KTH Computer Science
and Communication

Simultaneous Localization and Mapping with Robots

[1] JOHN FOLKESSON

Doctoral Thesis
Stockholm, Sweden 2005

TRITA-NA-0528
ISSN 0348-2952
ISRN KTH/NA/R--05/28--SE
ISBN 91-7178-145-5
CVAP-297

KTH
School of Computer Science and Communication
SE-100 44 Stockholm
SWEDEN

Akademisk avhandling som med tillstånd av Kungl Tekniska högskolan framlägges till offentlig granskning för avläggande av teknologie doktorsexamen i datalogi tisdag den 4 oktober 2005 klockan 13.00 i sal M3, Kungl Tekniska högskolan, Brinelvägen 64, Stockholm.

Tryck: Universitetsservice US AB

Abstract

A fundamental competence of any mobile robot system is the ability to remain localized while operating in an environment. For unknown/partially known environments there is a need to combine localization with automatic mapping to facilitate the localization process. The process of Simultaneous Localization and Mapping (SLAM) is the topic of this thesis.

SLAM is a topic that has been studied for more than 2 decades using a variety of different methodologies, yet it deployment has been hampered by problems in terms of computational complexity, consistent integration of partially observable features, divergence due to linearization of the process, introduction of topological constraints into the estimation process, and efficient handling of ambiguities in the data-association process. The present study is an attempt to address and overcome these limitations.

Initially a new model for features, inspired by the SP-map model, is derived for consistent handling of a variety of sensor features such as point, lines and planes. The new feature model enable incremental initialization of the estimation process and efficient integration of sensory data for partially observable features. The new feature model at the same time allow for consistent handling of all features within a unified framework.

To address the problems associated with data-association, computational complexity and topological constraints a graphical estimation method is derived. The estimation of features and pose is based on energy optimization. Through graph based optimization it is possible to design a feature model where the key non-linearities are identified and handled in a consistent manner so as to avoid earlier discovered divergence problems. At the same time any-time data-association can be handled in an efficient manner. Loop closing in the new representation is easily facilitated and the resulting maps show superior consistency even for large scale mapping problems.

The developed methods have been empirically evaluated for SLAM using laser and video data. Experimental results are provided both for in-door and out-door environments.

The methods presented in this study provide new solutions to the linearization problem, feature observability, any-time data association, and integration of topological constraints.

Acknowledgment

I would like to thank the people who made this thesis possible. My wife Maggie for relentless support. My children Nina, Sanya, Jasmin and Maxwell for putting up with a distracted father. My adviser Henrik for giving me the freedom to work on my ideas. My friend and co-coder Patric for knowing all the answers. And Anders for knowing the answers Patric didn't know. Andreas for taking the load off me. Without these people I would not have finished this.

Then there were all the good people in the lab whom it was a pleasure to work with. A few of these: Dani, Mårten, Jan Olof, Tony, Petter, Guido, Phillip, Carl, Elin, Ola, Nick, Peter Ville, the other Carl, Kai, Mattias,...

The present work has been performed within the Centre for Autonomous Systems, KTH with support for the Swedish Foundation for Strategic Research (SSF). The support is gratefully acknowledged.

Contents

Chapter 1

Introduction

A fundamental competence of any mobile robot systems is the ability to remain localized while performing tasks on behalf of a user. The problem of mobile robot localization and navigation is a well-known and widely studied problem. Actually the problem is not unique to robotics, it is a problem that has been studied for several millenniums, as discovers have traveled the world. Surveyors have generated map that travelers have used to go from one location to another. In the field of surveying there are well known methods for estimation of the position of a vehicle and also a wide diversity of methods for mapping of the world.

Despite these diverse efforts this is not entirely a solved problem. Localization involves five basic components

Sensory data about structures in the environment

A model for the motion of the platform

A model of the environment

A method for matching of sensory data to the model

Estimation of the position of the platform given a correspondence between the sensory data and the models

The basic mathematical framework for localization of robots is largely considered a solved problem. A more challenging problem is the ability to concurrently estimate the position of a vehicle and construct/update the model of the environment. This problem is normally referred to as the problem so simultaneous localization and mapping (SLAM) or concurrent mapping and localization (CML). The basic mathematics underlying SLAM is also widely considered a solved problem. Yet there are relatively few practical systems that can perform unconstrained SLAM in general environments. The problem of SLAM is similar to the localization problem outlined above, but the fifth component is updated to include estimation

of ego-position and location of features in the world, and a sixth component for the updating of the model as new structures are discovered.

Early work on SLAM was based on standard estimation methods such as the Extended Kalman Filter. That method has quadratic computational complexity, which poses a challenge to deployment in large scale environments. In addition the system is in general non-linear which has resulted in use of the Extended Kalman Filter which adds the linearization of all non-linearities to a standard Kalman Filter. It has recently been demonstrated that the linearization might result in divergence in the mapping process. Another complication is also consistent integration of information from different types of features such as points, lines, planes, etc. Each of these features offer varying degrees of information, which preferably should be handled in a consistent fashion. As a system travels through the environment it might return to the same location again, which allows one to impose topological constraints on the overall estimation process. The introduction of such constraints into the estimation process poses a challenge and the result is often less than optimal. The main challenges in design of SLAM systems are thus:

- Scalable handling of large scale environment
- A methodology that allow mapping without divergence in the presence of non-linearities
- Consistent integration of different types of sensory information
- Efficient and correct introduction of topological constraints

The objective of the present study is to provide a methodology for SLAM that addresses each of these challenges. In addition to formulation of a theoretical model that addresses these problems, it is to be evaluated on a number of real-world problems.

The thesis is organized in the following fashion. In Chapter 2 the basic SLAM methodology is introduced in a more formal way, and the literature on SLAM is briefly reviewed. Based on this the basic framework addressed in the thesis is detailed. Autonomous motion of the platform implies a change in the robot reference frame which influences the overall estimation process. The kinematic modeling of the change is discussed in Chapter 4. A model for consistent integration of different types of features is presented in Chapter 5 and the method is initially integrated into an Extended Kalman Filter framework. Unfortunately the framework is still not scalable or necessarily stable in the presence on non-linearities. To accommodate this a graphical model for SLAM is formulated in Chapter 6. The method enable handling of all the challenges outlined above, as demonstrated by a number of empirical evaluations. Chapter 7 presents a summary of the overall work, and a number of identified issues for future work are presented in Chapter 8. Finally an overall conclusion of the study is provided in Chapter 9. Supplementary information is available in appendices

Part I

Preliminaries

Chapter 2

SLAM Background

2.1 Problem Statement of SLAM

The SLAM problem can be stated in a couple of ways.

Have a robot drive around and make a map of the environment.

Or maybe:

Have the robot drive around without getting lost.

It turns out that the second definition is more useful. This is because the interpretation of the environmental sensor data is much easier if the sensor is well localized. So that even if it is the map we ultimately are interested in, it is necessary to have the robot stay well localized as it makes the map. Thus the map in this work is always one designed to keep the robot from getting lost.

Therefore the map is being made for the purpose of localization. Several types of maps have been used in SLAM. These include occupancy grids [1, 2, 3, 4, 5], reference sensor data (Scan matching) [6, 7, 8] , topological place maps [9, 10, 11, 12] and feature maps [13, 14]. In this work we will only consider feature maps. It has been found that when using a feature map for localization it is sufficient to have a few good features in view at all times. Having more than a few does not help. The common consensus is to have a small uniform density of very good features in the maps.

In SLAM we will deal with two types of measurements[1], robot motion measurements and relative measurements between features and the robot, figure 2.1. It is the feature measurements that will be used to estimate the map. However since these are relative to the robot pose the motion measurements also affect the map.

Thus a clean separation of the SLAM problem into a feature estimate and a robot pose estimate is difficult. There has been some work done using relative feature measurements [15]. The methods that decouple the mapping and localization problems rely on only using measurements that relate features to one another. Thus some of the information is not used.

[1] One can also use feature to feature measurements but we will not do so in this work

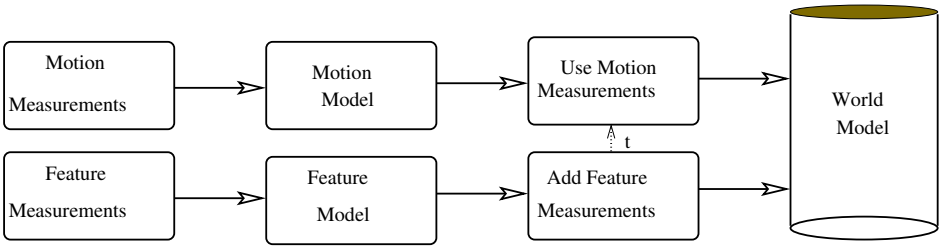

Figure 2.1: The SLAM estimation problem is split between a part that uses the motion measurements to predict the robot motion up the time of the feature measurements and a part that uses the feature measurements to update the world model.

If we want to use the measurements of the features relative to the robot directly, then the robot motion will effect the way we use the feature measurements. A more detailed and specific model of SLAM is shown in figure 2.2. Here we include the complications that arise from not knowing the size of the world model from the start. The world model will need to be augmented with new features when the measurements cannot be explained by the existing features. Figure 2.2 is the basic model we use in this work. We model an iterative SLAM algorithm as a four phase process.

The first phase, *predict*, is triggered by a new feature measurement and uses the proprioceptive measurements of the motion to bring the world model up to the time of the feature measurement.

The second phase, *update*, uses the feature measurements to improve the world model. This is refined in figure 2.3, where we show that the update phase must first try to match the feature measurement to some map feature and then make some new estimate of the map and pose based on this measurement. If a matching feature cannot be found then a new feature is created and added to the map. Note that this feature creation is unconditional. We can do this since we will not use the feature for SLAM estimation until it has accumulated enough information as described below. Thus if the measurement is spurious the feature will never be initialized and will have no effect on the rest of the world model.

The data association cannot in general be separated from the map update as the SLAM algorithm used will dictate the matching criteria. In the particular algorithm, it may be necessary to incorporate the new measurement into the world model in order to tests its fit before we can make a judgment of the data association.

The third phase is *add information*. Here the information in the measurement is added to the feature. This is used to estimate the parameters of the feature that have not been initialized yet. That is the information not used in the estimate of the world model in the update step. This might be all the information. Alternatively

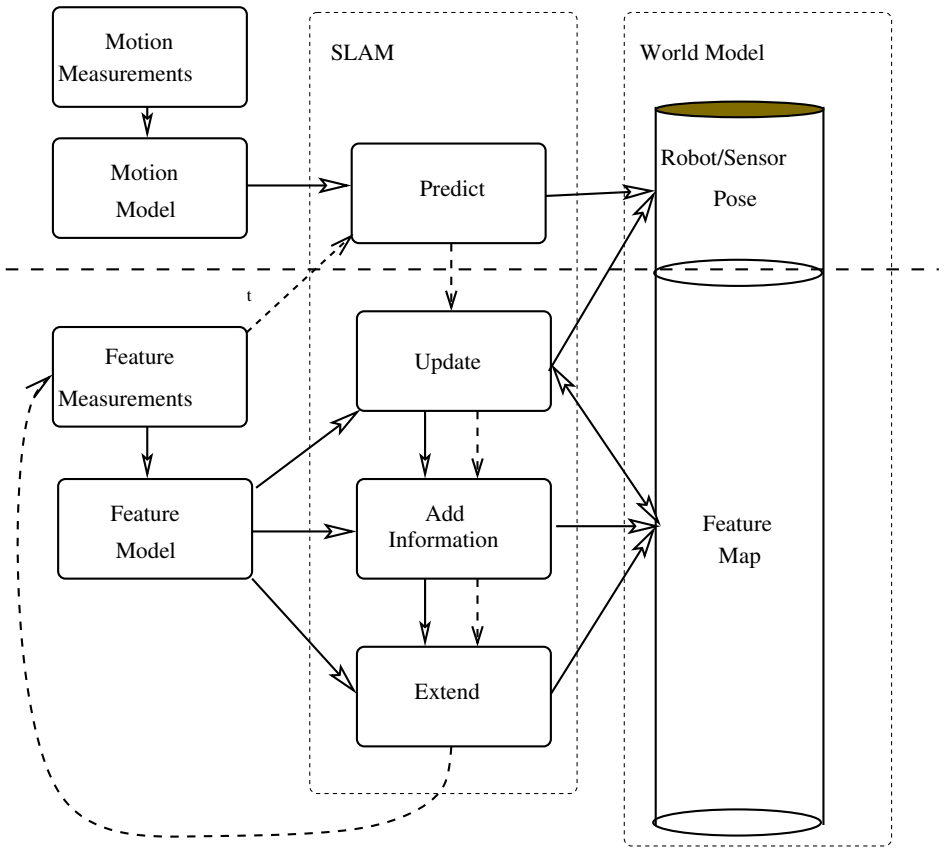

Figure 2.2: Here we show the structure of our incremental SLAM model in more detail. The dash arrows show the execution flow. New feature measurements cause a predict to bring the robot state up to the time of the feature measurements. Then the update, add information and extend can be done. The solid arrows indicate information flow. For example, the update phase must pass the match between measurements and features to the add information and extend phases. The part above the dashed line is the motion estimation part. This will effect the pose estimate and its correlations to the map. The part below the dashed line uses the feature measurements. It has three phases; In the update phase, the measurements are matched to the map features and then used to improve the world model. Then in the add information phase, information contained in the measurement but not used for the update is accumulated on the features. Then the extend phase tries to extend the map by using the information accumulated up to that time to initialize more parts of each feature on the map.

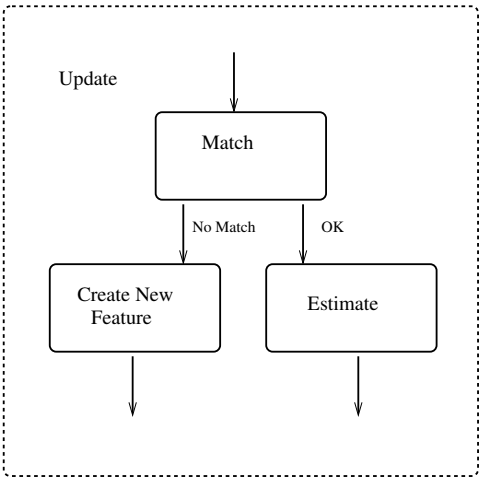

Figure 2.3: Here we show that the update phase integrates data association, map estimation and feature creation into one concept.

it might be only part of the information if the feature is partially initialized.

The forth and last phase is *extend*. Here the feature is initialized into the SLAM map if enough information has be accumulated on the feature. This can be done partially in cases where the features are only partially observable.

These four phases will be explained in more detail later.

2.2 The Stochastic Map

The idea of the stochastic map is that the solution to the SLAM problem is the one that best explains all the measurements $\mathbf{v}$, both feature measurements and dead-reckoning motion measurements. Typically 'best' is taken as the maximum a posteriori (MAP) hypothesis, confusingly the MAP map. Here we define a hypothesis h as the feature states ($\mathbf{x}_f$), robot path ($\mathbf{x}_r$) and data association h_a.

$$h = (\{\mathbf{x}_f\}, \{\mathbf{x}_r\}, h_a) \tag{2.1}$$

The MAP hypothesis is then:

$$h_{MAP} = argmax_h P(h|\mathbf{v}) = argmax_h \frac{P(\mathbf{v}|h)P(h)}{P(\mathbf{v})} \tag{2.2}$$

Now we assume that the *a priori* probability of all feature states and robot paths are equally probable. Furthermore we assume that they are independent of the data association h_a. Then we can simplify to,

$$h_{MAP} = argmax_h P(\mathbf{v}|h)P(h_a) \tag{2.3}$$

The $P(h_a)$ term is the *a priori* likelihood of a given data association. If not for this term the MAP hypothesis would equal the *Maximum Likelihood* hypothesis which is given by maximizing only the first term above. Finally, we note that since the log is a monotonic function we can take the log without changing the argument of the maximization:

$$h_{MAP} = argmax_h [\ln P(\mathbf{v}|h) + \ln P(h_a)] \tag{2.4}$$

A little later we discuss $P(\mathbf{v}|h)$, the probability of the measurements given the data association and state. What about the $P(h_a)$? This is rather hard to say much about. It is safe to assume that data associations that lead to simpler maps, (fewer features) are more likely than those that lead to complex maps. We assume an exponential dependence of $P(h_a)$ on N_f, the number of separate features matched to.

$$\ln P(h_a) = -\Lambda N_f + c_1 = \sum_{j=1}^{N_f} \Lambda(n_j(h) - 1) + c_2 \tag{2.5}$$

where $n_j(h)$ are the number of measurements associated with feature j, (c_1 and c_2 are constants that have no effect on the argument of the maximization).

Pulling this all together we have that the MAP hypothesis is:

$$h_{MAP} = argmax_h \{\ln P(\mathbf{v}|h) + \sum_{j=1}^{N_f} \Lambda(n_j(h) - 1)\} \tag{2.6}$$

The first term gives weight to well explained measurements. The second term gives a penalty for introducing new features to explain a measurement. Notice that if there is only one measurement associated with a feature we can make both terms equal to zero for that feature and measurement. If the measurement was instead associated with an existing feature the first term would be negative but the second would be positive leading to a trade off that might be better than 0.

Equation (2.6) will be the basis for estimating the map for most of this thesis. We will show how Λ is equivalent to the mahanolobis distance criteria typically used for matching with an extended Kalman filter.

