size is significant, is a three degree of freedom system, considering both the position and orientation of the robot. If we divide the orientation of the robot into 10° steps then a three-dimensional system requires 36 times as much memory as the same sized two-dimensional system. Even with memory available cheaply today, the memory requirements of grid-based mapping can be prohibitive, especially for higher order systems.

On the other hand, geometric approaches attempt to directly identify and model features of the environment. Geometric maps have two main advantages over grid-based approaches: firstly, they are a much more compact representation of the environment of the robot and, secondly, due to the compact representation, they are easier to use. The disadvantage of geometric maps is that they are harder to construct, requiring interpretation of the sensor data and extraction of geometric features. Crowley [8] developed a sonar mapping system which used recursive line fitting to fit a polyline to sonar measurements. McKerrow [13], Chong and Kleeman [7], and Araujo and Grupen [1] investigated sonar-based geometric mapping using primitive objects of **partial plane** and **corner**. Similarly, Cahut *et al.* [4] fit line segments to sonar data. Castellanos *et al.* [5, 6] extract **points**, **corners**, **segments** and **partial planes** from laser and vision information. Ayache and Faugeras [2] also extracted three-dimensional primitives using a trinocular vision system and Scheduling *et al.* [16] fit a polyline to laser range data. Janet *et al.* [11] used a neural network of hyper-ellipsoid clusters to group sonar readings and Kwon and Lee [12] use a similar stochastic clustering technique. Finally, Delahoche *et al.* [9] use an omni-directional vision system to extract the vertical lines that are formed by corners and doors in an office environment. Most research using geometric maps has concentrated on the sensor models and the relationship between the sensor and the geometric primitives used. Previous works [4, 5, 6, 7, 9, 11, 12, 16] use *ad hoc* rule-based or heuristic approaches to map updates and so provide few assurances about the quality of the resulting map. There has also been little study of the

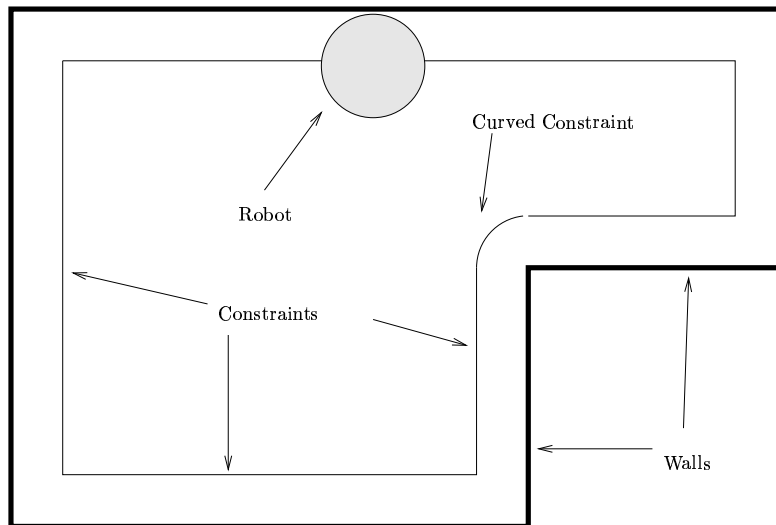


Figure 1: *Example of constraints imposed upon the robot due to a simple two-dimensional environment.*

appropriate geometric primitives to use. This paper will study a number of different geometric primitives and develops a rigorous process for fitting the primitives to data. A comparative study is also given which gives a guide for the selection of an appropriate set of primitives for any particular environment.

Almost all research to date has focused on identifying the obstacles in the environment, rather than the constraints that the obstacles impose upon the robot. In typical office environments, we cannot neglect the size of the robot and so there is a difference between the locations of the obstacle and the constraint. Figure 1 illustrates the difference between the obstacles (walls) and constraints due to the radius of the robot. Note that it is possible to determine the constraints using an accurate model of the robot and measurements of the obstacles, but it is simpler to identify the constraints directly. When navigating through the environment, the constraints, rather than the obstacles are of critical importance. If the robot becomes constrained, it is often advantageous to follow the constraint, using the constraint as a guide. In order to follow constraints the controller of the robot must have a good model of the constraint. Therefore, we choose to identify the constraints in this paper. The constraints are identified by observing points at which the motion of the robot is constrained and then constructing geometric objects to fit these points. Note that an office environment consisting of only planar walls, such as that shown in Figure 1, can induce curved constraints due to the radius of the robot. This is an important consideration when selecting

primitive geometric objects for mapping.

In this paper, we present a rigorous map building method which identifies geometric models of the constraints on the robot. The assumptions that we make are outlined in the next section. Section 3 presents the framework that we use for rigorous identification of the constraints and Sections 4 and 5 present applications of the identification process to two and three degree of freedom systems, respectively. In Section 4, we study primitive selection and the methods for assigning data points to primitives. Section 5 studies a three degree of freedom system and its inherent challenges.

2 Assumptions

For this paper we will consider a mobile robot moving about in the plane. Initially, we will consider a circular robot whose position can be completely described by its location $\langle x, y \rangle$ in the plane. In Section 5, we will consider a non-circular robot whose position is described by the triple $\langle x, y, \theta \rangle$, where θ is the orientation of the robot in some fixed reference frame.

The first major assumption that we make is that the robot can determine its position in a fixed reference frame with bounded uncertainty. We also require that positioning errors remain bounded over time and so we cannot rely on pure odometry to determine the position of the robot. However, either a beacon system or one of many self-localisation algorithms could be employed. We denote the bound on the robot positioning error ϵ . If the positioning errors do grow unboundedly then the robot will lose track of its position and it is not possible to develop a meaningful map. In the future we will investigate schemes which do not require accurate positional information but for now we will assume that the position of the robot is known so that we can investigate the constraint identification process.

In order to identify the constraints in the environment, we must be able to detect when the robot is constrained. We assume that the robot is equipped with an appropriate sensor to detect constraint and that the sensor is reliable. Almost all mobile robots are equipped with bumpers or sonar sensors which can provide this information. Note that the robot does not necessarily have to make contact with the

environment for constraint. In many cases it is undesirable to have the robot in contact with the walls and so it may be required that the robot maintain a certain minimum distance from the walls. This minimum distance then imposes the constraints upon the robot. Note that some form of range sensor is then required to measure the distance to the obstacles in the environment. In this paper we will assume that the robot makes contact with the walls and other obstacles and that a bumper is used to give a binary indication of constraint. However, a range sensor could be used with minimum distance requirement without changing the results presented here.

For this paper we will concentrate on the identification of the constraints in the environment and ignore the details involved in moving the robot around in the environment. We will merely assume that there is some controller moving the robot around the environment but make no further assumptions about the controller. Clearly it would be desirable to have the controller make use of the map information and explore areas of the map which are not well known. However, in this paper we will focus on the identification process.

3 Constraint Identification Framework

As the robot moves through the environment it will occasionally become constrained by obstacles. The points at which the motion of the robot becomes constrained are of particular interest here and we call them *constrained points*. We keep a record of the constrained points and denote them \mathbf{x}_i , $i = 1 \dots n$, where n is the number of constrained points observed so far. The process of constraint identification involves developing a model of the constraints from the observed constrained points. Here, a rigorous approach is taken to the process of constraint identification, which is cast as a minimisation problem. Geometric objects called *primitives* are used to model the constraints. A number of primitive geometric objects are used including line segments, arc segments, cubic segments and polygonal planar patches for three degree of freedom systems. Operations are then defined which integrate new sensor readings into the existing model. When a new sensor reading is taken, each of the operations is tried and the one which minimises

the error function is selected.

