

# Non-monotonic Geometric Mapping for Mobile Robots

David J. Austin

d.austin@computer.org

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

**Abstract.** A new method of map building for mobile robots is presented which can correct for past mistakes. This ability of correction of past mistakes is critical for building of accurate maps of the environment and for maintaining maps over a period of time. Previous approaches can be divided into grid-based and geometric map building. Grid-based maps are easy to build but do not scale to higher dimensional systems and require considerable storage space. Instead, a geometric mapping method is used here. The mapping problem is cast as a minimisation problem and the map is then constructed by gradient descent search. Preliminary experimental results in a real world environment are presented which demonstrate the applicability of the method and show the usefulness of non-monotonic map building.

## 1 Introduction

The problem of map building for mobile robots has been extensively studied because a map permits high level, “intelligent” planning of robot motions. The process of map construction requires the interpretation of noisy sensor information and the aggregation of that information into some global representation of the environment of the robot. One of the commonly neglected problems of map building is how to correct mistakes. However, we must accept that making decisions from noisy sensory information will invariably result in mistakes which, in the light of new information obtained in the future, must be corrected.

*It is logically impossible to reason successfully about the world around us using only deductive reasoning. All interesting reasoning outside of mathematics involves defeasible steps.* – John L. Pollock (1989)

The ability to correct past decisions is called non-monotonic reasoning in the artificial intelligence community. In this paper, we will study the problem of non-monotonic map building.

There are two basic approaches to map building, *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 [6] and, more recently, Thrun *et al.* [10, 9]. 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. Note that grid-based maps can correct mistakes but do so in a very slow way. Grid-based maps average a number of measurements into a single cell so that a single incorrect

measurement can be corrected in time. However, grid-based maps suffer from the big disadvantage of their large size.

On the other hand, geometric approaches attempt to directly identify and model features (e.g. line segments) 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 [4] developed a sonar mapping system which used recursive line fitting to fit a polyline to sonar measurements. McKerrow [7] investigated sonar-based geometric mapping using primitive objects of `partial plane` and `corner`. Castellanos *et al.* [2, 3] extract `points`, `corners`, `segments` and `partial planes` from laser and vision information. Also, Scheduling *et al.* [8] proposed a method for fitting a polyline to laser range data.

However, little has been considered in terms of non-monotonic reasoning for geometric mapping. It seems clear that non-monotonic reasoning is an important aspect of building accurate maps and is crucial for dealing with errors and changes in the environment. In order to develop non-monotonic reasoning it is necessary that a single framework is developed that integrates and regularises the decisions that are made during the map building process. Here, the map building process is cast as a minimisation problem so that a single cost function governs all of the map making decisions. The next section presents the few assumptions that are made and Section 3 presents the framework that is used for identification of geometric models. Finally, experimental results for a corridor and our laboratory are presented in Section 5.

## 2 Assumptions

For this paper, we will assume that the robot can determine its position in a fixed reference frame. This is a strong assumption but the focus of this paper is on non-monotonic mapping. Future research will extend the techniques presented here to situations where the robot position is only known approximately. In addition, we will assume that the robot is equipped with range sensors which return a number of *range readings* which are distances to objects in the environment. For the moment, we will concentrate on the SICK laser scanner which is mounted on a XR4000 robot. Each range reading has a start and end point which are the center of the laser scanner and a point on an object in the environment respectively. It is important to consider range readings in this manner because it tells us both where the free space is and where objects are in the environment.

## 3 Map Building by Minimisation

As mentioned above, building of geometric maps is a very difficult problem. Consider a naive approach to find the optimal solution : 1) build all possible groupings of the range readings, 2) fit line segments to the end points of the range readings in each grouping and 3) find the best resulting map. Given any metric or quality measure for maps, this procedure seems perfectly applicable. However, the sheer number of combinations of range readings into groups defeats this approach. For  $n$  range readings we can have from 1 to  $n$  groupings and the number of possible combinations is given by:

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

Even for a tiny experiment of 20 constrained points there are a staggering  $3.23 \times 10^{11}$  possible combinations. Clearly, we cannot try exhaustive, brute-force approaches and must use sub-optimal techniques. Most research in geometric map building neglects any quantitative measure of the map quality and relies on a set of parameters that have been tuned to give maps that appear “good”. Instead, we will explicitly form a quality function and define a set of *operations* which can be applied to the map. Map optimisation proceeds by finding the best operation to apply at each step. This is a sub-optimal process because the set of operations limits the search.