We recall that $\mathbf{v}$ includes both the dead-reckoning and the feature measurements. The h includes both the data association hypothesis and the state. The n_j were the number of matched measurements to feature j and N_f was the number of features in the map. In the estimation methods we will describe here the measurements are assumed to be statistically independent over time. That is to say, the measurements at time t are independent of the measurements at other times. Also, all the dead reckoning estimates are assumed to be independent of the feature measurements.

With these assumptions (2.6) becomes,

$$h_{MAP} = argmax_h\{\sum_i \ln P(\mathbf{v}_{di}|h) + \sum_i \ln P(\mathbf{v}_{fi}|h) + \sum_{j=1}^{N_f} \Lambda(n_j(h) - 1)\} \quad (2.7)$$

Where i is the index over the measurements. The $\mathbf{v}_{fi}$ and $\mathbf{v}_{di}$ are the feature and dead-reckoning measurements respectively.

2.3 Some of the Difficulties of SLAM

SLAM has evolved into a mature discipline and a number of successful systems have been reported. It is now possible to have a robot explore an area autonomously and return with an accurate map [16, 17, 18]. The basic estimation problem is well understood. However, there are still a number of open problems to be addressed. These include computational complexity, linearization effects, association of measurements to features, detection of loops in the robot path and maintaining topological consistency as the maps get very large. We will now discuss each of these in turn.

Scalability and Computational Complexity

Computational complexity is a problem for real-time SLAM implementations. In particular some iterative SLAM algorithms have computation time increasing with the size, N of the map. For example the extended Kalman filter, EKF, has a complexity that grows like N^2. This limits the size of the map that can be made in real-time. Several methods have been proposed that give faster calculation time.

Constant time algorithms take advantage of the fact that the effect of the current measurements falls off rapidly as one moves further away from the robots current position. This can be seen by examining the inverse of the covariance matrix, the information matrix. It is well known that the information matrix tends to be nearly sparse with only very small elements connecting distant features. Thus the changes can be well approximated by local updates which take constant time. In fact, the long range correlations are often not accurately modeled anyway making the global calculations unjustified. This is due to several factors, the inexact sensor models used, the cumulative effect of linearization errors, the spurious measurements and the disturbances that the robot encounters.

The problem with the local update, constant time approach is that eventually one would like a globally consistent map. In particular on return to a previously visited region. This global map can take a long time to compute as with the CEKF [19],[18]. One way out is to only calculate the links between local maps [20]. By leaving the individual maps unchanged the update time becomes dependent on the number of maps. This is of course also a measure of the size of the global map,

but it is a much smaller number than all the features. Trying to get the individual features that two local maps have in common to agree is much harder.

Linearization and Consistency

The effects of linearization in EKF SLAM can be quite serious [21, 22]. It seems that the effects of linearization become worse as the map grows. Eventually the map becomes inconsistent. The covariance matrix tends to become overly optimistic. The solutions include particle filter approaches and local map approaches.

The particle filter can eliminate some of the linearization approximations entirely. Using samples of the non-linear distribution to accurately model the probabilities. Practical considerations limit how much of the true distribution can be sampled and some linear modeling may still be done.

Local map methods exploit the fact that the linearization effects become important as the map grows. For small maps consistency is not usually a serious problem. The problem of consistency over large distances can then be looked at separately from the consistency of the finer details.

Another consistency problem arises when the robot discovers a closed loop in its path. Having detected a closed loop one must adjust the map to reflect this new constraint. If the linearizations were done in a global frame this cannot be done consistently. The successful methods for closing loops in the map all use a local frame for doing the linearization [20, 23, 24].

Data Association

Data association is critical in SLAM. The incorrect association of measurements to features can produce very large errors to the map. In most recursive methods data association is immediate and irreversible, which can lead to catastrophic failure in noisy environments. As it is important to not throw away information the need to decide at once can lead to bad, rushed decisions. Methods that allow reconsideration of the associations can wait until it is clear that the measurements are from good features before using them. This is closely related to the problem of when to introduce new features into the map. If we wait too long we will not get to use important information from the first few measurements of the feature. If we add the feature too soon we may be using measurements that do not come from the high quality static features we want to use.

One method that allows for flexible reconsideration of data associations is the expectation maximization method, EM [25, 5, 26]. One can also modify a Kalman Filter to become a Kalman smoother [27] which allows one to wait longer before committing to measurements.

The hardest data association is when closing a loop. This is when the robot returns to a previously mapped area. If the loop is large the robot pose cannot be relied upon for data association. Thus the closest feature may be not be the correct match. In this situation it is best to consider many features at the same

time. One must try to map a patch of the area and then match the features in the patch to the older map of the area [28, 29, 30]. The Joint Compatibility Test is a good criteria for matching multiple features. This test sets a threshold on the joint likelihood of all the feature matches of a hypothesis. Another approach is to form a unique signature for each map patch and then try to match the signatures [20, 31].

Our method has similarities to [32] in that we consider the likelihood of not seeing previously seen features when returning to a region. So a hypothesis consists of the matches and the lack of observation of some features in one or the other map patch. By considering the likelihood of not having seen a feature one can avoid the mistake of not matching important features in the chosen hypothesis.

2.4 Prior SLAM Work

The SLAM problem has several dimensions. One is the basic model for the map, topological vs. metric. A topological map consists of places and a graph of connections between the places. It can be used to plan paths from one place to another but does not give the geometric relations such as distances and angles. A metric map is a geometrically accurate model of the environment. Hybrids between topological and metric maps are common [33],.

Another dimension is the representation of the map knowledge itself. This can be as raw sensor data, as a discretized grid over the unstructured space of the environment or as the locations of certain abstract features in the environment.

A third dimension is the representation of uncertainty in the map. One choice is to represent the map uncertainty by a unimodal distribution such as a Gaussian. The alternative is to represent multi-modal distributions of maps in some way. In table (2.1) we show how one could separate some of the existing SLAM methods along this dimension.

Unimodal	Multi-modal
EKF	EM
SEIF	FastSLAM
Occupancy Grid	Rao-Blackwellized Particle Filters
Thin Junction Tree Filters	Graphical SLAM

Table 2.1: The separation of some SLAM algorithms by their ability to represent uncertainty

Map Models and the Representation of Uncertainty

There has been a lot of work done on SLAM. Early on a topological maps were prevalent. In [34, 9] a cognitive map of the environment was built. This model allowed one to reason about paths between locations on the map. Uncertainty was not explicitly dealt with but rather relationships on the map were either known or not known.

In [10] the map was a graph where nodes represent places. Uncertainty estimates were done rather crudely as worst case errors. This limited the robots ability to make consistent global maps but the robot was able to navigate using the map locally.

In [11] a geometric model of the world was a complement to the topological one. There sensor uncertainties and the distinction between absolute and relative frames of reference were also considered in estimating the map.

The occupancy grid [1] is a way to both deal with geometric information and uncertain measurements. The occupancy grid models the world as a grid of cells where each cell has a probability of being occupied. An occupied cell cannot be navigated through by the robot. These probabilities can be updated using the sensor measurement models and, for example, Bayesian reasoning. Getting good results depends on maintaining a good robot localization within the grid. This in turn depends on a fine grid which can lead to high computational and memory costs.

In [2] scan matching is used to reduce the robot localization errors and thus produce good occupancy grid maps. In [3] a more realistic sensor model gave improved occupancy grid maps. Later works using occupancy grids [4, 5], included using forward models to overcome some of the simplifying assumptions about independence of sensor reading for a cell from the information in neighboring cells. Using a more correct model helps resolve some situations that the prior work could not map properly.

In [35] the basic stochastic map framework was introduced and the uncertainty estimate became much tighter than the earlier estimates. This work introduced the Extended Kalman Filter, (EKF), to the SLAM problem.

In [36, 37] the EKF was refined by introducing the SP-model (symmetry and perturbation) for feature representation. This attached a reference frame to each feature and the linearization of the measurements occurred in this feature frame. That greatly reduced the effects of linearization on the solution. In [38, 16] the EKF was more or less confirmed as a successful solution to the SLAM problem for smaller static environments with good feature sensing.

Scalability and Complexity

The EKF has a problem with scaling quadratically with the size of the map. In CEKF [2] [19, 39, 40] the individual Kalman updates are accumulated in matrices

[2] Compressed Extended Kalman Filter

the size of the local map being currently observed. Periodically these matrices can be used to update the global map. The result can be shown to be equivalent to a standard EKF. The global updates still take order N^2 time but they are done at a much lower frequency.

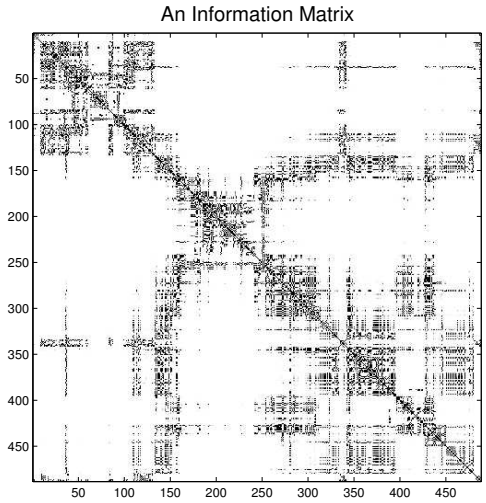

Figure 2.4: The information matrix is the inverse of the covariance matrix for the map features. It can be seen here that the matrix is approximately sparse. This is a result of the measurement process that is used to estimate the map. Measurements are made locally so the mutual information is also local.

An EKF is equivalent to the Extended Information filter, (EIF)[41]. The EIF uses the inverse of the covariance matrix, the information matrix (see figure 2.4), and a linear combination of the measurements to do constant time updates. The problem is that finding the mean of the distribution takes cubic time in the size of the map[3]. This mean state is needed to match the measurements to the state or to use the pose estimate of the robot for any task.

An observation that the information matrix is approximately sparse, (see figure 2.4), has lead to the Sparse Extended Information Filter (SEIF) [42, 43]. In order to achieve constant time updates exact sparseness is imposed on the information filter. This is done through a procedure which imposes independence between the current pose and more distant features. After doing so the results are no longer equivalent to the EKF. For the exactly sparse system the mean can be solved for in constant time by a method of relaxation. The system can be represented by a Gaussian graphical model where the edges are then sparse.

[3] A linear equation with the information matrix must be solved.

A method very similar to SEIF uses a thin junction tree filter, [44]. This starts with the Gaussian graphical model of the EIF and builds up a mechanism of message passing for the purpose of making inferences about the map. The model has only been tested in simple simulations and not on real data.

Linearization and Consistency

The more serious problem of the EKF is linearization [21, 22].. One way to address this is by Rao-Blackwellized particle filter techniques such as FastSLAM [45, 46, 43, 47, 48, 49]. The idea is to decouple the problem of finding the robot path from the map estimation problem. By introducing particles for a sample of the possible paths and then finding the best map for each of these paths one can avoid the linearization problem completely. Solving for the map given the path is rather easy. One can then calculate the likelihood of the measurements given the solution generated by the particle. If this is too low one can safely discard the particle reducing the computational burden. The main problem is that one must have relatively tight constraints on the path from the measurements in order to hold the number of particles to something manageable. For weaker measurements constraining the robot motion the number of particles needed can be very large. There is also no good way of determining beforehand how many particles are enough for good estimates.

A variation on these is DP-SLAM [50]. This uses a clever representation of the map as an occupancy grid and a tree on each particle. Each particle is a leaf in a ancestry tree rooted at the initial particle. The particle keeps a record of just what changes it made to the occupancy grid. Its ancestors can then provide the other information needed to complete the grid. With this representation a larger number of particles can be handled.

For SEIF the linearization leads to inconsistency as in the EKF but here the loss of information during the sparsification step worsens the inconsistency as shown in [51]. There they present a sparsification method that does not suffer this inconsistency but requires cubic time to calculate.

Data Association

A major problem in SLAM is the data association. How to connect the measurements with the map element that caused it. This is a problem for each iteration matching to the recently observed features and for closing large loops where there can be very large errors in the features relative to the robot. One solution proposed is expectation maximization, EM, [25, 5, 26]. This method repeats two phases of computation. First in the E step, it calculates an estimated probability density over robot paths given the current map. Then in the M step, it calculates the maximum likely map using this probability density. Since it explicitly considers all possible data associations when doing the E step, it has a better chance to find the correct one. The main problem is the large search space that results.

The Kalman smoother [27] tries to address the problem of rushed data association decisions in the EKF by maintaining a chain of the most recent robot poses in the Kalman state vector. By having the older poses explicit in the state one can add measurements relative to these poses after having seen the later measurements.

The laser range scanner has lead to a very successful method of pose estimation based on scan matching [6, 23, 2, 7]. In the IDC method individual scan points are matched to a reference scan by alternating between estimating the transformation between scans and the match between individual points. The covariance of the resulting pose estimate is estimated in [8]. This method can be used to close very large loops by imposing the topological constraints on a graph formed by the scan matches.

For detecting closed loops the joint compatibility test of [28, 29] is one excellent solution when the state covariance matrix is available. The covariance matrix needs to be consistent for this to be a correct test. Such consistency is not achievable with the EKF and many other methods do not explicitly calculate the covariance matrix.

Sub-map Methods

The problem of building very large yet consistent maps has lead to many sub-map approaches to SLAM. In [52] groups of features form a landmark with its own local frame of reference. These landmarks can then be included in a global map. In [53] local maps are similarly assigned a root based on some landmark. This root can then be shifted to another landmark that is better known globally, leading to a convergent global map estimate. Similar sub-map approaches are shown in [29, 54, 55] as in the ATLAS framework [20] which was a more general way to join arbitrarily built sub-maps into a global map.

These sub-map methods can close loops in the global map by imposing constraints on the transformations between the local frames of reference [23, 20]. The global map typically becomes a graph with local maps as the nodes. These are linked by some type of springs and the constraints can be imposed on the system by finding the equilibrium position of the poses under the action of these links and the constraints.

The idea of using a graph model to do SLAM estimation has been gaining popularity [56, 44, 57, 58, 59, 60]. There have also been a number of hybrid approaches to SLAM [61, 62, 63].

Vision SLAM

Most of the works cited above use a SICK laser scanner or sonar to build the map. There has also been much work done using vision. In [64] maps are made using both a laser and a camera. In [65] a method similar to scan matching is developed for camera image data. They used an omni directional camera and a Kanade-Lucas-Tomasi tracker to detect features. In [66] tracking three features from frame

to frame is sufficient to obtain a good pose correction for the robot. They solve the constraint equations directly to get the transformation between consecutive frames. They depend on tracking the landmarks from frame to frame in order to make correct data associations.

In [67] a stereo camera system provides 3D clouds of points which are fit into an octree map structure. The robot motion is again solved by using the constraints form the camera images. No other sensors are used. They solve first for the motion and then update the octree from the calculated camera pose. This is similar to using laser scan matching to correct odometry and then filling in an occupancy grid. It does not explicitly deal with the coupling between robot pose and map uncertainty. The distances covered are therefore limited to just a few meters.

In [68] SIFT features and a three-camera system are combined with an EKF to make maps. In [69] an EKF is used with a chi-square data association test to do bearing only SLAM. In [70] a combination of sonar and vision on an underwater robot is also used as input to a EKF. They use a Lucas and Kanade feature tracker and do not try to close loops when the features return to the field of view. In [71, 72] a single camera is also used with an EKF to do SLAM. There no other sensor is used and the system can close small loops with difficulty. A particle filter is used to solve the difficult feature initialization problem of bearing only SLAM.

In [73, 74] SIFT features are used to do so called vSLAM. The problem of learning good features from image data was addressed in [75] using a feed-forward back-propagation neural network.

Our work vs. Prior Work

Our method tries to combine the best parts of all these methods in one approach. We have independence between the size of the map and the calculation time for each iteration. Very similar to SEIF without the loss of any information. We linearize in a special way so that the non-linearities can never be large and all symmetries are preserved. Global constraints are imposed in a way very reminiscent of the method of Lu and Milos. We have a mechanism that can be used similarly to EM for data association. The final result is much like a network of local maps.

Chapter 3

Robot Motion Model

The SLAM prediction process starts with the raw measurements. These are the output of some sensor. These all have some errors that we will need to estimate in order to correctly evaluate equation (2.6).

3.1 Dead-reckoning

Dead-reckoning is an estimate of the robot pose from adding up the small changes from the incremental estimates of the motion. This is contrasted with absolute estimates of pose which are relative to some landmark such as GPS satellites or some external force such as gravity or the magnetic field of the earth. Dead-reckoning estimates have the property of ever increasing errors. This can be seen by examining the error propagation in the simple illustrative case of no angular errors. If the position of the robot at time $i+1$ is predicted from the position of the robot at time i,

$$\mathbf{x}_{i+1} = \mathbf{x}_i + R_i^{-1}(\Delta\mathbf{x}_{i+1,i}). \tag{3.1}$$

Where R_i is the rotation matrix for the robot pose and $\mathbf{x}_i$ is the position vector of the robot at time i. The $\Delta\mathbf{x}_{i+1,i}$ is the incremental motion in the robot frame at time i. Then if the incremental noise is Gaussian with covariance only in position, (ie orientation errors are 0), $C_{i,i-1}$ then,

$$C_{i+1} = C_i + R_i^{-1} C_{i,i-1} R_i^{-1}. \tag{3.2}$$

We see that the estimated covariance in position, C_i, is growing each iteration. Similar results hold if the covariance of the angles of the rotation matrix are included.

3.2 The Motion Sensors

In order to estimate the errors in our motion prediction we first need to model the sensors.