We will define the model of the constraints in the environment as a set of m primitives, each of which represents part or all of a constraint that exists in the environment. We will denote the model after n constrained points have been observed as $G(n)$

$$G(n) = \{prim_j : j = 1 \dots m\} \quad (1)$$

where $prim_j$ is the j^{th} primitive. Note that the number of primitives is less than or equal to the number of constrained points observed ($m \leq n$). Each of the constrained points is assigned to one of the primitives and the parameters of each primitive are chosen to best represent the constrained points that are assigned to the primitive. Note that the set $G(n)$ is assumed to be initially incomplete and may even be empty.

3.1 Primitives

The primitives are geometric objects that model the constraints on the motion of the robot. Each primitive object has a number of parameters that describe its size and shape as well as a distance function and an error function. For example, for a line segment primitive the shape is described simply by the coordinates of the two end-points and the distance function gives the minimum distance to any point on the line segment. The error function err_j of each primitive quantifies the quality of fit of the primitive to the set of points which is being approximated. We will use the error function as part of the identification process by assigning constrained points so as to minimise the error functions. For the line segment primitive the error function returns the sum of the distances from each of the assigned constrained points to the line segment. We denote the distance function of the j^{th} primitive $dist(prim_j, \mathbf{x})$ and the error function err_j .

The model predicts that the robot is constrained whenever

$$dist(prim_j, \mathbf{x}) < \epsilon \quad (2)$$

where \mathbf{x} is the position vector of the robot and ϵ is the positioning error of the robot. Note that the form of (2) is completely general and may use any primitive to model any constraint upon a robot whose state is described by the vector \mathbf{x} . The error function for each primitive depends on the geometric shape in question but the form of the error function is:

$$err_j = \sum_{i=1}^{n_j} [dist(prim_j, \mathbf{x}_i)] + \frac{\beta length}{n_j} \quad (3)$$

where n_j is the number of constrained points assigned to the j^{th} primitive, $dist(prim_j, \mathbf{x}_i)$ is the distance from the primitive to the i^{th} constrained point, $length$ is the length of the geometric object and β is a scaling constant. The first term of equation (3) measures the quality of fit of the geometric shape to the constrained points and the second term measures the confidence that we have about the shape. The constant β is used to adjust the relative importance of the two terms.

3.2 Model Update Process

We will cast the identification process as an optimisation problem. This allows us to take a rigorous, objective approach and also permits rapid experimentation or tuning of the parameters of the identification process. We will assume, without loss of generality, that the error function is non-negative and that we wish to minimise the error. Therefore, to find the model $G(n)$, we wish to minimise the following error function:

$$f_{err}(G(n)) = \sum_{j=1}^m err_j() + \alpha m \quad (4)$$

where err_j is the error function for the j^{th} primitive and α is a positive constant. The second term of equation (4) is included to promote simplification of the model and will be discussed further in Section 3.3 below.

After the observation of the n^{th} constrained point, we must determine an improved model $G(n)$ which encompasses the new data. When using an incremental update there are many possible approaches to the

problem. If we blindly attempt a brute-force approach we soon run into difficulties. One such approach might be to try all possible assignments of constrained points to primitives. However, for n constrained points we can have from 1 to n primitives and the number of possible combinations is given by:

$$n_{comb} = \sum_{i=1}^n i^{(n-i)} \quad (5)$$

Even for a tiny experiment of 20 constrained points there are a staggering 3.23×10^{11} possible combinations. Clearly, we cannot try exhaustive, brute-force approaches. Instead, we consider a small subset of map update steps called *operations*.

The incremental updates of the model are defined by a set of operations. The operations are the possible actions that can be performed to integrate each new observation \mathbf{x}_n into the model. Each operation is a function which, given the existing model $G(n-1)$ and the latest observation \mathbf{x}_n , determines a new model. Mathematically, we write:

$$G_k(n) = oper_k(G(n-1), \mathbf{x}_n) \quad (6)$$

where $oper_k$ is the k^{th} operation. Once we have observed the n^{th} constrained point, each of the available operations is tried and the model which minimises the error of equation (4) is selected:

$$G(n) = \arg \min_k (ferr(G_k(n))) \quad (7)$$

For example, a two-dimensional system could utilise line segment primitives and the operations might be “create new line primitive, “join to closest primitive and so on. Note that the set of available operations depends upon the primitives selected.

3.3 Model Simplification

Recall that equation (4) consisted of two terms. The first term was the sum of the error functions of the primitives and is clearly required for the constraint identification process. However, the requirement for the

second term is less obvious. The second term is included to penalise models which involve a greater number of primitives. Thus, the system will move towards simpler models with fewer primitives. The inclusion of the second term of equation (4) is necessary to prevent the system identifying the most complex model which uses one primitive to model each constrained point. Such a model would minimise the sum of the error functions but is not desirable. The constant α allows adjustment of the trade-offs made between model complexity and accuracy. Selection of small values for the constant α will result in a more complex model while selection of higher values of α will cause excessive joining of primitives, at the expense of higher fitting errors. Clearly, it is better to err on the side of smaller values of α and have a more complex but accurate model. During simulations, it was found that appropriate values of α lie in the range:

$$0.1 \times \epsilon \leq \alpha \leq \epsilon \tag{8}$$

where ϵ is the positional uncertainty for the robot. We can motivate the upper bound on α intuitively if we consider adding a single new constrained point which is at a distance greater than ϵ from an existing primitive. The identification process has two options: it could add the point to the existing primitive or it could form a new primitive. We can see from equation (3) that if the new point is added to the existing primitive then the error function will increase by at least the distance between the point and the primitive which is at least ϵ in this case. However, if the identification process adds a new primitive then the increase in error is α (see equation (4)). If the new point is at a distance greater than the positioning error ϵ from the existing primitive then it is clearly part of a separate primitive and the identification process should add a new primitive. We can see that $\alpha < \epsilon$ is required for this outcome. The lower bound on α has been established by simulation.

3.4 Quality of the Model

To measure the performance of the identification process, we must introduce some measures of how “good” a model of the constraints in the environment is. We introduce two measures of model quality, as follows.

Clearly, we want the model to be complete, representing all of the actual constraints. Additionally, we would like the model to be as simple as possible with the minimum number of primitive geometric objects.

The coverage of the actual constraints is measured by taking a uniform sample of points s_i along each of the actual constraints. The percentage of the actual constraints covered is then estimated by determining how many of the sample points lie within the measurement error of a geometric primitive. Formally, each of the sample points is said to be “covered” if

$$\min_j \text{dist}(\text{prim}_j, s_i) < \epsilon \quad (9)$$

The percentage coverage is then given by

$$\text{qual}_{cov} = 100 \times \frac{n_{cov}}{n_{sample}} \quad (10)$$

where n_{cov} is the number of sample points covered according to equation (9) and n_{sample} is the number of sample points.

The second quality metric is given by the number of primitives in the model:

$$\text{qual}_{num} = m \quad (11)$$

4 Application 1: Two-Dimensional Identification

As a first case for constraint identification, we will consider a circular robot moving in the plane. This is a two degree of freedom system as the x and y position of the robot are sufficient to describe its location.

The state of the robot is described by:

$$\mathbf{x} = [x, y] \quad (12)$$

Note that there is no limitation on the number of dimensions that we could consider using geometric methods. However, higher dimensional systems require more constrained points for identification and are harder to analyse and present. We will consider a three degree of freedom system in Section 5 below.

4.1 Primitives

Intuitively, it seems clear that the primitives selected should match the form of the constraints found in the environment. For the two dimensional case, the first, obvious primitive to consider is a line segment, which can be used to model the office walls. Also, note that there can be curved constraints even in environments made up of linear segments. The curved constraints arise from the non-zero radius of the robot and the convex corners formed by two walls, as shown in Figure 1. Line segments can be used to approximate the curved constraints but a number of line segments are required and there will be a higher modeling error than if a curved object was used. We will also introduce an arc segment primitive for comparison. The final primitive that we consider is a cubic segment, to see if higher order functions can be used effectively. We will use combinations of these three primitive functions to study the sensitivity of the modeling process to primitive selection.