We keep a record of the range readings and denote them  $\mathbf{x}_i$ ,  $i = 1 \dots n$ , where  $\mathbf{x}_i = [x_s \ y_s \ x_e \ y_e]^T$  contains the start and end points of the range reading and  $n$  is the number of range readings observed so far.

The process of map building involves developing a model of the environment from the observed range readings. The mapping process is cast as a minimisation problem. For this paper, we will model the environment with a set of points and line segments (other geometric objects have been considered in [1]). Two sets of operations are then defined which: a) integrate new sensor readings into the existing model and b) clean up the resulting model. When a new sensor reading is taken, each of the additive operations is tried and the one which minimises the error function is selected. After the integration of a sensor measurement, the cleaning operations are used to remove some of the errors caused by real-world data.

The map of the environment is defined as a set of  $m$  points and line segments, each of which represents part or all of an object that exists in the environment. We will denote the model after  $n$  range readings as  $G(n)$

$$G(n) = \{o_j : j = 1 \dots m\} \quad (2)$$

where  $o_j$  is the  $j^{th}$  object (point or line segment). Each of the range readings is assigned to one of the objects. Once the range readings have been assigned to objects, the parameters of the the object can be determined. The location of the point object is determined as the average of the end points of the assigned range readings. For line objects, the parameters are determined by an eigenvector fit to the end points [5]. Note that the set  $G(n)$  is assumed to be initially incomplete and may even be empty.

### 3.1 Map Quality Measure

An important aspect of this method is the quality measure used to determine quantitatively the quality of the map. Three factors were considered important in the selection of the measure:

1. a line segment should not be crossed by a range reading,
2. the end point of each range reading should be near to it's assigned object (point or line segment), and
3. the map should not contain too many objects.

These considerations led to the following choice of quality measure:

$$Q(G) = \sum_{i=1}^n \sum_{j=1}^m [d(o_j, \mathbf{x}_i) + i(o_j, \mathbf{x}_i)] + \alpha m \quad (3)$$

where  $d(o_j, \mathbf{x}_i)$  is the distance from the  $j^{th}$  object to the end point of the  $i^{th}$  range reading, the function  $i(o_j, \mathbf{x}_i)$  gives the amount that range reading  $\mathbf{x}_i$  passes through object  $o_j$ , and  $\alpha$  is a penalty constant for each new line in the map. Note that the intersection function  $i(o_j, \mathbf{x}_i)$  is defined to be zero when the object  $o_j$  is a point object. The quality measure of (3) is to be minimised to maximise the quality of the map. The first term of equation (3) measures the quality of fit of the objects to the range readings, the second term penalises any intersection of line segments and range readings and the third term promotes model simplification.

### 3.2 Model Update Process

After the observation of the  $n^{th}$  range reading, we must determine an improved model  $G(n)$  which encompasses the new data. There are two possible approaches: we could re-process the entire list of range readings or we could take advantage of the incremental nature of exploration and just update the model  $G(n-1)$  to include the new range reading. The incremental approach is considerably better because it provides an up-to-date model at every stage (useful for navigation) and requires less computation.

The incremental updates of the model are defined by two sets of operations. The *additive operations* are the possible actions that can be performed to integrate each new observation  $\mathbf{x}_n$  into the model. The *cleaning operations* operate on the existing model to improve it. Each additive 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:

$$G_k(n) = oper_k(G(n-1), \mathbf{x}_n) \quad (4)$$

where  $oper_k$  is the  $k^{th}$  operation. Once we have observed the  $n^{th}$  range reading, each of the available operations is tried and the model which minimises the error of equation (3) is selected:

$$G(n) = \arg \min_{G_k} (Q(G_k(n))) \quad (5)$$

## 4 Implementation

The previous section has established a systematic framework for building feature-based maps. The choice of the set of operations is clearly important and determines the quality of the resulting map. In [1] the selection of a set of operations was discussed in detail, both for two- and three-dimensional map building. Two types of operations are used: additive operations for integrating new measurements into the map and cleaning operations for fixing mistakes and improving the quality of the map.

### 4.1 Additive Operations

For the experiments presented here, two additive operations have been selected: `CreatePoint` and `JoinLineN`. The `CreatePoint` operation takes the new range reading and forms a new point from the end point of the range reading. The `JoinLineN` operation joins the range reading to the readings from the  $n^{th}$  closest object and creates a line segment from the resulting set of readings. For the results presented here, the `JoinLineN` operation was instantiated ten times as: `JoinLine1` to `JoinLine10`.