Odometry

Raw odometry data consists of the counting of the rotation of the robots wheels. This rotation is typically measured quite accurately. The errors occur due to the inexact relationship between wheel rotation and the movement of the robot. This is inexact due to wheel slippage, variations in wheel diameter/tire inflation and any incline in the ground surface. These errors typically mask the quantization error of the counters measuring the rotations.

The raw odometry is integrated at a relatively high frequency producing an estimate of (x, y, θ), the change in position and orientation, pose, since the robot was turned on. This pose is not accurate enough to use over longer distances but the incremental changes to it are reasonably accurate over short distances. Over the distance the robot travels between frames of the camera or scans of the SICK laser scanner the errors are small enough to be modeled as Gaussian noise around the incremental odometry value.

The model of this noise can be derived from studying the processes that give rise to it. Such studies [76] conclude that the covariance of the error should increase proportional to the distance traveled Δs and the angle turned $\Delta\theta$. Based on that we form a simple model and fit the parameters to it. The model is found by considering the motion during the increment as movement on a circle of radius r, figure 3.1,

$$r = \frac{\Delta s}{\Delta\theta} \tag{3.3}$$

If we then look at the change Δx, Δy in the robot frame at the start of the interval, (the robot pointing in the x direction), then we find:

$$\Delta x = r \sin \Delta\theta \tag{3.4}$$

$$\Delta y = r(1 - \cos \Delta\theta) \tag{3.5}$$

These give rise to the following Jacobian of $(\Delta x, \Delta y, \Delta\theta)$ with respect to $(\Delta s, \Delta\theta)$

$$J = \begin{pmatrix} \frac{\sin\Delta\theta}{\Delta\theta} & \Delta x \frac{\Delta\theta - \sin\Delta\theta}{\Delta\theta \sin\Delta\theta} - \Delta y \\ \frac{1-\cos\Delta\theta}{\Delta\theta} & \Delta x - \frac{\Delta y}{\Delta\theta} \\ 0 & 1 \end{pmatrix} \tag{3.6}$$

with the appropriate limit taken if $\Delta\theta$ becomes small.

$$C_{odometry} = J C_{raw} J^T \tag{3.7}$$

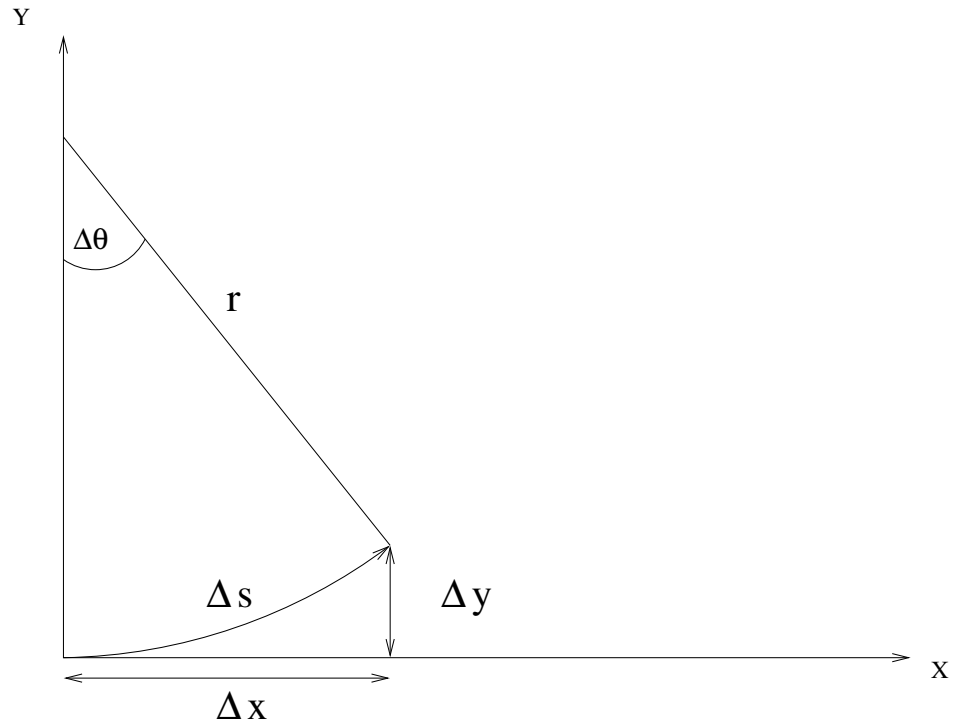

Figure 3.1: The motion of the robot over short time intervals can be approximated by motion along a circular arc.

$$C_{raw} = \begin{pmatrix} a_1|\Delta s| & 0 \\ 0 & a_2|\Delta\theta| + a_3|\Delta s| \end{pmatrix} \tag{3.8}$$

Here the a_i, are the free parameters of the model. One can of course use a more complicated model than this but this simple model works well with one adjustment. The covariance $C_{odometry}$ of $(\Delta x, \Delta y, \Delta\theta)$ above has rank two. This will lead to a very high correlation between the y error and the θ error over longer distances. The correlation will be approximately correct but not desirable due to the linearization effects. The non-linear system is not as strongly correlated. For this reason it is often desirable to add a small variance (about 10^{-6} m^2per m) in the y direction. This prevents a fully correlated covariance.

Inertial Sensor

The ATRV outdoor robot we use for some experiments was equipped with a Crossbow DMU-FOG 6-axis inertial sensor [77]. This device contains 3 accelerometers

and 3 fiber-optic gyros. The accelerometers were filtered internally and fused with the readings from the gyros to give an estimate of the vertical direction. The pitch and roll angles output from the device were adjusted to be correct over longer time periods by agreeing with gravity and over shorted time periods by integration of the gyro's angular velocity.

Thus the errors in pitch and roll were small and proportional to the angular velocity in those directions. For our experiments we took these errors to be 0. We thus use the full 6 dimensional robot pose but only consider errors in (x, y, θ).

The gyros also provide an integrated angle, θ, around the vertical z-axis. This angle estimate will be fused with the odometry angle estimate to obtain an better orientation dead-reckoning estimate. The covariance in this angle is proportional to the time interval.

The inertial sensor will have significant bias which must be corrected for. There is an internal heater to maintain a constant temperature in the sensor as the bias is temperature dependent. It is important to not begin using the sensor until it warms up. At that time an internal bias adjustment routine should be run with the robot at rest. Even after this we found some measurable bias in this sensor.

3.3 Fusion of Odometry and Gyro

The ATRV robot has a skid steering system with four large wheels that all point in the forward direction. In order to turn, the wheels on each side are driven at different rates which produce a large amount of wheel slippage. As a result the orientation angle from odometry while good when driving straight ahead is very poor when turning. We therefore fused the inertial yaw angle with the θ estimate from the odometry. This was done in such a way that the odometry angle was given a high weight while driving straight. When turning the inertial angle was given a high weight. This inertial angle is very accurate over short time intervals. The resulting dead-reckoning estimate was much better than either odometry or inertial estimates alone.

$$\Delta\theta_{fused} = w\Delta\theta_{odometer} + (1 - w)\Delta\theta_{inertial}. \tag{3.9}$$

$$C_{fused} = WC_{odometer} * W + (1 - W)C_{inertial}(1 - W). \tag{3.10}$$

Where if θ is the third row of C then

$$W = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & w \end{pmatrix}. \tag{3.11}$$

The two angle estimates were weighted by minimizing the covariance of the result.

$$w = \frac{C_{inertial\Delta\theta,\Delta\theta}}{C_{odometer\Delta\theta,\Delta\theta} + C_{inertial\Delta\theta,\Delta\theta}} \tag{3.12}$$

One complication was that the inertial yaw angle was in the earth frame while the odometry angle was in the frame of the robot at the beginning of the interval. As we have the pitch and roll angles we can rotate the inertial estimate and its error into the frame of the robot and then combine the odometry and inertial estimates of $\Delta\theta$.

Test of the Dead-reckoning Models

To evaluate our models we can compare the incremental change in the orientation, $\Delta\theta$, for the path obtained using a hand made map and an extended Kalman filter localization. The θ changes from odometry are subtracted from the localizer values and the result divided by the square root of the sum of the covariance estimate from the odometry model and from the localizer.

This comparison is not entirely correct as the path from the localization program is not perfect and is correlated to the odometry, inertial and fused estimates that contributed to the input of the EKF. If we could assume that the EKF estimate only depended on continuously observing walls from a known correct map using the SICK laser then the path would be both essentially correct and uncorrelated to the other measurements. The actual results are distorted to the extent that this is not the case.

When there are no walls in front of the robot the EKF is simply using the fused dead-reckoning and the correlation is 1. In this case the error is underestimated, (ie. it is nearly zero). Then when a wall comes into view the estimate can get a large correction which produces a large error estimate when the true error was in fact no larger than average. Thus one expects the estimated distribution to be more peaked near zero and to have more outliers than the true error distribution.

This is the qualitative appearance of the three histograms as compared to the assumed Gaussian distribution shown in figures 3.2, 3.3 and 3.4. We see that the correlation effects are most significant for the fused data that was the direct input to the EKF Localization.

Aside from these considerations the histograms do indicate that the covariance estimates are not terribly optimistic or pessimistic. We can also see that the fused estimate is substantially better than the odometry or inertial estimates alone. This is evident as well when we look at the path obtained from the fused estimate as compared to raw odometry figure 3.5 verses figure 3.6.

3.4 Summary of the Motion Model

We start with the raw error in the incremental odometry angle and distance:

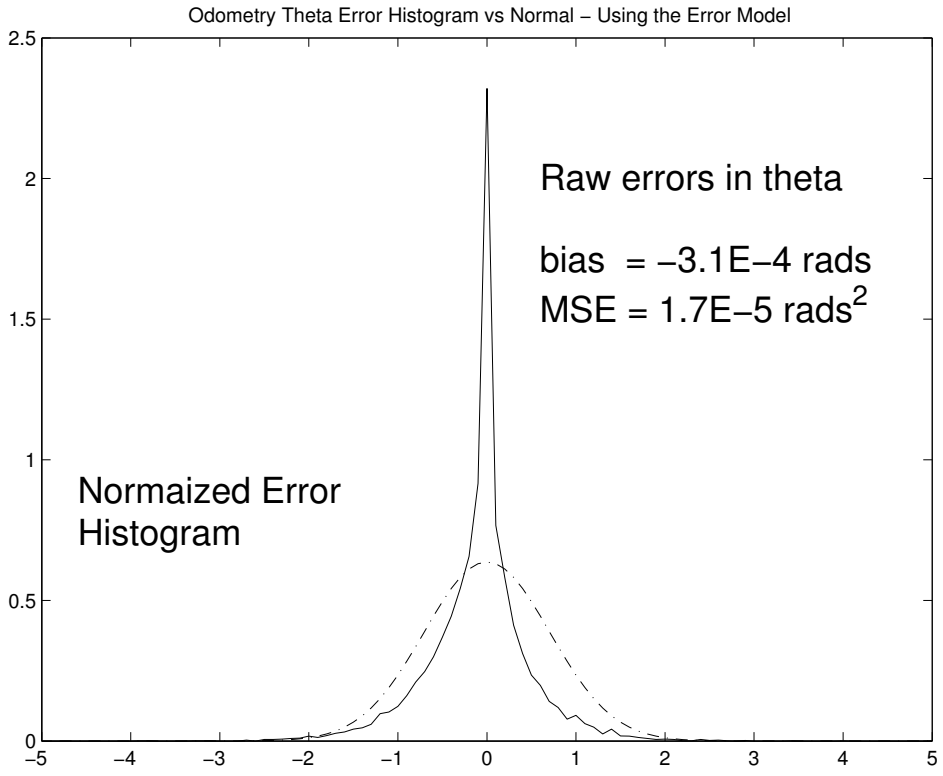

Figure 3.2: This shows an estimate of the odometry angle error over each of the 8,742, 200 ms intervals. The error has been normalized by dividing by the standard deviation as given by the models. There are some distortions due to the lack of a true ground truth path as discussed in the text.

$$C_{raw} = \begin{pmatrix} a_1|\Delta s| & 0 \\ 0 & a_2|\Delta\theta| + a_3|\Delta s| \end{pmatrix} \tag{3.13}$$

We then project that into errors in (x, y, θ) in the robot frame at the start of the interval with the Jacobian:

$$J = \begin{pmatrix} \frac{\sin\Delta\theta}{\Delta\theta} & \Delta x \frac{\Delta\theta - \sin\Delta\theta}{\Delta\theta \sin\Delta\theta} - \Delta y \\ \frac{1-\cos\Delta\theta}{\Delta\theta} & \Delta x - \frac{\Delta y}{\Delta\theta} \\ 0 & 1 \end{pmatrix}. \tag{3.14}$$

The odometry covariance estimate becomes:

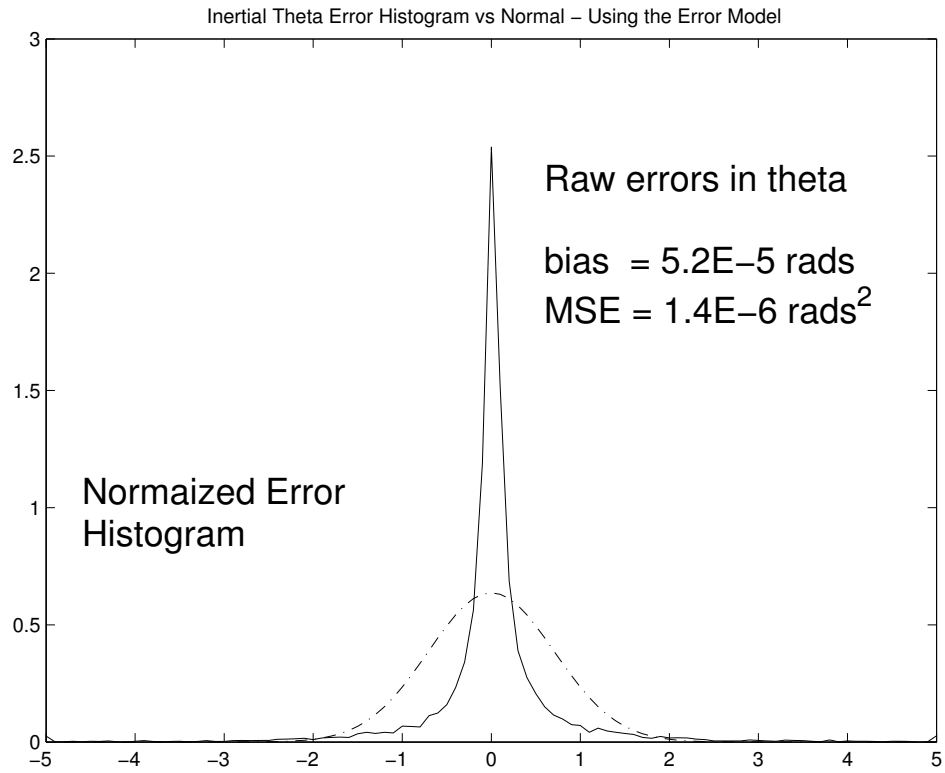

Figure 3.3: This shows an estimate of the inertial angle error over each 200 ms interval. The error has been normalized by dividing by the standard deviation as given by the models. Notice that this sensor has a bias of $2.5E-4$ rad per sec., (The value shown $5.2E-5$rads/.2sec= $2.5E-4$ rad/1 sec.)

.

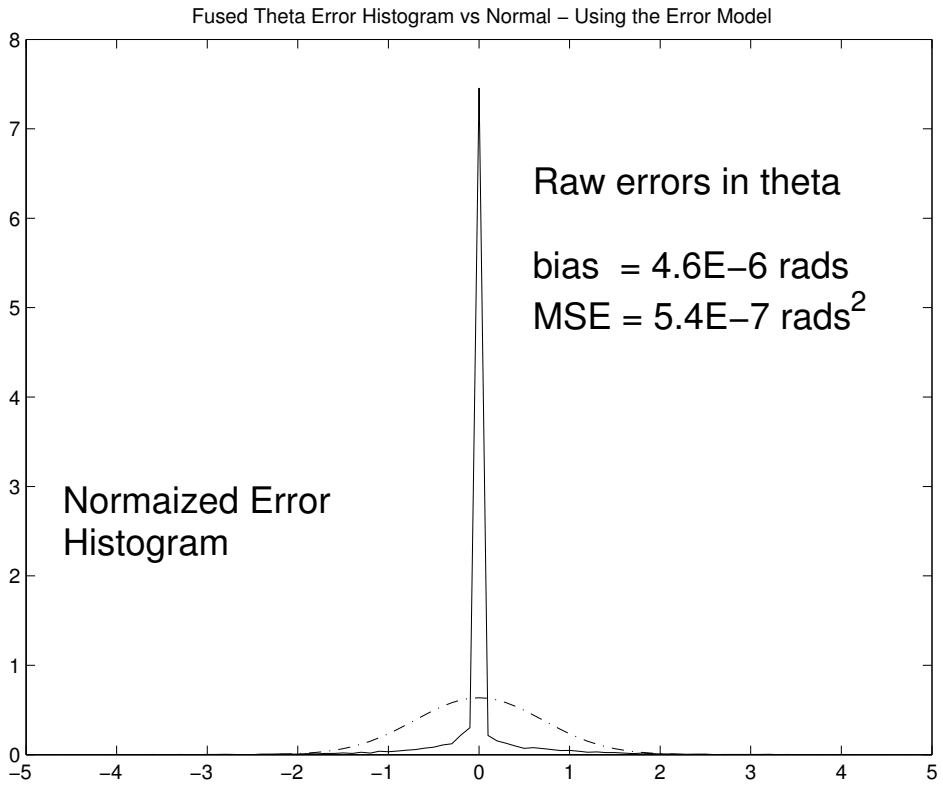

Figure 3.4: This shows and estimate of the fused odometry and inertial errors. A substantial improvement can be seen as compared to both odometry and inertial. The shape of the histogram is steeper near zero and the tail is longer due to the relatively strong correlation between the 'true path' and the fused estimate. Consequently, this histogram tells us less than the histograms comparing the odometry and inertial estimates to the 'true path'.