4.1.1 Line Segment Primitive

The line segment primitive is constructed by fitting a line through the assigned set of constrained points. Each of the constrained points is then projected onto the line and the pair of projected points which are furthest apart are selected as the end-points of the line segment. The line segment primitive has a normal which describes which side of the line is constrained. The normal is used to detect when the robot has traveled “through” the primitive. If a line segment primitive is erroneously joined across a passageway then it is possible for the robot to move through the primitive. If this is detected then the erroneously joined primitive can be split in two.

The error function for the line primitive is:

$$err_{line} = \sum_{i=1}^{n_j} dist(prim_j, \mathbf{x}_i) + \frac{\beta length}{n_j} \quad (13)$$

where $dist(prim_j, \mathbf{x}_i)$ is the distance from the constrained point \mathbf{x}_i to the line segment, $length$ is the length of the line segment, and β is a positive scaling constant. The first term of equation (13) is the obvious measure of the quality of fit of the line segment to the points. The second term is included to discourage spurious joining of distant constrained points. During simulations, it was found that values of β given by:

$$\beta \approx 50 \times \alpha \quad (14)$$

avoided spurious joining of distant points, where α is the constant used in equation (4) above to encourage joining of primitives.

4.1.2 Arc Segment Primitive

The arc segment is a chord of a circle. The primitive is constructed by finding the two points which are furthest apart and connecting these by a line. Next, the point which is furthest from the line is found. A circle is then constructed through these three points. The extents of the arc are determined by sorting the constrained points according to their angles from the centre. It is then assumed that the largest gap between two adjacent constrained points is the incomplete section of the arc. The arc primitive also has a flag which indicates if the robot is constrained on the inside or outside of the arc. This flag is used to determine if the robot has traveled through the primitive, indicating a modeling error, as discussed above.

The error function for the arc primitive is the same as that used for the line segment primitive in equation (13) except that the distance function now returns the radial distance from the constrained point to the arc and $length$ is the length of the arc segment.

4.1.3 Cubic Segment Primitive

The cubic segment is a piece of a cubic of the form:

$$f(x) = ax^3 + bx^2 + cx + d \quad (15)$$

where a , b , c , and d are the constant parameters of the cubic. The primitive is constructed by fitting the cubic to the data by solving the system below for a , b , c , and d .

$$\begin{bmatrix} \sum x^3 & \sum x^2 & \sum x & n_j \\ \sum x^4 & \sum x^3 & \sum x^2 & \sum x \\ \sum x^5 & \sum x^4 & \sum x^3 & \sum x^2 \\ \sum x^6 & \sum x^5 & \sum x^4 & \sum x^3 \end{bmatrix} \begin{bmatrix} a \\ b \\ c \\ d \end{bmatrix} = \begin{bmatrix} \sum y \\ \sum xy \\ \sum x^2y \\ \sum x^3y \end{bmatrix} \quad (16)$$

The extents of the cubic segment are determined by finding the two constrained points that are farthest apart.

The error function for the cubic segment primitive is the same as that used for the line segment primitive in equation (13) except that the distance function now returns the distance from the constrained point to the closest point on the cubic segment and *length* is the length of the segment. For the cubic segment, the distance function is approximated by choosing 100 sample points on the cubic and finding the minimum distance to the sample points. The length function is also approximated by dividing the cubic into 20 segments and computing the straight-line length of each of these.

4.2 Operations

There is a wide range of possible operations that one could consider for both line and arc primitives and, clearly, the selection of an appropriate set of operations is important for the successful and rapid identification of the model. The choice of the set of operations may not be clear but we can use intuitive

arguments to propose some basic operations and in Section 4.2.4, we will investigate the performance of various sets of operations.

Clearly, we require an operation which adds a new primitive to the model, using the new constrained point. Next, we require an operation which adds the new constrained point to an existing primitive. The first operation will tend to create multiple model primitives for a single actual primitive. Therefore, we require an operation which merges two model primitives together. We assume that the constrained points are reliably detected and so there will be no need to remove constrained points. However, it is possible that a set of constrained points could be wrongly assigned to one primitive, called false joining. In such situations, it is necessary to split the falsely joined primitive and so the final operation that we will consider is a primitive split.

4.2.1 Adding a New Constraint

The first operation that we will define is the “add new line segment” operation which adds a new line segment primitive to the model, using the new observed point \mathbf{x}_n . The new line segment primitive is created as a single point and the normal is estimated from the previous motion of the robot and the sensor information. Similarly, we define operations, called “add new arc” and “add new cubic” which add new arc and cubic primitives, using \mathbf{x}_n .

4.2.2 Extending Constraint

Having defined operations to add new primitives, we need operations to extend or augment existing primitives. The new point \mathbf{x}_n is added to the list of constrained points for the existing primitive and the geometric properties of the primitive are re-computed. It is possible to consider adding the observed point \mathbf{x}_n to each of the existing primitives. However, this is too computationally expensive and leads to much wasted effort. Instead, we propose to consider operations which add the observed point \mathbf{x}_n to the closest, second closest and third closest primitives. It seems clear that constrained points will almost always be added to existing primitives that are close by and so we only need consider operations to join to nearby

primitives. Thus, we have three more operations to join to the closest, second closest and third closest primitive.

4.2.3 Joining Two Constraints

The set of operations presented so far will tend to create multiple model primitives for a single actual primitive. Therefore, we require an operation which merges two model primitives together. There are two merging operations, the first, “merge line” creates a new line segment primitive from \mathbf{x}_n and the points approximated by the closest and second closest primitives and the third, “merge arc” and “merge cubic” create arc or cubic primitives similarly. Both of these operations merge the closest and second closest primitives and the new constrained point \mathbf{x}_n . The closest and second closest primitives are replaced by the new, larger primitive. In addition, we define three further operations which merge the closest three primitives with the new constrained point \mathbf{x}_n to form line, arc and cubic primitives. Note that the operations make no assumptions about the primitives that existed prior to the merge, the new primitive is depends only on the combined set of constrained points. Therefore, “merge line with line” is the same operation as a “merge line with arc”.

Note that these merging operation may not always be appropriate. For example, the closest and second closest primitives may be line segments that are not collinear. The application of “merge line” will result in a poorer model. This is detected by the minimisation process using the error functions. The error function of the new line segment added by “merge line” will have a high value, indicating the poor fit of the line to the constrained points. This high value will cause the minimisation to select an alternate operation, rejecting the merger of two non-collinear line segments.

4.2.4 Splitting Constraints

Having described operations for the creation and aggregation of model primitives, we now propose some operations for the subdivision of primitives. Splitting of primitives is necessary because it allows us to recover from erroneous joining of primitives. If we use a joining operation, then it is inevitable that

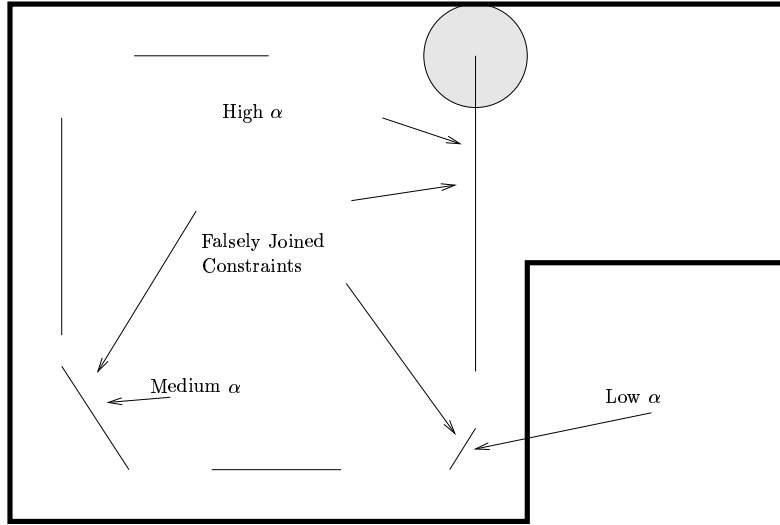


Figure 2: *Example of false joining of primitives. High values of α can lead to joining of primitives across passageways and medium values cause unacceptable false joining at corners but low values of α will still cause some false joining at corners.*

false joining of primitives will happen occasionally. For example, if the joining threshold α (see equation (4)) is greater than zero, then there will be cases where a join takes place around a corner, as shown in Figure 2. Also, we wish to use higher values for α to permit aggressive simplification of the model by joining primitives. A high value of α may result in more serious errors such as joining across a passageway, as shown in Figure 2. We are able to use a much higher value for α if we can correct the resulting mistakes.