### 4.2 Cleaning Operations

Cleaning operations do not add new data to the map; instead they try to remove errors from the map and to reduce the quality measure (equation (3)). Cleaning operations try to detect objects (or pairs of objects) which are suitable for a particular change and apply that change if it reduces the error function. For this paper, the following cleaning operations were implemented `MergePoint`, `MergeLineLine`, `MergePointLine`, `SplitPoint`, and `SplitLineLine`. The implementation of each of these operations will now be described in turn.

The `MergePoint` operation seeks to join a number of range readings and form a point object from them. The operation seeks a two target objects randomly from the set of objects and forms a point from the range readings of both objects. If the error of the new point object is less than the sum of the errors of the two prior objects then the prior objects are deleted and the new point is inserted into the map. When choosing the two target objects randomly, each existing object in the map is weighted with the error as quantified by equation (3). In this way, objects with high error are targeted by this cleaning operation. Note that the `MergePoint` operation is unlikely to be significant because point objects are not required to represent the environments used for the experiments.

The `MergeLineLine` operation seeks to merge two line segment objects to form a new line segment. First, each pair of line segments is weighted according to their collinearity and then a pair is selected randomly (using the weights). A new line segment is formed from the range readings of the pair of lines and is used in the map if it has a lower error. The collinearity weight is determined by computing the angle  $a$  of each line with the  $x$ -axis (in radians) and the distance  $d$  of each line from the origin. The weight is then computed according to:

$$weight = (d_1 - d_2)^2 + (a_1 - a_2)^2 L^2 \quad (6)$$

where  $L$  is the minimum length of the two line segments and is used to ensure that the distance and angle differences are comparable.

The `MergePointLine` operation seeks to join a point object to a line segment and form a new, longer line segment from the integrated data. Each line segment-point pair is first weighted by the perpendicular distance of the point from the line and then a pair is selected randomly using these weights. A new line segment is formed from the range readings of both objects. If the new line segment has a lower error than the existing two objects then the two objects are replaced by the new line.

The `SplitPoint` operation seeks to split a range reading with high distance between the end point of the range reading and its assigned object away from that object, to form a new point object. A target object is selected randomly from the objects in the map (weighted by their error) and then the worst range reading for that object is found. Two new objects are formed: a point using just the bad range reading and a replacement object for the target which uses all but the bad range reading. If the error of the two new objects is less than the target object error then the target object is replaced by the two new objects.

The `SplitLineLine` operation seeks to split a line segment with high errors into two smaller line segments. A target line segment is selected randomly from the objects in the map (weighted by their error) and then the range reading which passes through the target most is found. Two new line segments are formed by splitting the target line segment on either side of the intersecting range reading. If the two new line segments have lower error, then they are inserted into the map and the original line segment is deleted.

## 5 Results

The XR4000 robot used for the experiments is equipped with a SICK LMS laser range scanner. This sensor returns the distance at 361 angles (separated by  $0.5^\circ$ ) with a discretisation of 1cm. The sensor is mounted at a height of 490mm above the ground. This is quite low and so the results are quite susceptible to clutter in the environment. Therefore, results are given for two situations: in the corridor which is free of clutter and in our laboratory which has everyday furniture and is quite cluttered. For the results presented here, each of the cleaning operations was tried 10 times after every 50<sup>th</sup> range reading was added. Experiments were performed in two environments, where a number of scans were taken at several positions and orientations. The scans were matched by hand because the odometry of the XR4000 is poor, particularly in terms of orientation.

Figure 1 shows results obtained in the corridor using the algorithm described above. This experiment consists of 2050 range readings taken from three positions (two orientations at each position) outside our laboratory. The map shown in Figure 1(b) shows the potential of this method and also shows one of the most important features of the method. Since this method can correct for mistakes, it is able to jump to conclusions. Thus, given the quality function above and two range readings (in the absence of any other information) the method will connect the two range readings into a line segment. In some cases, this will be a gross error as the two range readings may lie at some great distance and there may be nothing but free space in between. However, many methods for geometric mapping will not consider fitting such a line because there is no method to correct it later.

Also, Table 1 shows how many times each of the operations was used in forming the map of Figure 1. Here we have the expected result that the `JoinLine` operations are used less as the distance to the line increases. Also note that the cleaning operations were not used terribly frequently. This indicates that more work is needed in this area.