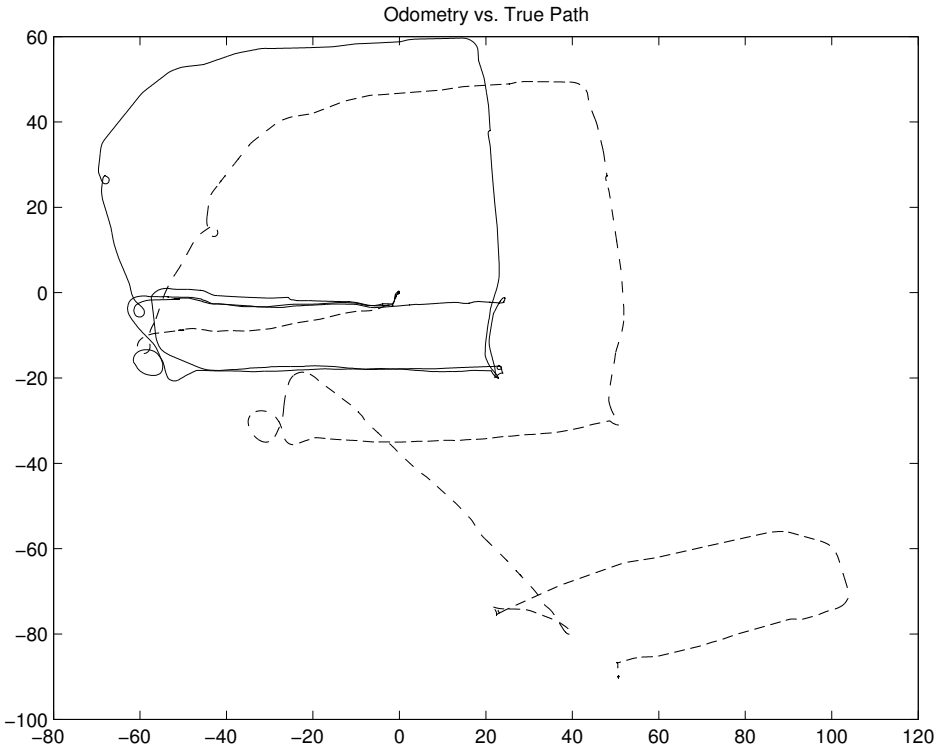

Figure 3.5: This is a typical comparison between raw odometry data (dashed) and the true path of the robot (solid). The pose error is growing without bound. The distances shown are in meters.

$$C_{odometry} = JC_{raw}J^T. \tag{3.15}$$

Weight the θ estimate with that obtained from the inertial sensor using the weight w:

$$w = \frac{C_{inertial\Delta\theta,\Delta\theta}}{C_{odometer\Delta\theta,\Delta\theta} + C_{inertial\Delta\theta,\Delta\theta}} \tag{3.16}$$

$$\Delta\theta_{fused} = w\Delta\theta_{odometer} + (1-w)\Delta\theta_{inertial}. \tag{3.17}$$

$$C_{fused} = WC_{odometer} * W + (1-W)C_{inertial}(1-W). \tag{3.18}$$

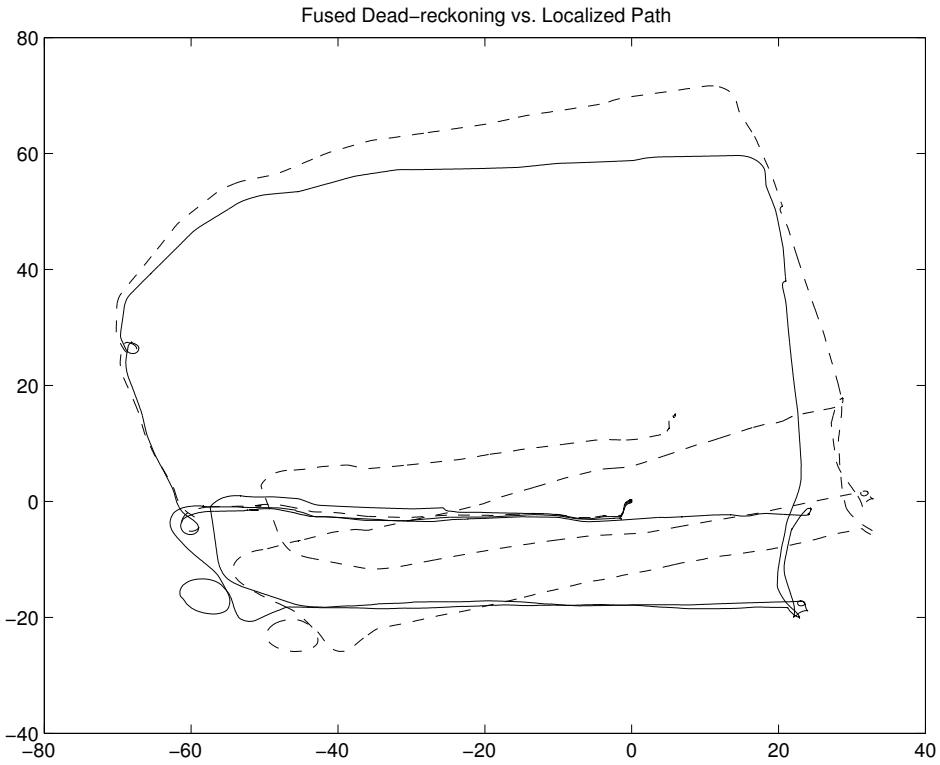

Figure 3.6: This shows how by fusing the odometry and inertial estimates (dashed line) we can get a dead-reckoning estimate far better than odometry alone. The pose error is still growing without bound. The true path is shown as the solid line. The distances are in meters.

robot	a_1 (m^2/m)	a_2 (rad^2/rad)	a_3 (rad^2/m)
ATRV	0.0005	0.0025	0.000001
PeopleBot	0.001	0.002	0.00004
Pioneer	0.001	0.002	0.00004
PowerBot	0.001	0.002	0.0000004
Custom Built	0.001	0.0005	0.0000004

Table 3.1: This lists the typical odometry model parameters used for the experiments in this work.

Where

$$W = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & w \end{pmatrix}. \tag{3.19}$$

and $C_{inertial} = 1E-6\ (rads^2/sec)\ \times \Delta t(sec)$.

Chapter 4

Features for Localization

The features are geometric landmarks designed to provide localization information for the robot pose [78, 79, 80]. In this chapter we start with a discussion of the sensors, SICK laser and camera, used to detect the features and then discuss the representation of the features.

4.1 The Feature Sensors

We cannot begin to discuss the problem of feature representation without first considering how the features are measured. Particularly since our representation is designed around these measurements.

SICK Laser Scanner

The SICK laser scanner is a sensor that can measure the distance to objects over a 180 degree arc, figure 4.1. This is done by shining an IR laser on a rotating mirror. The mirror deflects the light out from the device. The light then strikes an object in the environment and is reflected back to the mirror and into the sensor where it is detected. The time delay between the emitted and reflected light is measured to high precision [81, 82, 83]. The resulting scan lies in a plane and, in this work[1], consists of either 181 or 361 range values depending on whether the full or half degree scan spacing is selected. The maximum distance is either 8 or 32 meters for mm mode or 82 meters for cm mode. Outdoors[2] the cm mode is preferred as the distance to objects can be quite large. Indoors[3] the better resolution of mm mode is the sensible choice. Note that in cm mode the range values are in whole cm and have therefore 5 mm quantization errors. The finite beam width of the laser is also

[1] The SICK scanner has more possible modes than those used in this work
[2] LMS291 SICK Laser is used outdoors as it has greater power and thus longer range.
[3] the LMS200 is used on all our indoor robots

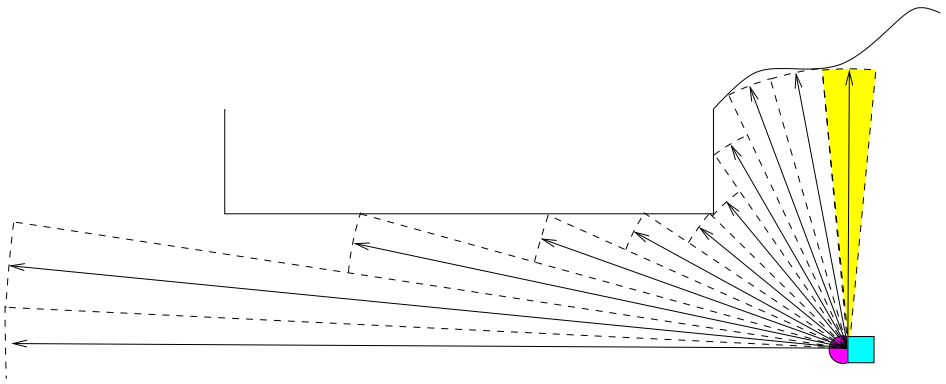

Figure 4.1: An illustration of the SICK laser scanner showing half of a complete scan, one quadrant. The beam width has been exaggerated for illustration and the number of range measurements per scan reduced.

more pronounced on the higher power laser used on the outdoor platform. Left uncorrected this beam width can lead to significant bias in the measurements.

When the scanner is being moved rapidly during the scan, 181 whole degree scan mode is to be preferred as the 361 half degree scan mode is done in two revolutions which then are stretched over twice the time interval. Thus the adjacent range readings have a significant timing difference. This effect is not a problem at the speeds in this work and we use both settings. Using whole degree mode results in twice as many scans and thus more computations. Alternatively, one could correct the scan points fitted to the line for the the time differences between them using odometry.

The angles of the scan can jitter around the nominal position a bit as the rotating mirror drifts in and out of sync. This effect is much smaller than the beam width effects which can be as wide as 1 degree in some situations see figure (4.1).

There are a few problems with the SICK sensor. One is that it is not unusual for bright sunlight to blind the sensitive photo detection circuit of the device. When this happens there is no alternative but to reboot the sensor by sending a reset command. This takes some seconds during which the scanner should be repositioned away from the sun.

In addition, dark and/or shiny cars can be totally invisible to the laser. Not enough light is reflected back into the device for a measurement. Glass and mirrors are obviously also a problem. This should be understood by anyone relying on a SICK for obstacle avoidance. Some second sensor is needed such as sonar or camera.

As indicated the finite beam width of the laser can produce a bias. The problem is that it depends on the angle of reflection, the reflectivity, the range and the power of the laser. The effect is not large enough to warrant an attempt at modeling all

of these effects and we found that assuming a constant beam width for each type of SICK scanner worked adequately well.

We extract lines from the scan [84] using a range weighted hough transform, see Appendix (C) for the details of the extraction algorithm. We optimized this algorithm for the characteristics of the SICK scanner. Thus the accumulator cells are sized for maximum efficiency without losing any lines. The hough transform works on the principle of voting. The line hypotheses are parameterized by perpendicular distance from the origin (the scanner's mirror), and the angle between the line normal and the x-axis. Each scan point is consistent with many such hypotheses and votes for each one with a voting strength equal to its range value. The reason for weighting the votes is that further range values correspond to longer length sections of the line.

We do the hough transform in two stages. First a very coarse one that has large cells. This gives us an idea of where to look harder. The cell with the most votes is subdivided into finer cells and those are then filled by the scan points from the original large cell. The maximum fine cell is then tested by forming a least square fit line of its scan points. After throwing away any outliers and applying continuity, length and number of points thresholds, we remove the resulting scan points associated with the found line from the other accumulator cells. We can then repeat the procedure until we have found all the significantly long lines. The result is a repeatable line extraction that is as fast as schemes that rely on random sampling to speed them up.

We adjust each point to the line based on its angle of incidence and the beam width, see figure (4.2). This is to remove any bias from the angle estimate. Thus a constant (.02 rad for the LMS291 and .01 rad for the LMS200) beam width is assumed and the wall is assumed to be smooth with the parameters found. It is then trivial to determine whether the upper, lower or middle of the beam wedge hits the wall first. The angle of the range reading is then adjusted accordingly. We then re-fit the line parameters to a least squares from the new points.

After finding a set of points that form a line we need to estimate the uncertainty in the line endpoints. This starts by calculating the statistics of the point set. The sum of square distances of the points from the line plus some small sensor variance [4], $\sigma^2_{sensor,i}$, for each point is divided by the number of points less 2 (2 parameters are estimated from the point cloud, the angle and distance of the line). This is then the covariance of the endpoints in the normal direction. In the direction tangent to the line we use the bigger of two error sources, the beam width of the laser and the separation between the points at the end of the line.

[4] The sensor variance is taken as $1mm^2$ in the range and $1mm^2$ per m^2 of range in the direction perpendicular to the scan. This is smaller than the actual sensor errors but is not meant to model the errors. Instead these are used to inflate the statistically measured variance of the actual point cloud in situations where there are not enough scan points to give reliable variance estimates. Thus they make the line covariance estimate slightly conservative on average but avoid the random seriously underestimated variance.

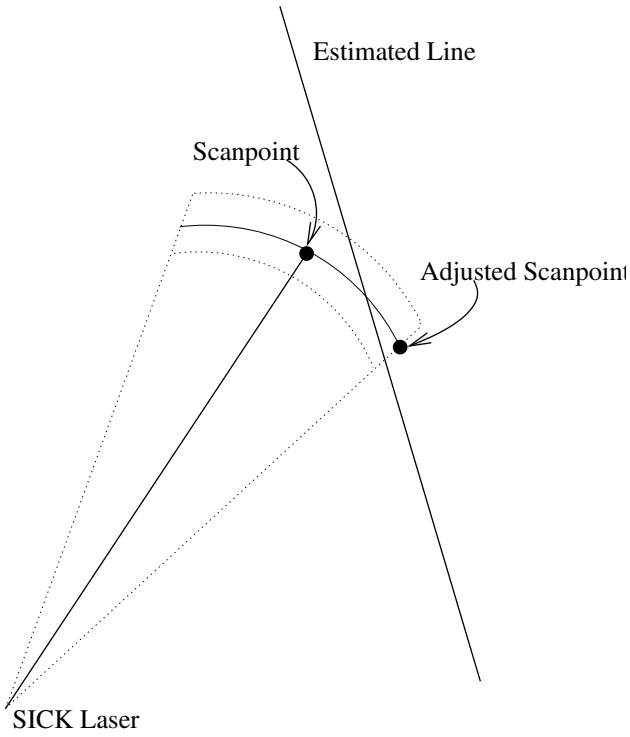

Figure 4.2: The SICK laser range point angle is adjusted to agree with the assumptions of a smooth line and a finite constant beam width. In the situation illustrated the right side of the beam will be the first to be reflected back to the scanner.

$$\sigma_\rho^2 = \frac{\sum_{i=1}^{N}[\sigma_{sensor,i}^2 + (\Delta\rho_i)^2]}{N-2}. \tag{4.1}$$

Here N is the number of scan points and $(\Delta\rho_i)$ is the perpendicular distance of the scan point to the line.

Additionally, we check if the end was unambiguously detected. By that we mean that we could see that the line did not extend any further than the endpoint,see figure (4.3). In practice this is a requirement on having seen points behind the line beyond the endpoint. Such detections help to better characterize the wall.

Cameras

Image data from a camera are also used for feature extraction. The well known pin-hole camera model is used. For recovery of metric data there is a need to

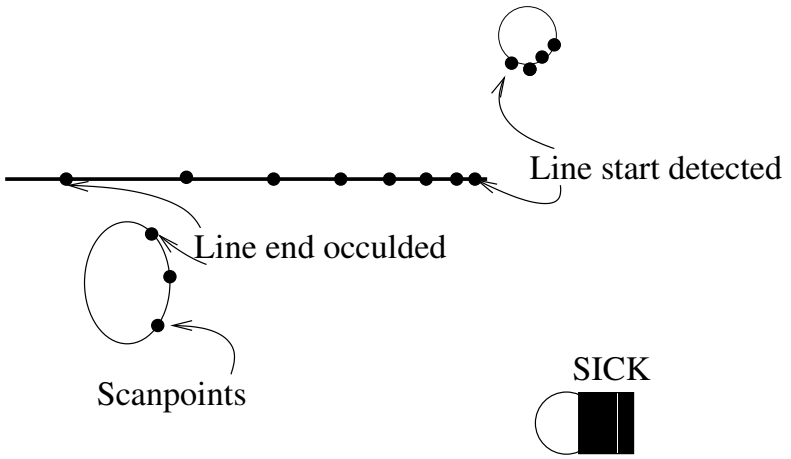

Figure 4.3: The endpoint of a line can be detected if the SICK scanner can see behind the line beyond the endpoint. Here the start of the line can be detected but not the end.