We propose three splitting operations “split to lines” which splits a primitive into two line primitives, “split to arcs” which splits to two arc primitives, and “split to cubics” which splits to two cubic primitives. The split is conducted at the point towards the middle of the existing primitive which has maximum deviation from the existing primitive. Split operations are also forced if the robot moves through a modelled primitive.

A simulator for the robot and constraint identification was implemented with all of the above primitives and operations. The robot motion is simulated in steps of $0.01m$ and, hence, the robot positioning error is $\epsilon = 0.01m$. Two different environments were used for the simulations, shown in Figure 3 and Figure 4. The robot has a diameter of $1m$ and the environment of Figure 3 is $32m$ by $23m$ whilst the environment of Figure 4 is approximately $85m$ long. To measure the coverage of the actual constraints, a set of 595 sample

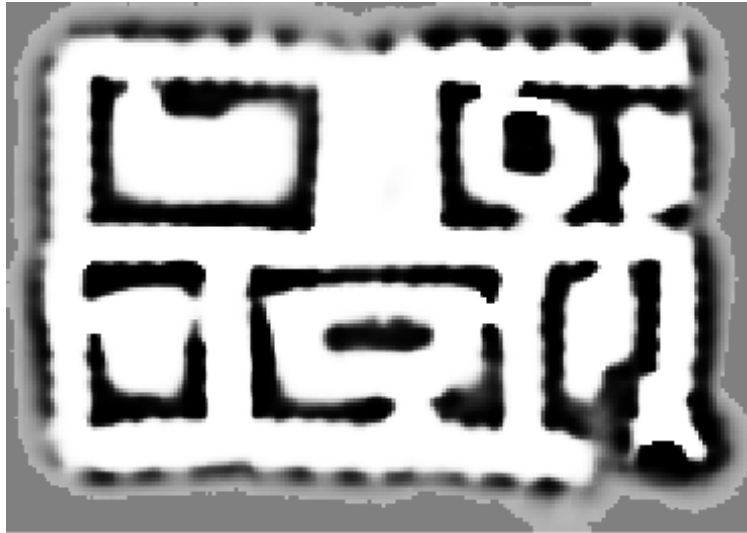
points was taken at uniform spacings along the constraints of Figure 3. Similarly, 509 sample points were used for the environment of Figure 4. For these results, we used values of $\alpha = 0.002$ and $\beta = 0.1$.

Figures 3(b) and 4(b) show the geometric models (primitives) of the constraints that were identified for the two different environments using the techniques presented above. The identified map shown in Figure 3 has 197 primitives and covers 99.3% of the sample points. The second identified map, shown in Figure 4, has 125 primitives and covers 98.6% of the sample points. Note that, for our implementation, each line segment primitive requires 48 bytes of storage, each arc 56 bytes and each cubic 72 bytes. Hence, the two maps presented in Figures 3(b) and 4(b) use less than 14k and less than 9k respectively. This illustrates the compact nature of geometric mapping. These two maps show that the constraint identification process can successfully determine a compact geometric map for complex, practical environments.

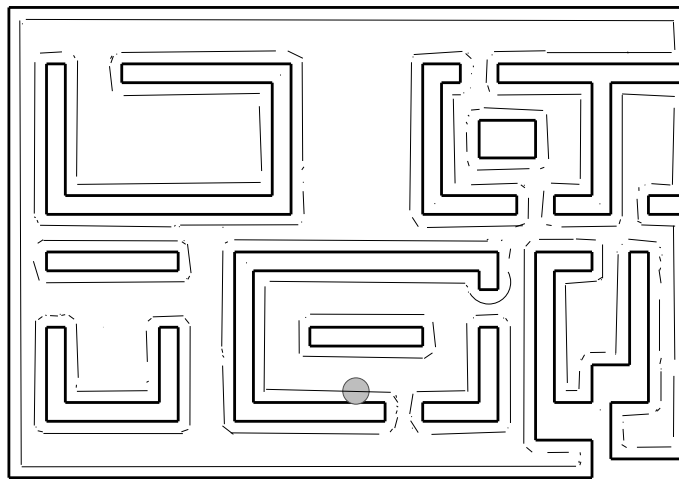
Figures 3(b) and 4(b) also show the difference between the obstacles in the environments and the constraints upon the motion of the robot. In Figure 3(b) we can see the curved constraints that result from the radius of the robot and Figure 4(b) shows the complex constraints that can occur when a number of simple obstacles are in close proximity. The two smaller circular objects in the left room of Figure 4 illustrate this. The robot can just fit between the upper small circle and the adjacent obstacle whilst the robot cannot fit between the lower circle and its adjacent obstacle. This environment illustrates some of the difficulties in determining the constraints from the obstacles in general environments and further motivates the direct identification of the constraints.

4.2.5 Primitive Selection

Of particular interest here is how the selection of the primitives and operations affects the performance of the identification process. First, we will consider the effect of the choice of primitives when identifying a model of the map shown in Figure 3(a). We have developed three different primitives: line, arc and cubic segments. Figure 5 shows the number of objects in the model and the percentage of the map covered for three different runs: lines only, lines and arcs, and lines, arcs and cubic primitives. To better compare the

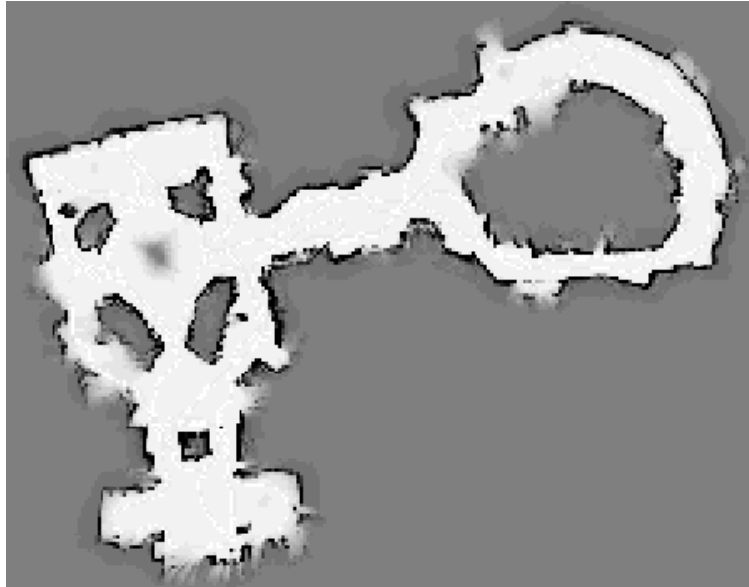


(a)