Our laboratory has been furnished like a living room and contains a dining table, chairs, shelves, sofa and coffee table. This is a real world environment. However, the clutter that is in the room, particularly with the laser scanner placed so low (490mm), significantly affects the ability of the method. When clutter is present it is not possible to model the environment as line segments and so the method produces poor maps. Figure 2 shows the results of the map built from 2527 range readings, again taken from three different positions (with two orientations at each position). This map could do with some improvement but it has captured the dominant features in the environment. It can be seen from the range readings that the environment is very complex and contains many curved features. For example, at the bottom of the map, towards the left side, the outline of the sofa is visible, with the curved shapes of the three cushions showing. Future work will investigate the use of curved objects in the same framework. Table 2 shows how often each of the operations was used to build the map of Figure 2. Here we can see that the method is not performing as expected because the use of the `JoinLine` operations does not decrease with distance.

### 5.1 Computational Complexity

While the computational complexity of this method is generally low, one aspect leads to high computational cost. The part of the cost function that requires that range readings not intersect line segments leads to consider-

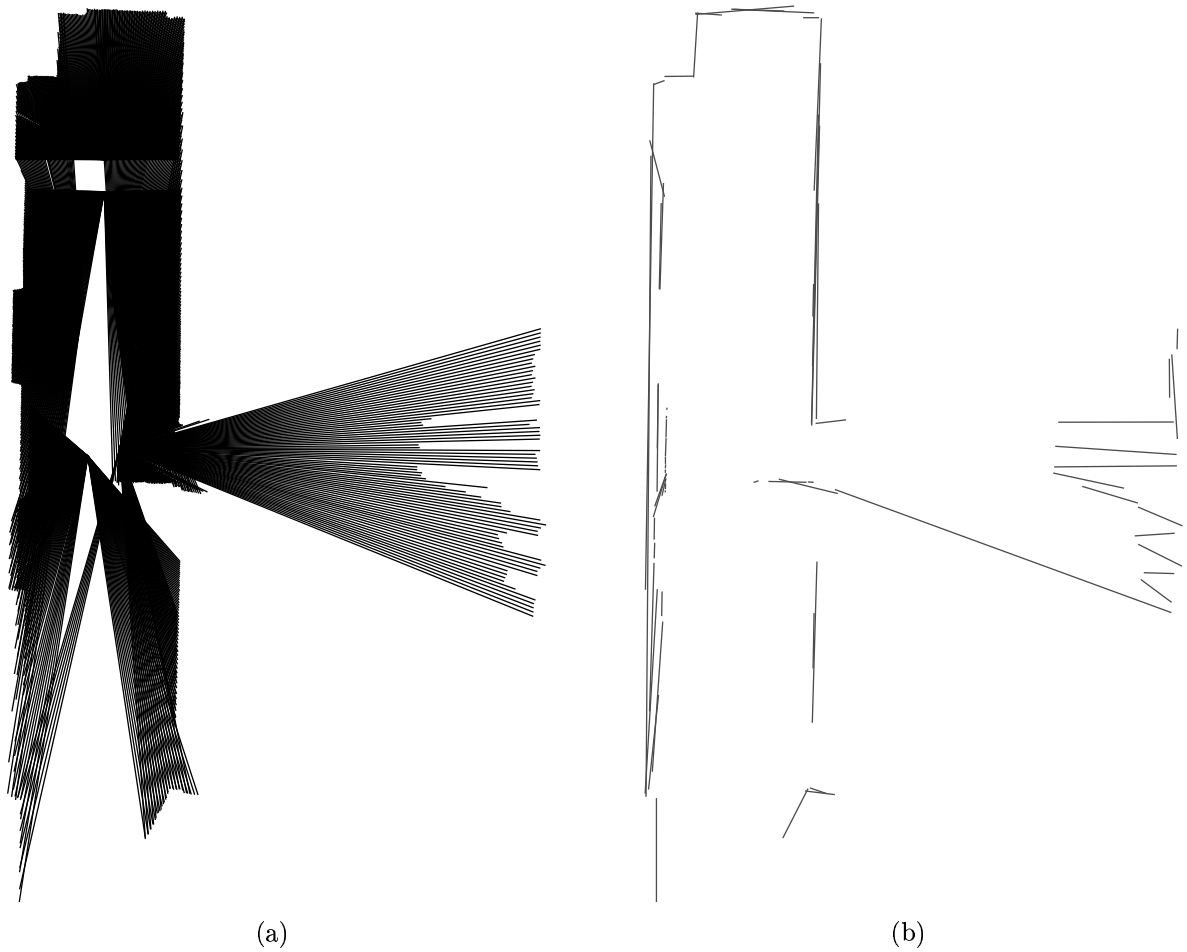


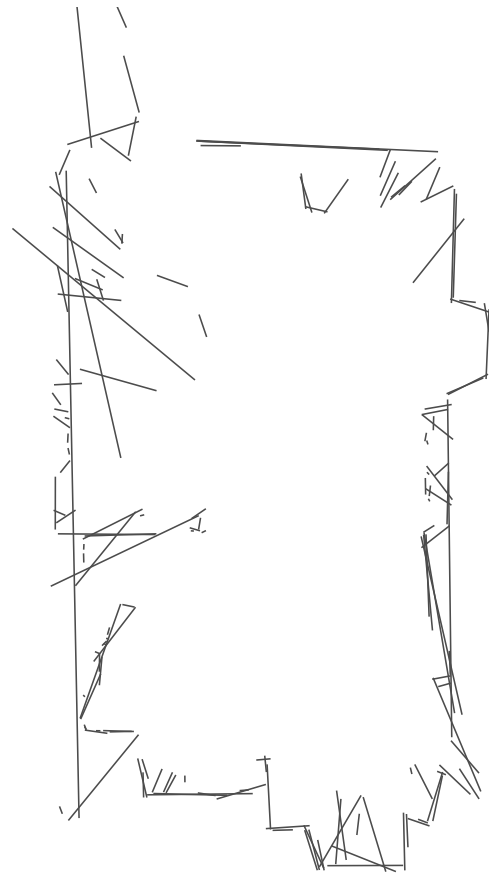
Figure 1: *Results for a corridor: (a) 2050 range readings (b) map formed from the 2050 range readings. Note that only readings which hit an object in the environment are shown.*

Operation	Frequency
CreatePoint	97
JoinLine1	629
JoinLine2	377
JoinLine3	460
JoinLine4	365
JoinLine5	74
JoinLine6	32
JoinLine7	5
JoinLine8	4
JoinLine9	2
JoinLine10	5
SplitPoint	0
MergeLineLine	6
MergePointLine	3
SplitLineLine	2

Table 1: *Usage information for the operations used to build the map shown in Figure 1*



(a)



(b)

Figure 2: Results for the laboratory: (a) 2527 range readings (b) map formed from the 2527 range readings.

Operation	Frequency
CreatePoint	343
JoinLine1	386
JoinLine2	368
JoinLine3	209
JoinLine4	220
JoinLine5	190
JoinLine6	139
JoinLine7	105
JoinLine8	167
JoinLine9	165
JoinLine10	235
SplitPoint	0
MergeLineLine	11
MergePointLine	0
SplitLineLine	0

Table 2: Usage information for the operations used to build the map shown in Figure 2