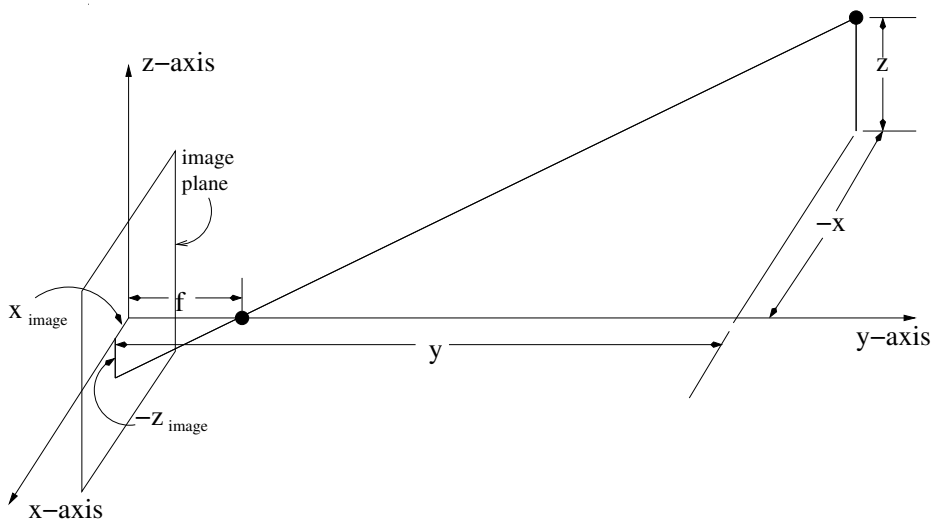

Figure 4.4: The camera is pointing in the y direction and the center point of the image plane is at the origin.

calibrate the camera. The Camera Calibration Toolbox for Matlab package [5] is used to estimate the focal length and image center pixel of our camera. We found that the skew for the camera used was not significant and the pixels were square. The coordinate system is shown in figure 4.4. As shown the y-axis is equal to the camera optical axis and the 'pinhole' at a point (0,f,0) where f is the focal length. The image center is subtracted from the pixel values to obtain the x and z values in the figure. The choice of the y-axis along the optical axis will assure us that the Jacobians of the transformations from the robot to the sensor frame will not become singular. The singularity now corresponds to a rotation in the robot frame of 90 degrees about the optical axis. That is something we can more easily avoid doing.

The image coordinates of a point at (x,y,z) in the camera frame will be:

$$x_{image} = -\frac{xf}{y-f} \tag{4.2}$$

$$y_{image} = 0 \tag{4.3}$$

$$z_{image} = -\frac{zf}{y-f} \tag{4.4}$$

Where we have converted all pixels to meters in a right handed coordinate system with the image plane lying in the x-z plane. The error sources are then the quantization error of the pixels and the error in our estimates of the focal length and image center pixel. The estimate of the image center is particularly important as any error in it will produce a bias in all our measurements. This error is essentially indistinguishable from an error in the camera orientation.

As a camera only gives us relatively accurate bearing information to the features it is more sensitive to orientation errors than the laser scanner. This orientation is rather hard to estimate accurately. We tried two approaches, one was to drive the robot using the SICK laser to localize and manually entering in matching pixels in images taken from different locations. These then gave an overdetermined set of constraints on the camera orientation. We could solve this by taking the pseudo inverse.

As an alternative method to measure the camera orientation we include the orientation parameters in an EKF SLAM filter and then drive the robot around using vision features. The filter then converges to the orientation parameters of the camera.

Both methods gave satisfactory results in the end but the estimation problem of these parameters can be ill posed depending on the movement of the camera between observation of the vision features.

The camera is also much harder to use for doing SLAM than the laser scanner because one is faced with the problem of not being able to initialize a feature until the movement of the camera allows triangulation of the feature to approximate its

[5] available at *http : //www.vision.caltech.edu/~bouguetj/calib_doc/*

distance from the camera with a Gaussian distribution [71]. In order to allow fast initialization one needs the camera to be pointed perpendicular to the direction of motion. This then introduces a problem related to the narrow field of view of the camera. With the camera pointed perpendicular to the direction of travel, the features will disappear from view quite quickly.

For these reasons it is generally better to have several cameras or an omni-directional mirror system to give long tracking times for features and quick initialization. In our work we have opted for quick initialization of high quality features which do disappear from view quickly. We can make the observation that this is not ideal.

4.2 Feature Representation in the M-Space

The first step of automatic map making is representing the features in a way that a computer can work with. The features have a variety of characteristics some of which determine its size, position and orientation in the environment. Others give more qualitative information about the feature such as texture or color. It is the geometric information about the feature that we need to represent in the map. It is this information that will assist in localization of the robot. The other information can be helpful for uniquely identifying the feature. There are a number of ways of representing the geometric information of the features. A central issue is the way that the feature will be measured. If the measurements are orthogonal to some changes in the parameters of the feature then this invariance should be explicit in the representation. Otherwise we may move the feature in these directions without any support from the measurements but rather due to some of the approximations made by the SLAM algorithm

To enable use of a generic estimation framework it is necessary be able to treat the features in a uniform way. By this we mean that the code that does the estimation of the pose and map should not need be modified for each new feature. All features should be treated the same.

We also need to consider the process of initialization of the features in the map. This is rather tricky as the geometric information on the feature will typically not be collected at the same rate for all directions[85]. For example we might learn the angle and perpendicular distance to a wall long before we can be sure of where the ends of the wall are.

Therefore, we need a way to deal with partial initialization of the feature. By this we mean that some directions can begin to be used while we wait to collect information to initialize the other directions. This collecting and testing the information is specific for each feature and needs to be part of the feature specification. We call this information that is used by the feature prior to initialization *dense information*. It is used to estimate the parts of the feature not yet initialized in the SLAM algorithm. It is also tested to see if the feature can be extended in the SLAM algorithm to include these other directions. We refer to this as the fea-

ture growing dimensions or being extended. We will call the number of initialized dimensions as the *P-dimension*, (*p-dim*), of the feature.

The *dense information* can be clouds of scan points, sets of bearing vectors, occupancy grids, sums of Gaussians or some other type of 'raw' measurement data. For the example of walls and the laser scanner this cloud of points is first tested to see if it has a well defined direction with a small enough variance normal to the line, a sufficient density of points and that it is long enough. If it passes these criteria it can be extended from *p-dim* 0 to 2. These two dimensions correspond to the angle and perpendicular distance to the wall. The endpoints are set by the extent of the cloud, but not initialized. Each new iteration will add new *dense information* which will be used to adjust the ends along the line. When one endpoint is clearly seen in the raw scan the endpoint can be initialized and the *p-dim* increased to 3.

Another property of features that we need to have in our representation is the sharing of some dimensions with other features. So to take the example of walls again, two walls can share the same endpoint, a corner. If this relationship is explicit in our representation we will not have to enforce it separately.

Finally the transformation rules of the features under coordinate transformations must be well defined. This is because the observations of these features are to be made from a moving robot. The transformation of the features to this moving frame will be of central importance in SLAM.

We have developed a feature representation the "measurement subspace" or simply the M-Space [86, 87]. There are 3 types of coordinates and a feature may contain several of each type. There can be $3D$ points, $2D$ points and scalars. These have well defined transformation rules under coordinate transformations.

We define $\mathbf{x}_f$ to be this set of feature parameters. and $\mathbf{x}_r$ as some 'robot frame', a translation and rotation. Then

$$\mathbf{x}_o = T(\mathbf{x}_f|\mathbf{x}_r) \tag{4.5}$$

are the feature parameters in a new reference frame and the changes to these coordinates satisfy:

$$\delta\mathbf{x}_o = J_{or}\delta\mathbf{x}_r + J_{of}\delta\mathbf{x}_f. \tag{4.6}$$

The point here is that the Jacobians above depend on the type of coordinate only. So that the calculation of these is the same for any feature.

Now we need to address the issue of symmetries. The invariances in the measurements need some explicit treatment that is in some way generic for all features. The measurements give us some information on some direction in the features parameters space. Small changes to the feature parameters will cause some perturbation in these directions. The perturbations $\delta\mathbf{x}_p$ are projected from changes in the parameter space $\delta\mathbf{x}_f$ by the B matrices.

$$\delta \mathbf{x}_p = B(\mathbf{x}_f)\delta \mathbf{x}_f, \tag{4.7}$$

$$\delta \mathbf{x}_f = \widetilde{B}(\mathbf{x}_f)\delta \mathbf{x}_p, \tag{4.8}$$

$$I_{pp} = B(\mathbf{x}_f)\widetilde{B}(\mathbf{x}_f). \tag{4.9}$$

The B matrices are non-linear functions that define the projection at some point in the state space. These matrices then define the symmetries and constraints of our features. We will show several examples of B matrices later. We see that these matrices will allow us to restrict the SLAM estimation to the subspace that was actually measured.

Our feature representation is inspired by the SP-model [37, 88]. There the representation is the coordinates of transformations projected to a smaller subspace with the B matrices. Our method also has a frame attached to the orientable features. This avoids the problem shown in figure 4.5.

4.3 Feature Measurements

The features are measured by some sensor such as a camera or a laser scanner. These measurements $\mathbf{v}$ then can be combined with the state to form an innovation $\eta(\mathbf{v}, \mathbf{x}_f, \mathbf{x}_r)$ We can require that the expectation value of $\eta = 0$. Furthermore, we know that the sensor is making relative measurements of the feature so that,

$$\eta(\mathbf{v}, \mathbf{x}_f, \mathbf{x}_r) = \eta(\mathbf{v}, \mathbf{x}_o). \tag{4.10}$$

Here $\mathbf{x}_o$ is the feature coordinates in the frame of the robot sensor. These innovation are ideally measurements of the noise in our sensor readings,$\mathbf{v}$, plus the error in our state. They obey some probability distribution $P(\eta|\mathbf{x}_o)$. We will assume this distribution to be Gaussian.

We can then relate the uncertainty in the measurements $\mathbf{v}$ to an uncertainty in the measured innovation η:

$$C_{\eta\eta} = J_{\eta v} C_{vv} J_{\eta v}^T. \tag{4.11}$$

Where $C_{\eta\eta}$ is the covariance matrix of the innovations. Furthermore, we can calculate the linear change in the innovation due to changes to the relative feature positions as:

$$\delta\eta = J_{\eta o}\delta \mathbf{x}_o. \tag{4.12}$$

The advantage of the M-Space representation is that now we can calculate the $\delta \mathbf{x}_o$ using (4.6).

$$\delta\eta = J_{\eta o}\{J_{or}\delta \mathbf{x}_r + J_{of}\delta \mathbf{x}_f\}. \tag{4.13}$$

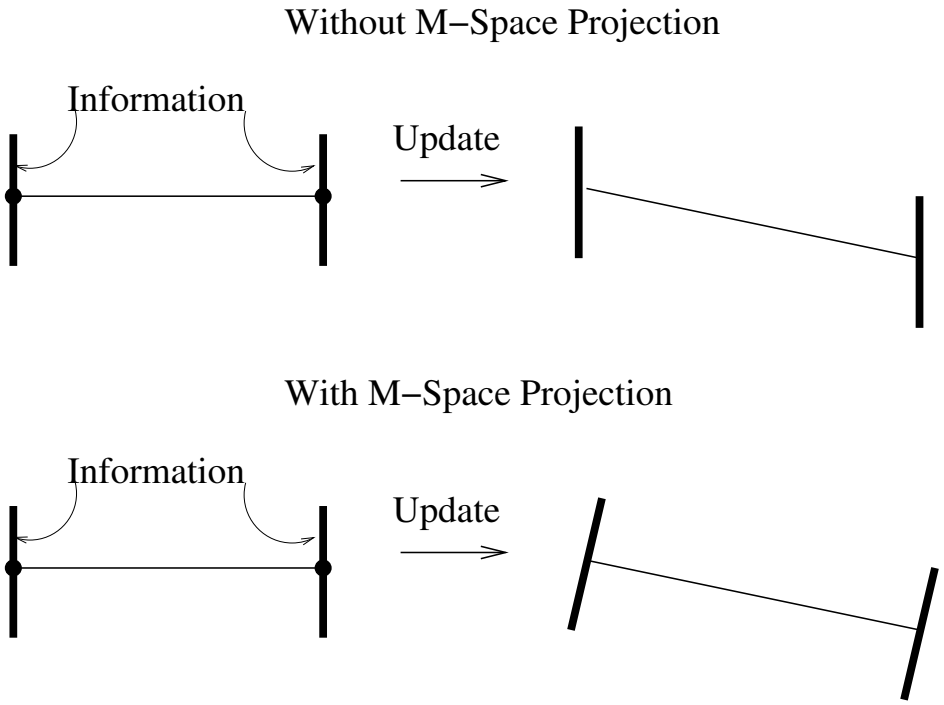

Figure 4.5: Here we see how a line feature parameterization without projecting out the M-Space dimensions can lead to information after the SLAM update that is not consistent with actual measurements. The information before the update was normal to the line but afterwards has some component tangent to the line. Using the M-Space representation the information essentially rotates with the line in this case. This greatly reduces the effect of linearizations on consistency.

Finally the symmetries can also be accounted for with a generic formula by using (4.8).

$$\delta\eta = J_{\eta o}\{J_{or}\delta\mathbf{x}_r + J_{of}\widetilde{B}(\mathbf{x}_f)\delta\mathbf{x}_p\}. \tag{4.14}$$

Equation (4.17) is rather fundamental as it gives the factorization of the measurements into a part that depends only on the invariances and symmetries and a part that is specific to the measurement. We will try to match this factorization later when we find a form for the invariant linearized sums of measurements.

We can even formulate this as a relative derivative of the innovation,

$$D_o\eta = J_{\eta o} = \frac{d\eta}{d\mathbf{x}_o} \tag{4.15}$$

times a relative incremental change in the state vector $\mathbf{x}_s$,

$$D^o \delta \mathbf{x}_s = (J_{or}, J_{of} \widetilde{B}(\mathbf{x}_f)) \begin{pmatrix} \delta \mathbf{x}_r \\ \delta \mathbf{x}_p \end{pmatrix}. \tag{4.16}$$

$$\delta \eta = D_o \eta D^o \delta \mathbf{x}_s \tag{4.17}$$

If one formulates a SLAM algorithm in terms of products like this then one is sure to not violate the invariance of the measurements. The D^o matrix projects arbitrary perturbations of the state to the space of perturbations of the parameters relative to some base frame here taken as the sensor frame of the robot, $\mathbf{x}_r$. Here, the parameters are coordinates of the points in the feature representation $\mathbf{x}_f$, but they could, in general, also include other pose coordinates than the base frame. For the pose coordinates the B matrix is the identity matrix.

Walls

Walls are parameterized by two 2D points, the start and end points. The start point is to the right when facing the wall.

$$\mathbf{x}_f = \begin{pmatrix} x_{start} \\ y_{start} \\ x_{end} \\ y_{end} \end{pmatrix}, \tag{4.18}$$

For the case of two M-Space dimensions, ($p\text{-}dim = 2$), the B matrix looks like:

$$B_{wall} = \begin{pmatrix} \frac{\cos\gamma}{L} & \frac{\sin\gamma}{L} & \frac{-\cos\gamma}{L} & \frac{-\sin\gamma}{L} \\ \cos\gamma & \sin\gamma & \cos\gamma & \sin\gamma \end{pmatrix} / \sqrt{2}, \tag{4.19}$$

Here γ is the angle between the normal of the line and the x-axis.

$$\tan\gamma = \frac{x_{start} - x_{end}}{y_{end} - y_{start}} \tag{4.20}$$

L is the length of the wall.

$$L = \sqrt{(x_{end} - x_{start})^2 + (y_{end} - y_{start})^2} \tag{4.21}$$

This B matrix, and thus the M-Space, corresponds to a rotation of the line in the horizontal plane around its center and a parallel motion of the wall perpendicular to the wall. We also have

$$\widetilde{B}_{wall} = \begin{pmatrix} L\cos\gamma & \cos\gamma \\ L\sin\gamma & \sin\gamma \\ -L\cos\gamma & \cos\gamma \\ -L\sin\gamma & \sin\gamma \end{pmatrix} / \sqrt{2}, \tag{4.22}$$

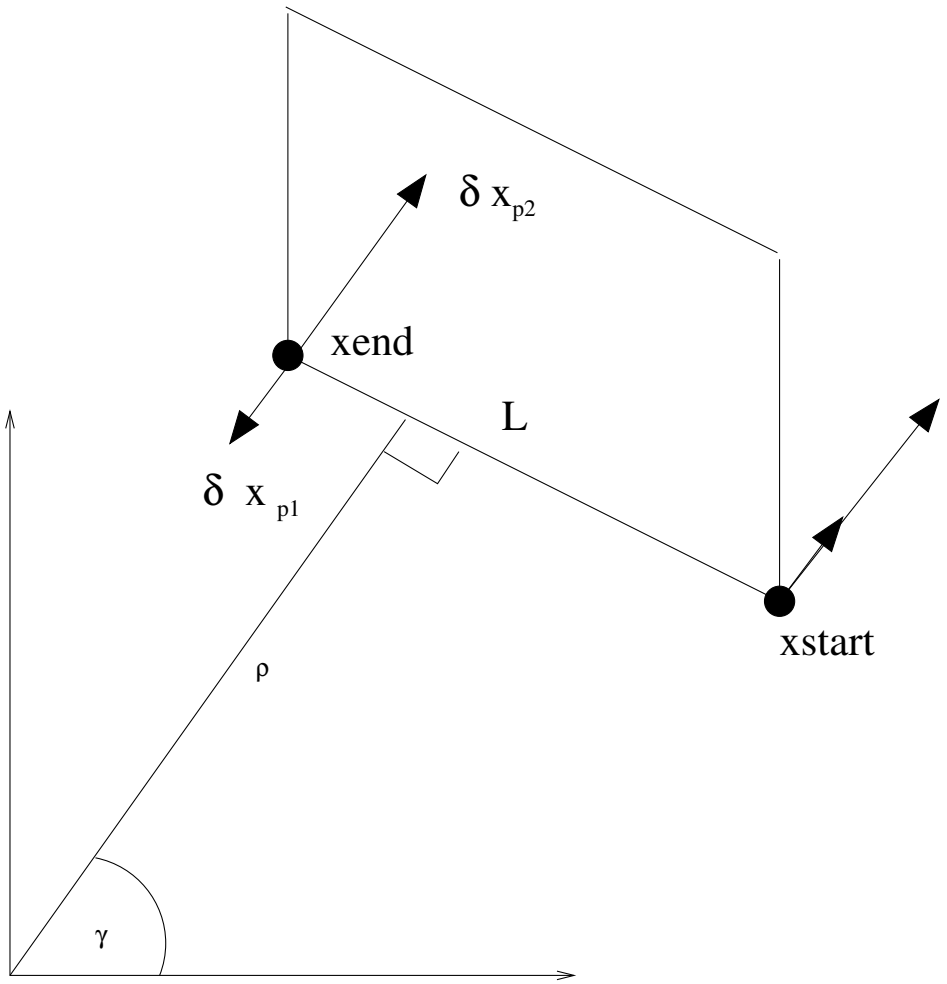

Figure 4.6: Illustration of the M-Space directions for the wall representation. The shorter arrows are the first M-Space direction, rotations. The longer arrows show the position M-Space direction. Typically wall measurements only provide these two directions.