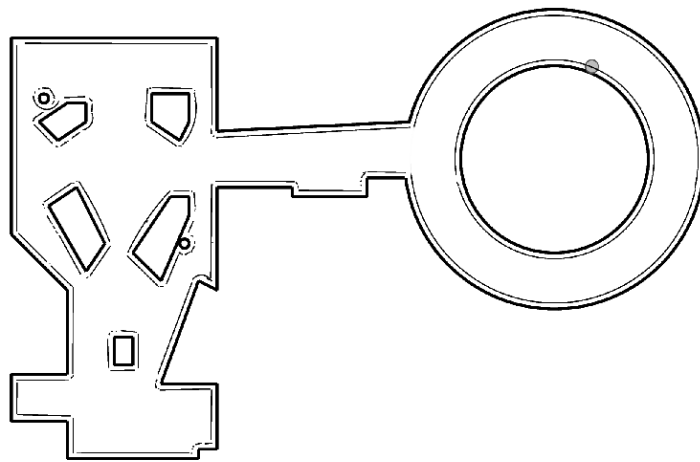


(b)

Figure 3: *Environment used for the 1994 AAAI autonomous mobile robot competition (approximately 32m by 23m). (a) Grid-based map identified by Thrun et al. [17] and (b) geometric map identified from 4000 random constrained points.*



(a)



(b)

Figure 4: *Map of the National Museum of American History (approximately 85m long). (a) Grid-based map identified by Thrun et al. [18] and (b) geometric map identified from 3500 random constrained points.*

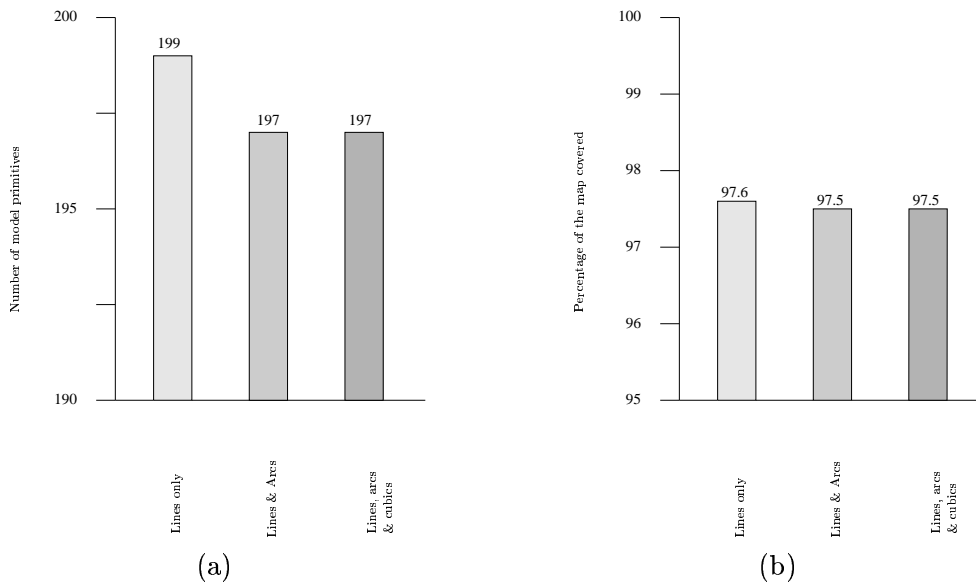


Figure 5: Comparison of (a) the number of primitives and (b) the percentage coverage of the map for the three primitive sets for the environment of Figure 3.

three cases, a set of constrained points was recorded and replayed for each of the runs. As can be seen from Figure 5, the different primitive sets make little difference for the environment of Figure 3. This is due to the small number and low significance of the curved constraints in the environment of Figure 3. Note however, the inclusion of the arc primitive does slightly decrease the number of primitives required for the model.

To provide a better idea of the role of the primitive in the identification process, we also made the comparison with the environment of Figure 4, which has a number of more significant curved constraints. The results of the identification for the three primitive sets are shown in Figure 6. For this environment, we see the expected result: including the arc primitive improves the resulting model by representing the curved constraints as a single arc and, hence, dramatically reducing the number of primitives in the model. Addition of the cubic primitive makes little difference to the number of primitives or the coverage of the model. This result indicates that there is no requirement for a cubic primitive in either of the environments.

From the comparisons shown in Figures 5 and 6, we can see that the resulting model is considerably simpler if the geometric objects selected for the primitives are of similar order to the constraints in the environment well and that primitives of higher order than the environment will provide little benefit. For

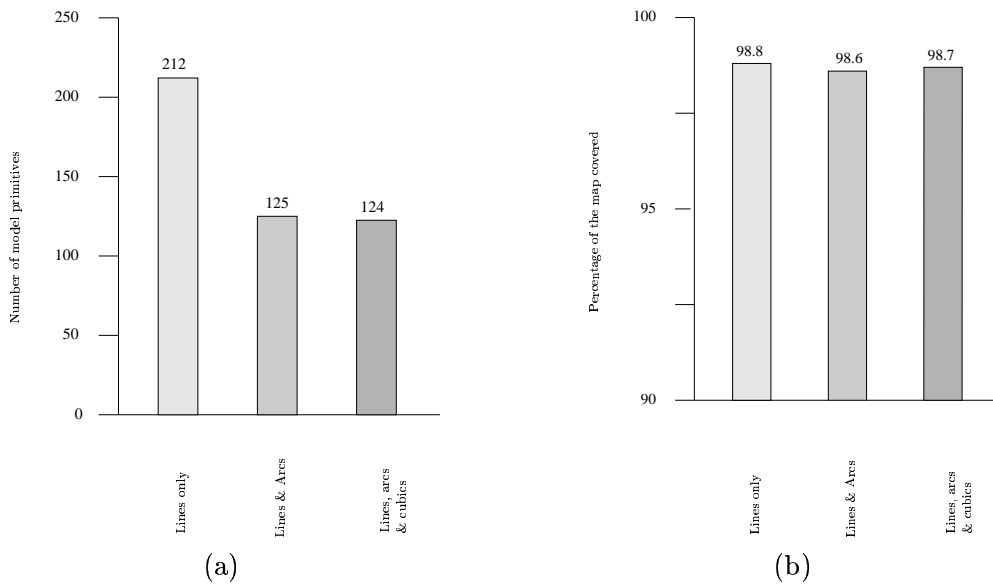


Figure 6: Comparison of (a) the number of primitives and (b) the percentage coverage of the map for the three primitive sets for the environment of Figure 4.

the environment of Figure 3 the lines segment primitive is perfectly appropriate but the addition of the arc and cubic primitives gives very little improvement. For the environment of Figure 4 which contains circular walls, the line segment primitive is inadequate and a curved primitive is required. Once again, the cubic primitive gives no improvement.

4.2.6 Operation Selection

The next aspect of the identification process to consider is the choice of operations and their effect upon the identification speed and the quality of the resulting model. For this comparison, we used the line primitive only and considered five different sets of operations, as set out in Table 1.

Operation	Operation Set				
	A	B	C	D	E
Add new primitive	×	×	×	×	×
Join closest	×	×	×	×	×
Join second closest		×	×	×	×
Join third closest				×	×
Merge closest and second closest			×	×	×
Merge closest, second and third closest				×	×
Join and split					×

Table 1: The five different sets of operations tested.

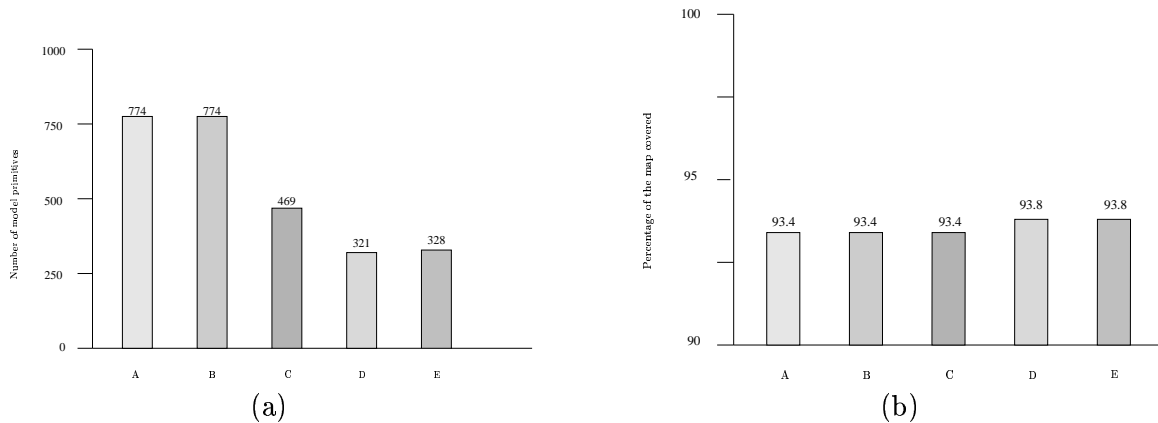


Figure 7: Comparison of (a) the number of primitives and (b) the percentage coverage of the map for the five different sets of operations for the environment of Figure 3.