able computation. For example, during construction of the two maps shown, 75% of the time was spent checking for intersections. The time taken to build the two maps shown was 72 seconds for the corridor of Figure 1 and 135 seconds for the laboratory of Figure 2. While this amount of computation is acceptable for mapping of small areas it is clear that for larger scales it is not possible to check for intersection between every line segment and range reading. The process of checking is simply too time consuming and scales linearly with the number of range readings.

## 6 Conclusion and Future Work

In this paper a method for map building has been presented which can correct for past mistakes. The map building process was cast as a minimisation problem by the introduction of a quality measure and a set of operations that can be applied to the map. This mathematical framework for map building gives a more rigorous approach and gives non-monotonic reasoning. The results presented clearly demonstrate the ability of this method to jump to conclusions and still be able to build reasonable maps. To be able to do both of these depends entirely upon the ability to correct past mistakes.

The results presented in this paper are very promising and indicate the strengths of the method for building geometric maps. However, a two areas remain to be investigated. First, it is necessary to better target the cleaning operations: they were used relatively infrequently in the experiments. The second area for further development is in determining if range readings and line segments intersect. Either a considerably more efficient method for determining intersection must be developed or a technique for determining significant range readings (and discarding the others) is required.

## References

- [1] D. J. Austin and B. J. McCarragher. Geometric constraint identification and mapping for mobile robots. Accepted for *Journal of Robotics and Autonomous Systems*, 2000.
- [2] 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.
- [3] 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.
- [4] J. L. Crowley. Navigation for an intelligent mobile robot. *IEEE Journal of Robotics and Automation*, 1(1):31–41, 1985.
- [5] R. O. Duda and P. E. Hart. *Pattern Classification and Scene Analysis*. Wiley, 1973.
- [6] A. Elfes. Sonar-based real-world mapping and navigation. *IEEE Journal of Robotics and Automation*, 3(3):249–265, 1987.
- [7] P. J. McKerrow. Echolocation - from range to outline segments. *Journal of Robotics and Autonomous Systems*, 11:205–211, 1993.
- [8] 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.
- [9] 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.
- [10] 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.