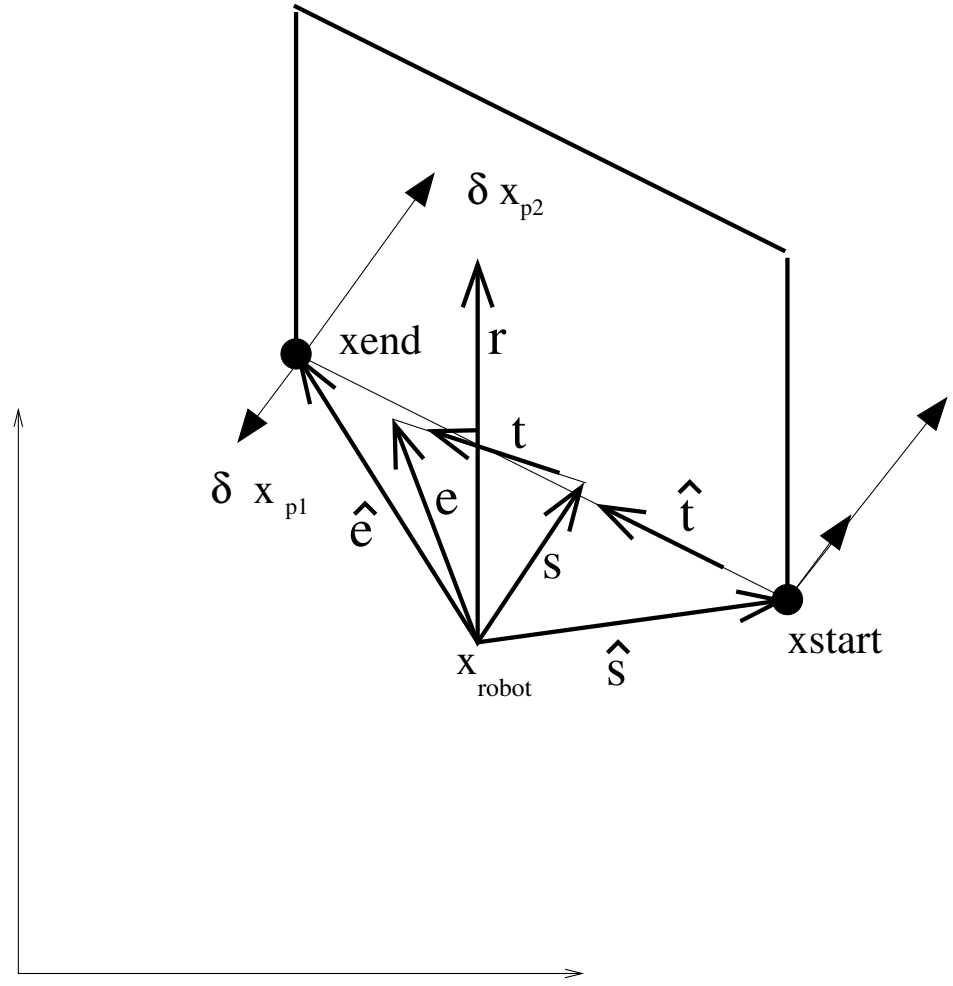

Figure 4.7: Illustration of the vectors involved in defining the innovation for wall measurements. The vectors with 'hats' are the 'predicted' values while the plain letters are the measured vectors.

The wall measurements are 4 dimensional, the two 2-D endpoints. These measurements give two vectors in the laser frame which point to the start and end points of the detected wall. Call these vectors $\mathbf{s}$ and $\mathbf{e}$.

$$\mathbf{s} = (v_0, v_1), \text{ and } \mathbf{e} = (v_2, v_3). \tag{4.23}$$

Where (v_0, v_1) and (v_2, v_3) are the start and end points in the laser frame. We form two more vectors out of these,

$$\mathbf{t} = \frac{\mathbf{e} - \mathbf{s}}{|\mathbf{e} - \mathbf{s}|}, \ \mathbf{r} = \mathbf{e} + \mathbf{s}, \tag{4.24}$$

The innovation function is two dimensional,

$$\eta = \begin{pmatrix} \mathbf{t} \times \hat{\mathbf{t}} \\ (\hat{\mathbf{r}} - \mathbf{r}) \times \hat{\mathbf{t}} \end{pmatrix}. \tag{4.25}$$

Here $\hat{\mathbf{t}}$ denotes the prediction of $\mathbf{t}$ calculated using predictions $\hat{\mathbf{s}}$ of $\mathbf{s}$ and $\hat{\mathbf{e}}$ of $\mathbf{e}$.

$$\hat{\mathbf{s}} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{pmatrix} (\mathbf{x}_o), \tag{4.26}$$

$$\hat{\mathbf{e}} = \begin{pmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} (\mathbf{x}_o). \tag{4.27}$$

We can then calculate $J_{\eta o}(\hat{\mathbf{x}}_o, \mathbf{v})$ and $J_{\eta v}(\hat{\mathbf{x}}_o, \mathbf{v})$ by differentiating this innovation.

$$J_{\eta o}(\hat{\mathbf{x}}_o, \mathbf{v}) = \frac{\partial \eta}{\partial \hat{\mathbf{x}}_o} \tag{4.28}$$

$$J_{\eta v}(\hat{\mathbf{x}}_o, \mathbf{v}) = \frac{\partial \eta}{\partial \mathbf{v}} \tag{4.29}$$

If the *p-dim* has grown to 4 after observation of both endpoints the B matrix is,

$$B_{wall} = \begin{pmatrix} \frac{\cos\gamma}{\sqrt{2}L} & \frac{\sin\gamma}{\sqrt{2}L} & \frac{-\cos\gamma}{\sqrt{2}L} & \frac{-\sin\gamma}{\sqrt{2}L} \\ \frac{\cos\gamma}{\sqrt{2}} & \frac{\sin\gamma}{\sqrt{2}} & \frac{\cos\gamma}{\sqrt{2}} & \frac{\sin\gamma}{\sqrt{2}} \\ -\sin\gamma & \cos\gamma & 0 & 0 \\ 0 & 0 & -\sin\gamma & \cos\gamma \end{pmatrix} \tag{4.30}$$

Here the endpoints motion tangent to the line has been added. Now measurements can be as long as 4 dimensional:

$$\eta = \begin{pmatrix} \mathbf{t} \times \hat{\mathbf{t}} \\ (\hat{\mathbf{r}} - \mathbf{r}) \times \hat{\mathbf{t}} \\ (\hat{\mathbf{s}} - \mathbf{s}) \cdot \hat{\mathbf{t}} \\ (\hat{\mathbf{e}} - \mathbf{e}) \cdot \hat{\mathbf{t}} \end{pmatrix}. \tag{4.31}$$

Point Features

Point features are parameterized by one 3-D point,

$$\mathbf{x}_f = \begin{pmatrix} x \\ y \\ z \end{pmatrix}, \tag{4.32}$$

These features have a B Matrix that is simply the identity matrix.

$$B = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}. \tag{4.33}$$

The measurement vector is the point in the image plane and the focal length of the camera. (Our image plane has the y axis coming out the center of the image).

$$\mathbf{v} = \begin{pmatrix} x_{image} \\ z_{image} \\ focal \end{pmatrix}. \tag{4.34}$$

The innovation function for these is simply the vector in the image plane from the predicted image point to the measured image point.

$$\eta = \begin{pmatrix} \frac{-v_2 x_o}{y_o - v_2} - v_0 \\ \frac{-v_2 z_o}{y_o - v_2} - v_1 \end{pmatrix}. \tag{4.35}$$

Horizontal Line Features

The horizontal line features are parameterized by two 3-D points.

$$\mathbf{x}_f = \begin{pmatrix} x_{start} \\ y_{start} \\ z_{start} \\ x_{end} \\ y_{end} \\ z_{end} \end{pmatrix}, \tag{4.36}$$

For the case of $p\text{-}dim = 1$, B matrix looks like:

$$B = \frac{\begin{pmatrix} \cos\gamma & \sin\gamma & 0 & -\cos\gamma & -\sin\gamma & 0 \end{pmatrix}}{L\sqrt{2}}. \tag{4.37}$$

Here γ is the angle between the projection of the normal to line into the horizontal plane and the x-axis.

$$\tan\gamma = \frac{x_{start} - x_{end}}{y_{end} - y_{start}} \tag{4.38}$$

This B matrix corresponds to a rotation of the line in the horizontal plane around its center.

The horizontal line measurements are 5 dimensional, two pixels in the image corresponding to the line end points and the focal length of the camera.

$$\mathbf{v} = \begin{pmatrix} x_{image-start} \\ z_{image-start} \\ x_{image-end} \\ z_{image-end} \\ focal \end{pmatrix}. \tag{4.39}$$

These measurements give two vectors in the camera frame which point to the end points of the detected line segments. Call these vectors $\mathbf{s}$ and $\mathbf{e}$ (see figure 4.8).

$$\mathbf{s} = (-v_0, v_4, -v_1), \text{ and } \mathbf{e} = (-v_2, v_4, -v_3). \tag{4.40}$$

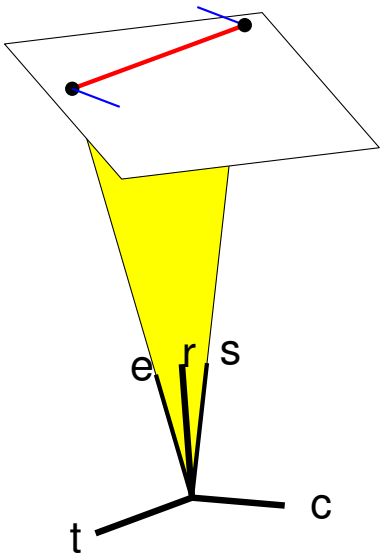

Figure 4.8: A horizontal line feature showing the 1 dimensional M-Space.

As we have said the camera axis is along the y direction and (v_0, v_1) and (v_2, v_3) are the start and end points in image plane and v_4 is the focal length. We form some unit vectors out of these, (see figure 4.8),

$$\mathbf{c} = \frac{\mathbf{s} \times \mathbf{e}}{|\mathbf{s} \times \mathbf{e}|}, \ \mathbf{r} = \frac{\mathbf{e} + \mathbf{s}}{|\mathbf{e} + \mathbf{s}|}, \text{ and } \mathbf{t} = \frac{\mathbf{e} - \mathbf{s}}{|\mathbf{e} - \mathbf{s}|}, \tag{4.41}$$

Now we can define an innovation as,

$$\eta = \mathbf{c} \cdot \hat{\mathbf{t}}. \tag{4.42}$$

Here $\hat{\mathbf{t}}$ denotes the prediction of $\mathbf{t}$ calculated using predictions $\hat{\mathbf{s}}$ and $\hat{\mathbf{e}}$ of $\mathbf{s}$ and $\mathbf{e}$.

We can then calculate $J_{\eta o}(\hat{\mathbf{x}}_o, \mathbf{v})$ and $J_{\eta v}(\hat{\mathbf{x}}_o, \mathbf{v})$ by differentiating this innovation.

For the three dimensional M-Space the B matrix looks like,

$$B = \begin{pmatrix} \frac{\cos\gamma}{L} & \frac{\sin\gamma}{L} & 0 & \frac{-\cos\gamma}{L} & \frac{-\sin\gamma}{L} & 0 \\ \cos\gamma & \sin\gamma & 0 & \cos\gamma & \sin\gamma & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 \end{pmatrix} / \sqrt{2}. \tag{4.43}$$

where the second row corresponds to motion in the horizontal plane orthogonal to the line and the third row corresponds to changing the height of the line. Now the innovation becomes two dimensional and for the second component we take:

$$\eta_2 = \mathbf{c} \cdot \hat{\mathbf{r}}, \tag{4.44}$$

4.4 Feature Matching

The matching of a feature measurement with the map feature that gave rise to it is a critical part of doing SLAM. The matching is done in two phases first we form a list of candidate matches in a generic way. Then we test the list using some specific test for the SLAM algorithm being used. The list is meant to be rather loosely matched. Its purpose is to reduce the number of checks needed by the more intensive test. To form the list we use a series of lists where each list is used as an input to some test that then reduces the length of the list.

1. Form near list
2. Transform all near list features to Sensor-Frame
3. Form visible list
4. For each feature measurement

   4.1 Form list of rough matches

   4.2 For each rough match

      4.2.1 Calculate the fine match

      4.2.2 Order the list by fine match

   4.3 If the best fine match is within a threshold

      accept the preliminary match list.

      Else create a new feature for the measurement.

The first list is the 'near list'. Each feature has a method that will return true if the feature is inside some rectangle. For line features or other extended features inside means if any part of the feature is inside the rectangle. This list can be generated every few iteration with a rectangle centered on the robot and large enough to ensure any feature within range of the sensor is included.

The near features are all transformed to the sensor frame for further testing. For vision features that have not been fully initialized we now check if the bearing to the feature is very different than it was the last time we saw it. If so we cannot reliably match to it using geometric information alone so we remove this feature from the list.

The features on the near list are then tested with the current sensor pose to see if they are visible. Visible features are ones in front of the sensor and facing the sensor, (walls are considered invisible when viewed from behind). The possible occlusions are also taken into account when forming the 'visible list'.

It is then the visible list that is matched against the measurements. A rough match test is done for each measurement to the visible features. This consists of first putting a smaller rectangle around each measurement and then testing if the feature is in this rectangle.

Then a fine test is applied. This fine test is a threshold applied to the metric distance between the measurement and the feature. For the example of walls, we use a threshold on the difference in angles squared plus the difference in perpendicular distance squared, with an appropriate scaling of each term.

$$finetest = (\mathbf{z} - \hat{\mathbf{z}})^T M (\mathbf{z} - \hat{\mathbf{z}}), \tag{4.45}$$

Where $\mathbf{z}$ is a column vector such as for example, the angle and perpendicular distance to the measured line in the sensor frame. $\hat{\mathbf{z}}$ is the prediction of these quantities given the feature match. M is a metric chosen for this type of measurement.

After passing all these 'generic tests' the SLAM algorithm will generally apply its own final test to the match such as mahanolobis distance or the energy criteria to be introduce later.

4.5 Feature Initialization

Feature initialization can be somewhat tricky. For EKF SLAM one would like to initialize the features quickly so that they can be used by the SLAM algorithm before they disappear from view. This rush to initialize must be balanced against the need to be sure that the feature is a real and static feature and that the m-space direction is known well enough to ensure valid linearization.

The feature M-Space dimensions are initialized and the soft dimensions [6] estimated by using dense information. This information is accumulated on the feature and evaluated to set the soft dimensions and eventually extend the P-dimensions.

[6] The soft dimensions are the complement of the M-Space.

The accumulated information is tagged with an age. The age grows as the distance traveled by the robot between calls to add information. This distance includes rotations with rotation by 1 radian weighted to equal travel by 5 meters. When the dense information gets too old it is discarded and will not be used for initialization.

The dense information is collected in the dead-reckoning frame which is continuous. This allows us to avoid the problem of sudden adjustments of the pose by the SLAM algorithm causing the new dense information to be not comparable to the old. Over time the accumulated errors in the dead reckoning are what motivate the throwing away of too old dense information. The transformation from the dead reckoning to the map frame is used to adjust the results.

We will see that when using the graphical method we can repeatedly add the dense information and thus use the map frame instead of the dead reckoning. This is because in the graphical method the recent robot path is updated each iteration and thus a continuous path is available for the dense information. Each time we check to see if we can extend the *p-dim* of the feature we must add the information from all the measurements using the corrected poses along the path.

Walls

For walls the dense information added to the feature consists of clouds of points from a laser scan. This cloud is added to the accumulated cloud and tested. First the list of points is ordered by the distance along the line. Then sections of the cloud that have distances between adjacent points less than some 'tightness' threshold are formed. The the section must have enough points, enough length and small enough variance off the line. After passing all these tests the wall is initialize to $p\text{-}dim = 2$.

After this the endpoints will be extended along the wall as more scan points are added to the cloud within the tightness threshold distance from the current end.