The environment of Figure 3 was used and the results are given for runs of 1500 constrained points. Figure 7 shows the results of the identification process for each of the sets of operations. Comparison of the results for operation set B and C shows that the “merge closest and second closest” operation is very significant in simplifying the model. A similar comparison between sets C and D shows that the operations involving the third closest primitive significantly reduce the number of primitives in the model but the reduction is not as great as achieved by the operations involving the second closest primitive. However, the operations involving the third closest primitive help to achieve a superior model as shown by the 93.8% coverage achieved by sets D and E. Finally, the results for operation set E demonstrate that we can include splitting operations to correct mistakes as discussed in Section 4.2.4 without any negative impact on the quality of the model. The number of constraints does increase when the splitting operations are included but this is due to the seven cases where false joining occurred. The splitting operations have corrected these mistakes.

4.2.7 Parameter Selection

The remaining aspect of the constraint identification process is the selection of the parameters α and β . The parameter α is included in equation (4) to promote model simplification. Figure 8 presents the results of a number of runs with various values of α . Figure 8(a) shows that small values of α lead to increasing numbers of primitives in the model and Figure 8(b) shows that large values of α lead to excessive joining

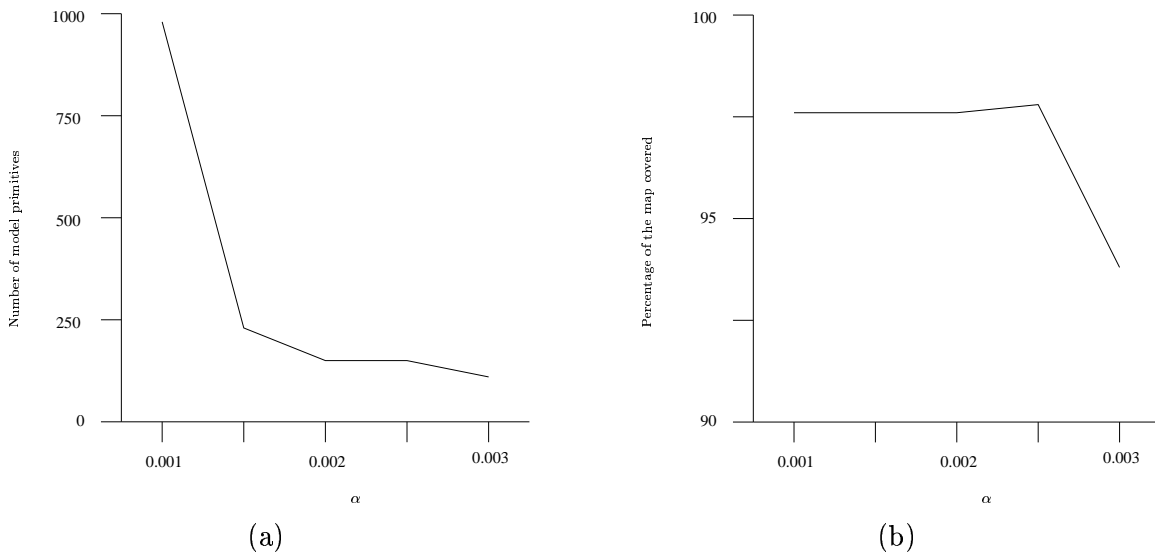


Figure 8: Comparison of (a) the number of primitives and (b) the percentage coverage of the map for a range of values of α . Note that $\epsilon = 0.01$.

of primitives and a corresponding loss in model quality. It can be seen that values of $\alpha \approx 0.002$ give a good tradeoff between model quality and number of constraints. Figure 9 shows the results of a number of runs with a range of values of β . These results show that the good values of β lie in the range $0.05 < \beta < 0.15$. Smaller values of β lead to spurious joining of distant points and a drop in model quality as shown in Figure 9(b). Similarly, larger values of β tend to prevent primitives growing longer and there is a greater number of primitives (see Figure 9(a)), as well as a drop in model quality (see Figure 9(b)).

4.2.8 Identification Rate

The rate at which constraints are identified clearly depends upon the controller being utilised. A random controller such as that used for Figures 3 and 4 is good for testing the constraint identification process as the constrained points thus obtained are uncorrelated. However, the random controller has very poor exploration qualities because it does not move towards areas which have not yet been explored. Figure 10 presents the map identified for the environment of Figure 4 with a wall-following controller. Clearly, the wall-following controller cannot fully explore this environment because it has islands in the middle of rooms. However, the controller does identify 62% of the map using 42 geometric objects after 969 constrained points were observed. This result indicates the potential for rapid map development when

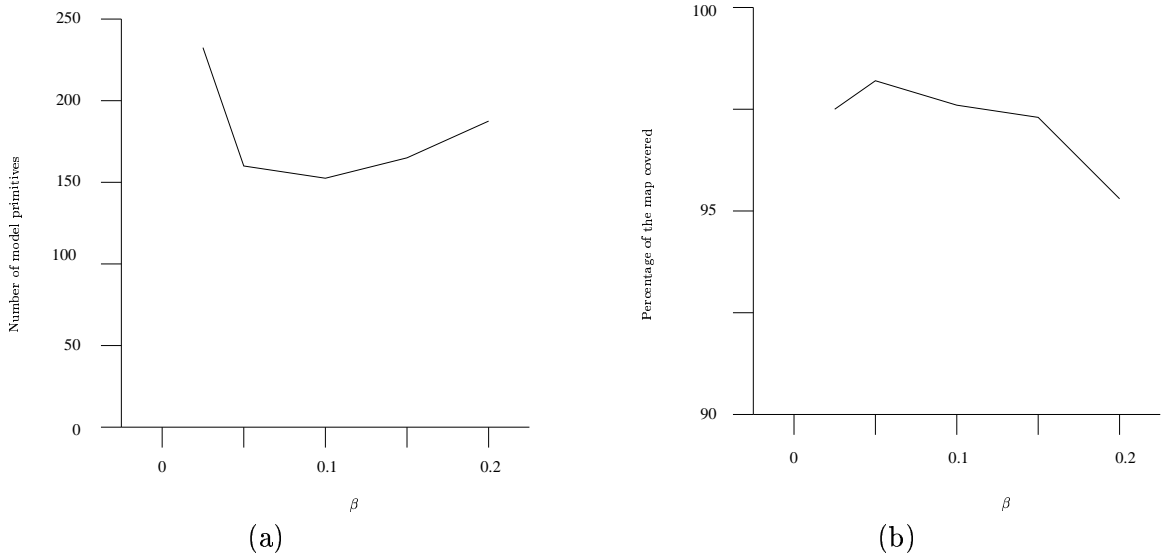


Figure 9: Comparison of (a) the number of primitives and (b) the percentage coverage of the map for a range of values of β . Note that $\alpha = 0.002$.

geometric mapping techniques are combined with an exploratory controller. Future works will investigate this area.

5 Application 2: Three-Dimensional Identification

A major advantage of geometric methods over grid-based maps is their ability to generalise well into higher dimensional spaces. In this section, we will study a rectangular mobile robot which operates in a three degree of freedom space. We assume that the robot has car-like kinematics. For a rectangular robot, the orientation of the robot is significant because the distance between the center of the robot and the extremities depends on the orientation of the robot. Thus, the location of the robot must be described by the triple $\langle x, y, \theta \rangle$, where θ is the orientation of the robot in some fixed reference frame. Hence, the state is given by:

$$\mathbf{x} = [x, y, \theta] \quad (17)$$

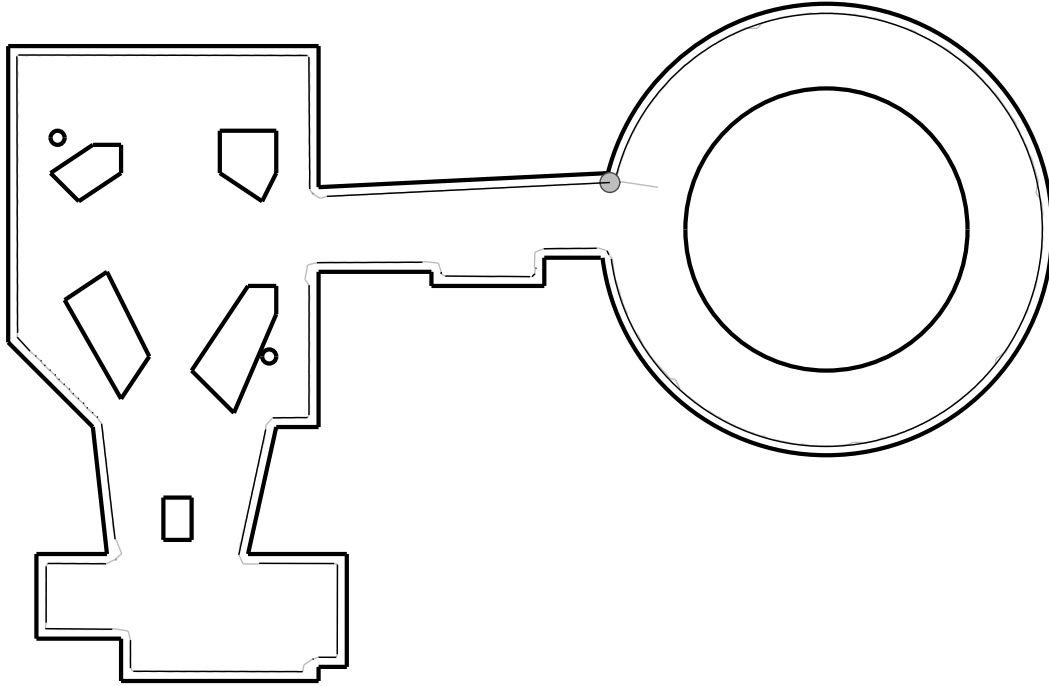


Figure 10: *Simulation results with wall-following controller. The grey line is the path followed by the robot, after starting near the junction in the annular room.*

5.1 Primitives

For the three-dimensional experiments, we chose to implement a planar polygon primitive. The polygon primitive is constructed by fitting a plane through all of the constrained points, projecting the points onto the plane, and then constructing a convex polygon around the projected points. The normals of the plane are also found and the appropriate normal is selected to indicate which side of the plane the constraint lies on. Implementation of primitives is considerably more difficult in three dimensions than for the two-dimensional case above and this primitive has the advantage that it is easy to implement. However, the actual constraints upon the robot will contain sinusoidal terms and so the planar polygon will always be approximating the actual constraints. This means that a number of planar polygon primitives will be needed for each sinusoidal constraint. This minor problem is outweighed by the ease of implementation of the planar polygon.

The error function for the planar polygon primitive is:

$$err_{polygon} = \sum_{i=1}^{n_j} dist(prim_j, \mathbf{x}_i) + \frac{\beta length_p()}{n_j} \quad (18)$$

where $dist(prim_j, \mathbf{x}_i)$ is the perpendicular distance from the constrained point \mathbf{x}_i to the plane, $length_p()$ is the greatest distance between any two constrained points, and β is a positive scaling constant. The first term of equation (18) measures the quality of fit of the plane to the points and the second term is included to discourage spurious joining of distant points. Again, β was selected according to equation (14) above.

5.2 Operations

For the three-dimensional case, we used the same motivations as in Section 4.2 and implemented a similar set of operations. The operations implemented are shown in Table 2 below.

Add new primitive
Join closest
Join second closest
Join third closest
Merge closest and second closest
Merge second closest and third closest
Merge closest three primitives
Join and split

Table 2: *The eight operations implemented for the three degree of freedom experiments.*

5.3 Results

The environment used for the three-dimensional identification is shown in Figure 11 and consists of a $2m$ square with a $1m$ square added on at the side. The robot was modelled as a rectangle $40cm \times 20cm$. The three-dimensional constraint identification proved very successful and developed the model shown in Figure 11 after 500 randomly distributed constrained points were observed. To identify a complete two-dimensional model of the same environment required only 350 random constrained points. As expected, the three-dimensional identification required higher numbers of constrained points than the comparable

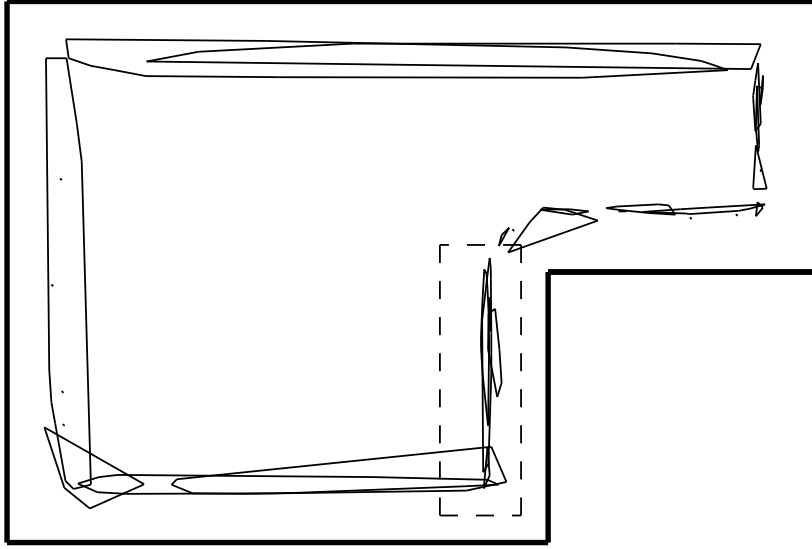


Figure 11: *Environment used for 3 degree of freedom experiments. The room is a 2m square with a smaller, 1m square added on the side. The robot is a 0.4m by 0.2m rectangle. This figure also shows the projection of the polygon primitives identified onto the x-y plane. Note that the primitives are no longer thin lines as the orientation of the robot affects the distance at which the constraint applies. The primitives in the dashed box are shown projected on the y- θ plane in Figure 12 below.*

two-dimensional case. Note that it is very difficult to determine an accurate model of the constraints in this environment for comparison with the identified primitives. The constraint functions are complicated and the extents of the constraints are poorly defined. For these reasons, we cannot find the percentage coverage of the actual constraints by the modelled constraints. However, a visual inspection of the model in Figure 11 gives some idea as to the quality of the model.