When the measurements include the detection of the endpoint of the wall, the feature will initialize the corresponding M-Space dimension as long as the measured endpoint is in good agreement with the current best guess of the endpoint.

Point Features

For point features measured with a camera the dense information consists of the bearing vector to the feature and the location of the camera. For initialization, these bearing vectors are examined pairwise. If the distance traveled between the two bearings was sufficient for meaningful triangulation the pair will contribute a weighted estimate to the older bearing's estimate of the range to the point. The weight is $sin^2(\theta_{ij})$, where θ_{ij} is the angle between the two bearing vectors figure 4.9.

$$r_{ij} = \frac{\sin \phi_{ij} d_{ij}}{\sin \theta_{ij}} \tag{4.46}$$

$$w_j = \sum_{i>j} \sin^2 \theta_{ij} \tag{4.47}$$

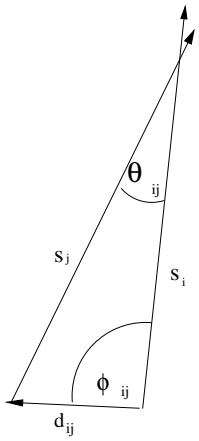

Figure 4.9: A point feature showing the triangulation between two observations.

$$r_j = \frac{\sum_{i>j} r_{ij} \sin^2 \theta_{ij}}{w_j} \tag{4.48}$$

If the total weight of all the vectors is larger than a threshold the point is initialized.

$$w_{total} = \sum_i w_j \tag{4.49}$$

A cloud of points is formed from all the individual bearing vectors and their weighted average range values. The weighted center of this cloud is the estimate of the point feature location and the weighted variance of the cloud becomes an estimate of the initial uncertainty relative to the current pose.

$$\mathbf{x}_j = r_j \frac{\mathbf{s}_j}{|\mathbf{s}_j|} \tag{4.50}$$

$$\mathbf{x} = \frac{1}{w_{total}} \sum_i w_j \mathbf{x}_j \tag{4.51}$$

Horizontal Line Features

For lines measured by the camera and known to be horizontal we can initialize the tangent vector direction of the line before we can initialize the position of the line. The dense information here is the bearings of the start and end points and the position of the camera. These give an estimate of the tangent direction which we weight and add to the accumulated weighted tangents.

We take the weight of each measurement as $sin(\theta_{se})$, where θ_{se} is the angle between the two bearing vectors of the start and end points of the line as measured at the camera figure 4.10.

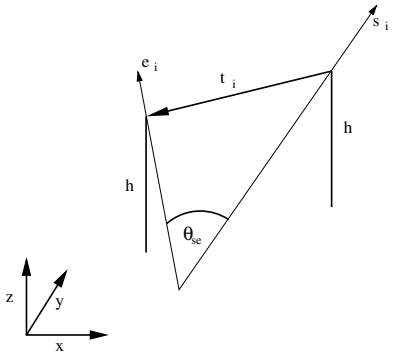

Figure 4.10: A horizontal line feature showing the calculation of the tangent from one measurement. $|\mathbf{t}_i| = 1$ and the height of the tangent vector, h, is the same at both ends.

$$w_i = \sin\theta_{se} \tag{4.52}$$

$$w_{total} = \sum_i w_i \tag{4.53}$$

$$\mathbf{t} = \frac{\sum_i w_i \mathbf{t}_i}{w_{total}} \tag{4.54}$$

When the total of the weights, w_{total}, exceeds a threshold the tangent direction is initialized.

For the position information we use a weight for each pair of measurements equal to $sin^2(\theta_{ij})$, where θ_{ij} is the angle between the two planes defined by the line measurements. Each such pair of measurements define the distance of the line from the origin. When the total weight is above a threshold we initialize the position with the weighted average of the positions defined by each measurement. This is totally analogous to the point feature triangulation except that everything is projected to the plane perpendicular to the line tangent vector.

These vision initialization schemes while crude were sufficient for the ceiling features as we had a variation in height above the camera of between 1.5 and 2.5 meters. This relatively small variation in depth made initialization much easier. For a more general case more sophisticated methods might be needed.

4.6 Summary

The Features to be used in the following are wall observed with a SICK laser, ceiling lines and lamps observed with a camera. The walls are extracted from the laser scans by means of a Range Weighted Hough Transform. This algorithm has been optimized by using a two stage accumulator.

The camera images are processed using standard software to extract lines and lamps on the ceiling.

The features are represented in a general framework called the M-Space feature representation. The M-Space allows one to generically transform the features. It also describes the constraints and symmetries of the features by means of the b matrices. These then help to formulate SLAM algorithms that respect these symmetries and constraints.

Part II

Estimation

Chapter 5

Extended Kalman Filter Based Methods for Localization and SLAM

Localization and mapping can be phrased as a standard least square estimator. When applied recursively the typical formulation, for feature systems, is the Kalman filter. As the system is non-linear the linearized version, Extended Kalman filter, is used.

The Extended Kalman Filter, EKF, is a very good method for both localization and full SLAM. The idea is to linearize the current measurements around the current best state estimate and then feed the measurements into a standard linear Kalman filter. For this to work the the linearized measurements must be Gaussian.

For a linear Gaussian system, the Kalman Filter will solve (2.7) exactly using an iterative formula in i. For the SLAM problem the EKF is only an approximation to the solution [36, 37, 38, 16].

5.1 EKF Localization

When one has an a priori map of the environment one can use it to localize the robot. One way to achieve this is with an EKF. The main requirement is that the robot must remain localized in the map at all times. If the robot loses track of its pose it will be impossible to regain it using the EKF alone. By lose track we mean that the error in the robot pose is so large that it can no longer distinguish between map features based on the location alone.

The idea of the EKF is to represent the state as the pose of the robot at time t, $\mathbf{x}_r(t)$, and the estimated covariance of that pose, $C_r(t)$. The time variable here is discrete. As the robot moves the dead reckoning will allow us to predict the new state by.

$$\mathbf{x}_r(t) = f(\mathbf{x}_r(t-1), \mathbf{x}_d(t, t-1)) \tag{5.1}$$

where $\mathbf{x}_d(t, t-1))$ is the dead reckoning incremental coordinates and f is a non-linear vector valued function describing the motion. The covariance will also change,

$$C_r(t) = J_{rr} C_r(t-1) J_{rr}^T + J_{rd} C_d(t, t-1) J_{rd}^T \tag{5.2}$$

Here we assume the dead-reckoning estimate is statistically independent of the state at t-1. The J's here are the Jacobians of the function f.

When features on the map are observed the update step can be invoked using the matrix K,

$$K = C_r(t) J_{\eta r}^T S^{-1} \tag{5.3}$$

where S is,

$$S = (J_{\eta r} C_r J_{\eta r}^T) + C_\eta(t) \tag{5.4}$$

The updated robot pose state is then $\mathbf{x}_r(t) + \delta\mathbf{x}_r(t)$,

$$\delta\mathbf{x}_r(t) = -K\eta(\mathbf{x}_r(t)). \tag{5.5}$$

The updated covariance of the robot $C_r(t) + \delta C_r(t)$

$$\delta C_r(t)' = -K J_{\eta r} C_r(t) \tag{5.6}$$

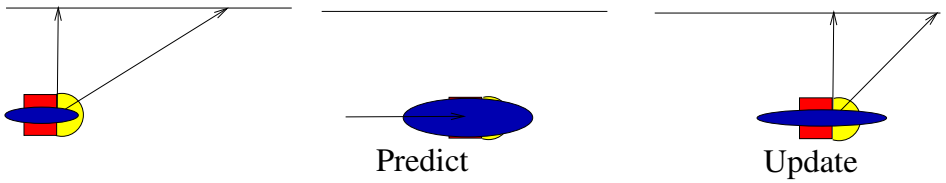

Figure 5.1: This shows the effect of dead-reckoning prediction step followed by a feature measurement update step on the covariance of the robot position estimate. The error grows during dead-reckoning but shrinks after using the wall measurement. The error along the wall does not decrease.

We notice that the predict step always increases the covariance and the update step always decreases it, as illustrated in figure 5.1. The final matching test is usually taken as test of the mahanolobis distance,

$$\delta_M = \eta(\mathbf{x}_r(t))^T S^{-1} \eta(\mathbf{x}_r(t)). \tag{5.7}$$

If this δ_M is below some threshold the match is considered sufficiently likely to be used. If several features match the same measurement the lowest δ_M is used but this is a sign that the robot is not sufficiently well localized to resolve these features

unambiguously. Problems might then occur. Figure 3.6 shows an example of EKF localization for a data set that we will use in later experiments. Localization given a map is a solved problem using the EKF as long as the robot is localized from the start and does not lose sight of the map landmarks.

5.2 EKF SLAM

The SLAM solution works like the localization EKF except that now the state perturbations include the feature M-Space perturbations,

$$\delta x_s = \begin{pmatrix} \delta \mathbf{x}_r \\ \delta \mathbf{x}_p \end{pmatrix}. \tag{5.8}$$

and the $J_{\eta r}$ from the localization case (5.4) gets replaced by $J_{\eta s}$,

$$J_{\eta s} = J_{\eta o} \begin{pmatrix} J_{or} & J_{of}\widetilde{B} \end{pmatrix}. \tag{5.9}$$

Here $\widetilde{B}(\mathbf{x}_f)$ is calculated every time that $\mathbf{x}_f$ changes. The variation direction in the global frame that the covariance matrix describes is changing after each update. We can project the change in the M-Space to changes to the feature parameters by

$$\delta x_f = \widetilde{B}\delta \mathbf{x}_p. \tag{5.10}$$

We notice that the filter must keep track of the current robot poses $\mathbf{x}_r$ and the uncertainty in that pose. It also tracks the covariance estimate of the errors in the pose and the M-Space perturbations. The state of the features is not part of the filter. [1] The features separately track the $\mathbf{x}_f$ and the Kalman filter will provide adjustments to those coordinates as shown above.

We can also include any parameters or the robot to sensor pose in $\mathbf{x}_r$. We just need to calculate their Jacobians each time.

Correspondingly the update step using our relative form is:

$$K^T = S^{-1} D_o \eta D^o C_s(t) \tag{5.11}$$

where S is,

$$S = (D_o \eta D^o C_s (D_o \eta D^o)^T + C_\eta(t) \tag{5.12}$$

The updated state is then $\mathbf{x}_s(t) + \delta \mathbf{x}_s(t)$,

$$\delta \mathbf{x}_s(t) = -K\eta(\mathbf{x}_s(t)). \tag{5.13}$$

The updated covariance of the state,

$$C_s(t)' = C_s(t) + \delta C_s(t) = (I - K(D_o \eta D^o))C_s(t) \tag{5.14}$$

[1] $\mathbf{x}_p$ is not a meaningful expression in our theory.

Experiments

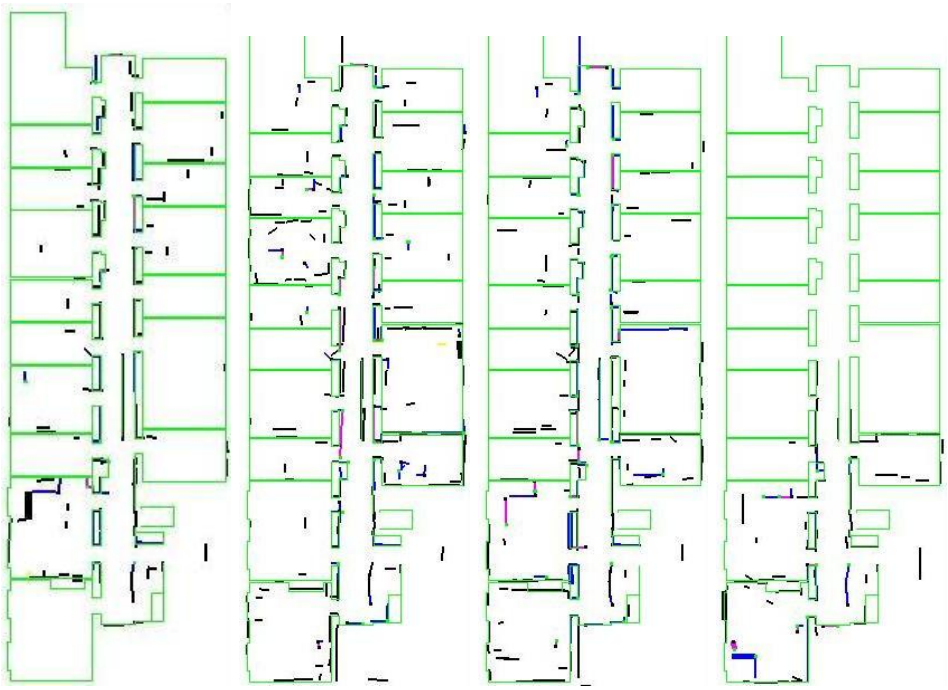

Figure 5.2: Four EKF maps made by from left to right a PeopleBot, Pioneer, PowerBot and a custom built robot. The number of walls in the maps were 120, 232, 218 and 86 respectively. The lighter (green) lines represent the hand made map while the darker (black) lines are the SLAM map. The size of this building is 13 by 39 meters.

The EKF SLAM implementation shown here uses the M-Space feature representation. Here we show some of the extensive test results for this SLAM implementation which can handle vision and laser data, separately or in combination.

We tested our EKF SLAM on four different indoor robots in the lab. Each of these robots was equipped with a SICK laser scanner LMS-200. The results are shown in figure 5.2. The four robots used the same configuration parameters for the EKF SLAM program. Three robots were from ActivMedia, a Peoplebot, a Pioneer and a PowerBot. The fourth was a custom made robot.

In figure 5.3 we show a comparison between the maps made with the different robots. We matched the maps pairwise by matching walls that were within 50 cm and 6 degrees of each other. The best match was taken if more than one was found. As the experiments were performed on different days and the paths only partially

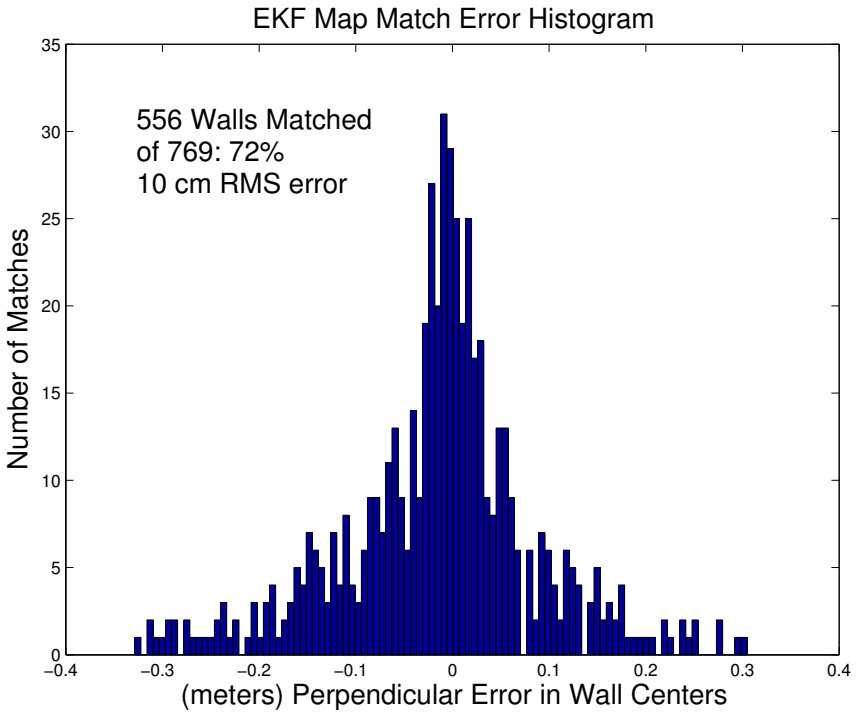

Figure 5.3: Here we see the distribution of the differences between the location of the wall centers after matching the maps pairwise. The walls were matched using a simple matching of all walls within 50 cm and 6 degrees. We matched all 6 pairs of maps and cumulated the results in this histogram. 72% of the walls matched. The RMS Error in the perpendicular distance of the wall centers was 10 cm.

overlapped there were some non-matched walls. Figure 5.4 show the extent of the matching for the two largest maps, made by the Pioneer and the PowerBot robots. Overall we found 556 matches out of a total of 769 walls. We tried to match each wall from the smaller map to the walls of the larger. The RMS difference in the center locations was 10.1 cm, measured normal to the wall.

One can see that the EKF can make accurate maps consistently if enough good features are always in view. And the data association is unambiguous. For these maps the calculation time was not an issue. Notice that some walls shared endpoints with adjacent walls forming a corner. These corners had one 2D point that, when updated, changed both walls. In other word the corner constraints were explicit in the representation.