Figure 12 illustrates one of the problems with the three-dimensional model: the use of the orientation as the third dimension causes some primitives to be falsely divided. The inclusion of the orientation causes significant problems for the identification process. The first problem is that the magnitude of the angular measurements is different from the position measurements. The angles range from 0 to 2π whilst $0 < x < 3m$ and $0 < y < 2m$. This disparity causes difficulties for the error function of equation (18). To solve this problem, the angular measurements were scaled and $\frac{\theta}{2}$ was used for the third axis. The second problem is that angular measurements are periodic with $0 = 2\pi = 4\pi = 2n\pi$. We must choose a single period of 2π to work with but some primitives will lie across the border of whichever period we choose. For

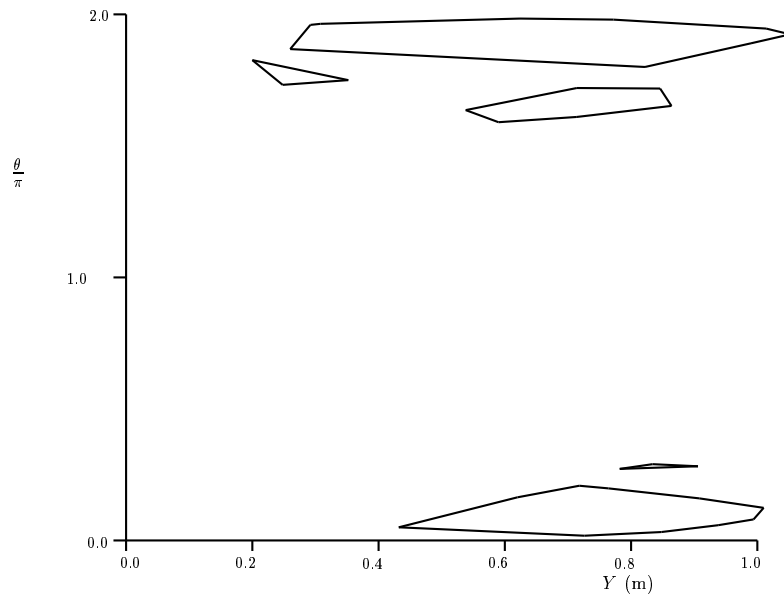


Figure 12: *Polygons from the box of Figure 11 projected onto the y and θ axes. Note that the polygons are split into two regions due to the use of a range of 0 to 2π for the angle θ .*

these simulations, we restricted θ so that $0 \leq \theta < 2\pi$. Figure 12 shows a projection of the boxed primitives of Figure 11 onto the y - θ plane. This figure clearly illustrates the problem with the primitive being split into two regions by the choice of $0 \leq \theta < 2\pi$. Both the problems of scaling and of splitting are inherent to the use of angles and cause some tolerable difficulties.

It is also worth noting that the random controller used is very bad at exploring the space, particularly at ensuring that exploration takes place in regions that have not been well identified. Figure 11 shows that the random controller does not explore the region of the smaller square and the fragmented polygons indicate that the model is poor in this area. Future work will investigate controllers which make use of the identified constraints and also explore the environment efficiently. Such controllers should be able to develop an accurate map many times faster than the random controller that we have used here.

6 Conclusion

A rigorous new method of map building for mobile robots was presented which uses geometric models of the constraints upon the robot. Geometric maps provide a compact alternative to grid-based maps, requiring considerably less memory and hence less computation for path planning. We have presented a new mapping method which identifies geometric models of the constraints imposed upon the robot by the environment. Simulation results have demonstrated the effectiveness of the constraint identification process for two very different real-world environments. The identification process is general and performs well, even with differing geometric primitives and operations to update the primitives. A key advantage of geometric models, the ability to cope with higher numbers of dimensions, was demonstrated with a simple three degree of freedom rectangular robot. This example also highlighted some of the problems that occur when one of the dimensions is an angular measure. The simulated results show that the identification framework presented here is a flexible and effective tool for finding geometric maps of complex environments. The simulation results also give a guide for the selection of geometric primitives to best match a particular environment.

7 Acknowledgments

The authors wish to thank Sebastian Thrun for providing the maps of Figures 3(a) and 4(a) and for his helpful advice.

References

- [1] E. G. Araujo and R. A. Grupen. Feature detection and identification using a sonar array. In *Proc. of 1998 IEEE Intl. Conf. on Robotics and Automation*, pages 1584–1589, May 1998.
- [2] N. Ayache and O. D. Faugeras. Maintaining representations of the environment of a mobile robot. *IEEE Transactions on Robotics and Automation*, 5(6):804–819, 1989.
- [3] J. Borenstein and Y. Koren. Histogramic in-motion mapping for mobile robot obstacle avoidance. *Journal of Robotics and Automation*, 7(4):535–539, 1991.

- [4] L. Cahut, K. P. Valavanis, and H. Delic. Sonar resolution-based environment mapping. In *Proc. of 1998 IEEE Intl. Conf. on Robotics and Automation*, pages 2541–2547, May 1998.
- [5] J. A. Castellanos, J. M. Martinez, J. Neira, and J. D. Tardos. Simultaneous map building and localization for mobile robots: a multisensor fusion approach. In *Proc. of IEEE Intl. Conf. on Robotics and Automation*, pages 1244–1249, May 1998.
- [6] J. A. Castellanos, J. D. Tardos, and G. Schmidt. Building a global map of the environment of a mobile robot: The importance of correlations. In *Proc. of 1997 IEEE Intl. Conf. on Robotics and Automation*, pages 1053–1059, May 1997.
- [7] K. S. Chong and L. Kleeman. Sonar based map building for a mobile robot. In *Proc. IEEE Intl. Conf. on Robotics and Automation*, volume 2, pages 1700–1705, 1997.
- [8] J. L. Crowley. Navigation for an intelligent mobile robot. *IEEE Journal of Robotics and Automation*, 1(1):31–41, 1985.
- [9] L. Delahoche, C. Pegard, E. M. Mouaddib, and P. Vasseur. Incremental map building for mobile robot navigation in an indoor environment. In *Proc. of IEEE Intl. Conf. on Robotics and Automation*, pages 2560–2565, May 1998.
- [10] A. Elfes. Sonar-based real-world mapping and navigation. *IEEE Journal of Robotics and Automation*, 3(3):249–265, 1987.
- [11] J. A. Janet, S. M. Scoggins, M. W. White, J. C. Sutton, E. Grant, and W. E. Snyder. Self-organising geometric certainty maps: A compact and multifunctional approach to place recognition and motion planning. In *Proc. IEEE Intl. Conf. on Robotics and Automation*, volume 4, pages 3421–3426, 1997.
- [12] Y. D. Kwon and J. S. Lee. A stochastic environment modelling method for mobile robot by using a 2-d laser scanner. In *Proc. of IEEE Intl. Conf. on Robotics and Automation*, pages 1688–1693, April 1997.
- [13] P. J. McKerrow. Echolocation - from range to outline segments. *Journal of Robotics and Autonomous Systems*, 11:205–211, 1993.
- [14] D. Murray and C. Jennings. Stereo vision based mapping and navigation for mobile robots. In *Proc. of IEEE Intl. Conf. on Robotics and Automation*, pages 1694–1699, April 1997.
- [15] U. Raschke and J. Borenstein. A comparison of grid-type map-building techniques by index of performance. In *Proc. of IEEE Intl. Conf. on Robotics and Automation*, pages 1828–1832, May 1990.
- [16] S. Scheduling, E. M. Nebot, M. Stevens, and H. Durrant-Whyte. Experiments in autonomous underground guidance. In *Proc. 1997 IEEE Intl. Conf. on Robotics and Automation*, pages 1898–1903, April 1997.
- [17] S. Thrun and A. Bucken. Learning maps for indoor mobile robot navigation. Technical Report CMU-CS-96-121, School of Computer Science, Carnegie Mellon University, April 1996.
- [18] S. Thrun, D. Fox, and W. Burgard. Probabilistic mapping of an environment by a mobile robot. In *Proc. of IEEE Intl. Conf. on Robotics and Automation*, pages 1546–1551, May 1998.
- [19] B. Yamauchi, A. Schultz, and W. Adams. Mobile robot exploration and map-building with continuous localization. In *Proc. 1998 IEEE Intl. Conf. on Robotics and Automation*, pages 3715–3720, May 1998.