We also mounted a web camera pointing straight up at the ceiling. We then

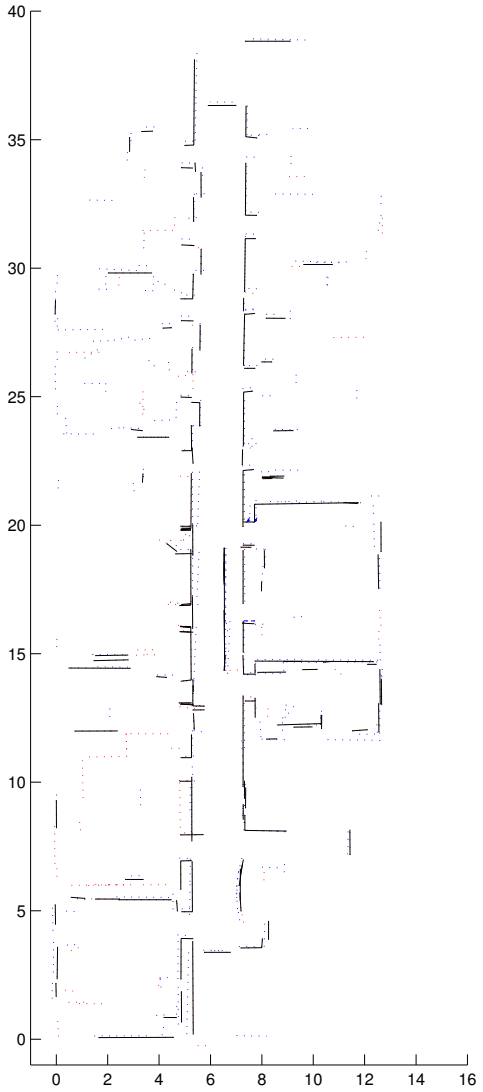

Figure 5.4: This illustrates the extent of the matched sections between the two largest maps. The walls of the original two maps are shown dotted while the matched walls are in solid black. Distances are shown in meters.

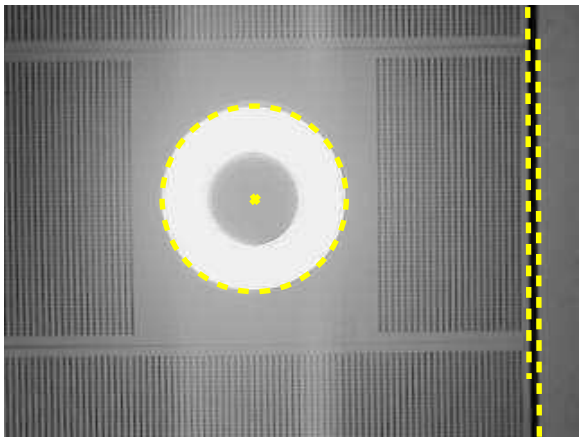

Figure 5.5: Example image with two line features and a lamp feature detected.

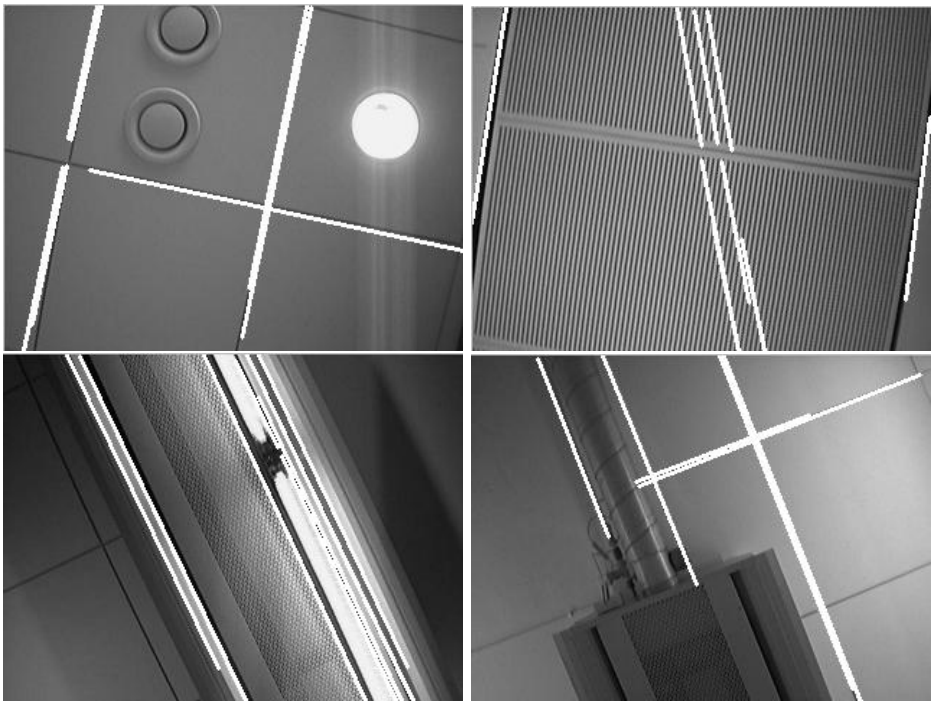

Figure 5.6: Snapshots of the ceiling along the path showing the line detection output.

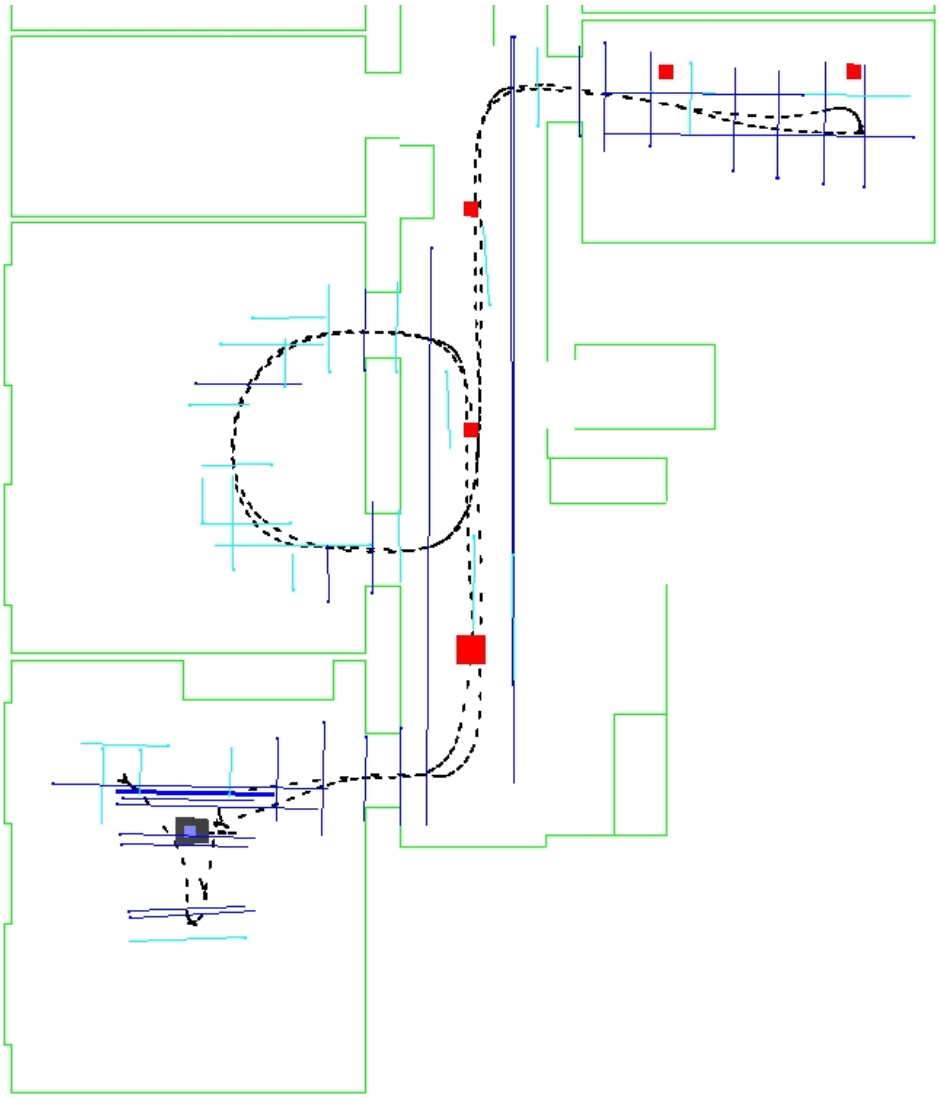

Figure 5.7: The map obtained using only the camera images consists of lines and lamps on the ceiling. The true map of the walls in the lab is also shown for reference. One can see that the robot passes through the center of the doorways and that the rectangular nature of the ceiling lines is maintained indicating good accuracy. The lighter lines have M-Space dimension of 1 (direction only) while the darker lines have 3. The squares are the ceiling lamps which were also used for doing SLAM. The size of the mapped area is 13 by 15 meters

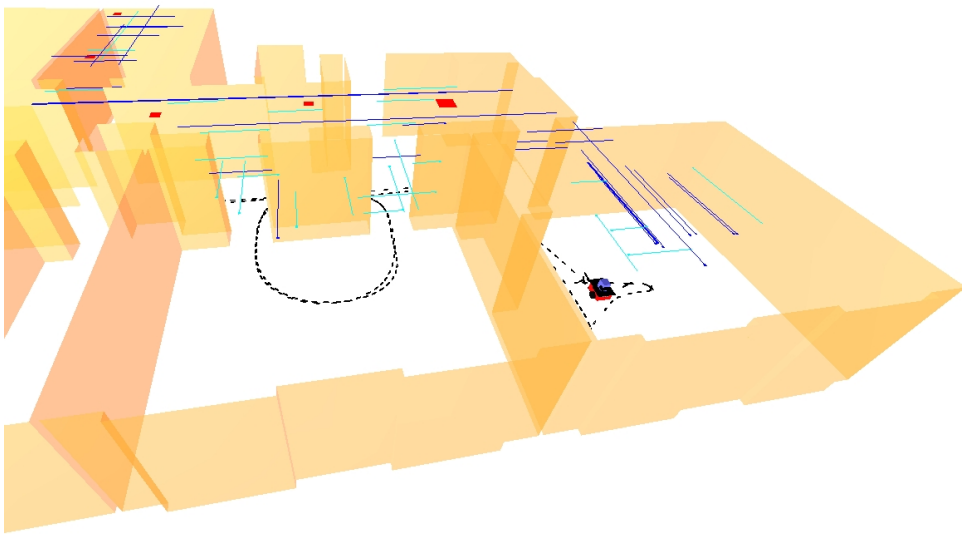

Figure 5.8: The same map from another viewing angle. One can see that the features are indeed 3D. The ceiling height is higher in the rooms than in the hallway.

Figure 5.9: This shows a larger, (13 m x 39 m), EKF map made using vision features on the ceiling with the pioneer robot. The building walls are shown for reference only and were not part of the SLAM estimate. The darker lines are the full 3D horizontal ceiling lines while the lighter (blue) ceiling lines are one dimensional (orientation only). The squares show the locations of ceiling lamps.

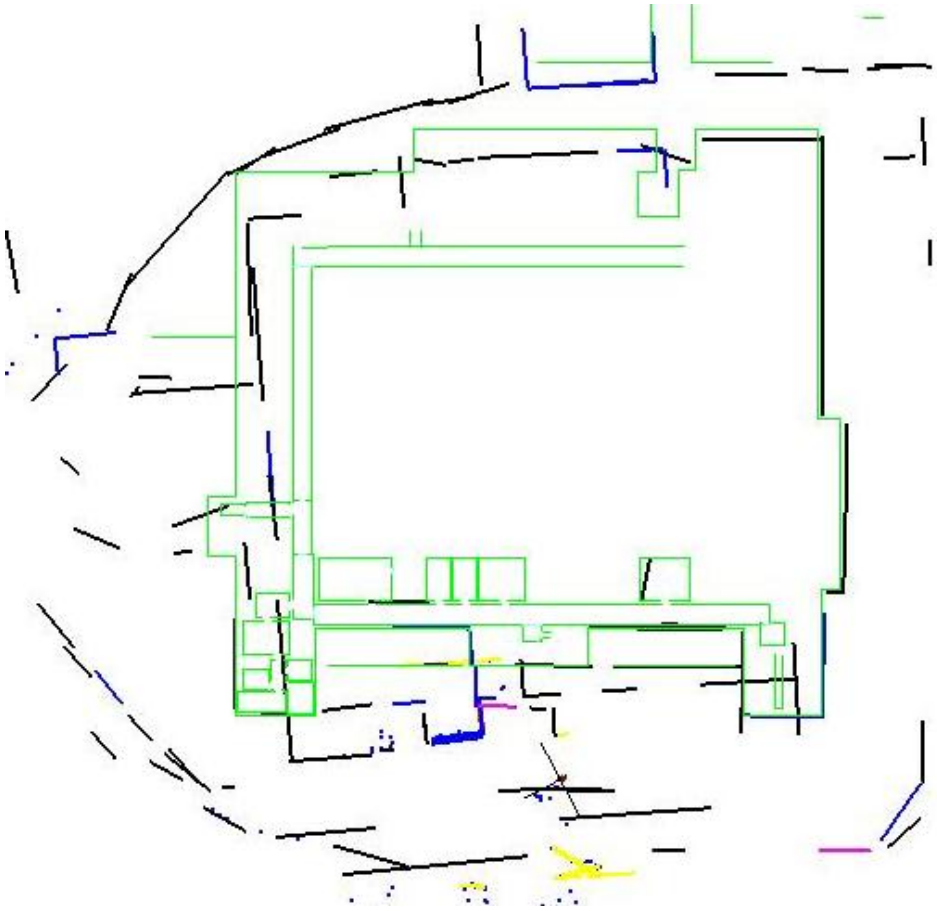

Figure 5.10: Here we show the EKF solution to a data set made with the ATRV robot exploring autonomous. The EKF algorithm cannot consistently be updated with the large error on closing the loop here. The size of this area is about 100 x 100 m.

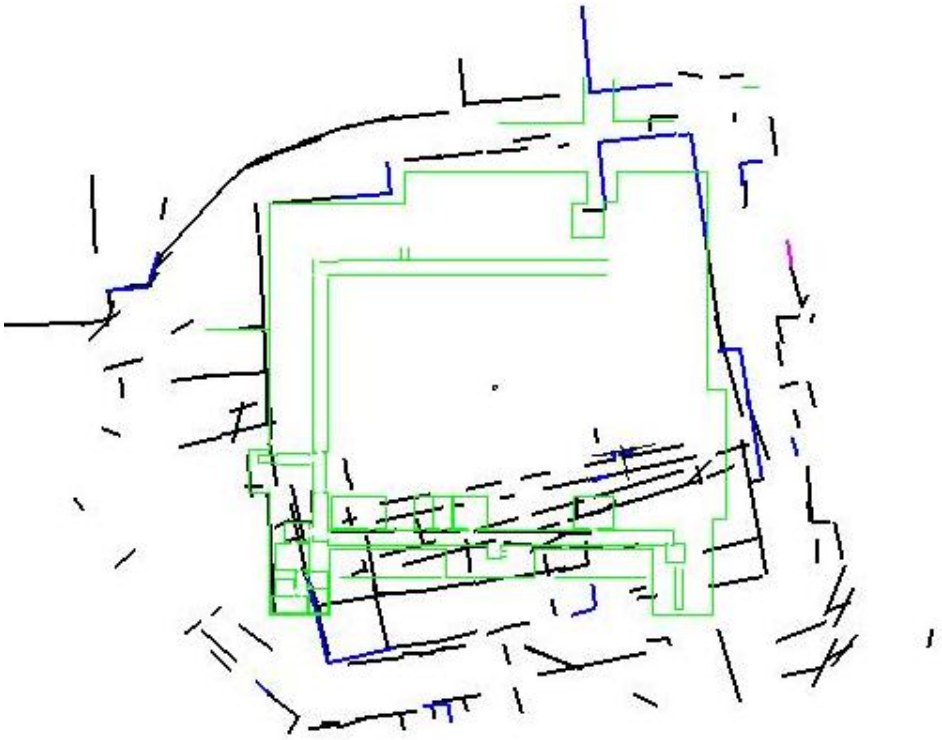

Figure 5.11: Here we show the EKF solution to a data set with 3 closed loops. The algorithm again has no way to deal with the large errors that have accumulated around the loops. The size of this area is about 100 x 100 m.

used the camera and odometry for doing SLAM while using the SICK to check our accuracy.

The images were 320x240 acquired at 10Hz. The features were found by using the OpenCV [2] library. Lines and points were extracted. Lines were extracted using the Hough Transform. For point features we used the center of lamps of circular shape that were detected based on their intensity. Figure 5.5 shows an example image. The height of the linear structures above the camera varied between 1.5 and 2.5 meters. Figure 5.6 shows image snapshots from the environment.

The ability to use the lines for orientation almost immediately was of significant benefit in the corridor where the robot moved parallel to the line for a long distance

[2]OpenCV means Intel: Open Source Computer Vision Library

before turning into a room and finally being able to triangulate the line's position. This helped the robot to stay localized.

Figures 5.7 and 5.8 show the results. Here the walls of a hand made map, (not used by the SLAM program), are shown to help in understanding the path the robot followed. The robot starts and ends the experiment in the room to the lower left in the figure. the dashed curve is the estimated path calculated by the EKF program. The dark lines represent the ceiling lines put into the map. These are the lines that were fully initialized. That is to say 3 M-Space dimensions. The lighter lines are the partially initialized lines. These have 1 M-Space dimension orientation only. The square are the locations of ceiling lamps put into the map. One can see that the return path passes over nearly the same points as the outward path and that it passes through the centers of the doorways. Also the lines are oriented to the building walls parallel. All this indicates good accuracy. The final position was correct to a less than few 5 cm.

We also tried the variation of a Quick Cam camera with wide-angle lens and the pioneer robot. This camera had a focal length of 283 pixels as compared to 503 for the Phillips camera used on the first data set. The wider field of view helped by maintaining the features in view for a longer period of time. Figure 5.9 shows the results for a larger map. For this map we allowed the fixed rotation of the camera relative to the robot to be a variable parameter estimated by the EKF. This helped to model the vibrations of the camera in the robot frame. Without these additional parameters in the EKF we found it impossible to get good results for the camera with this robot. By comparing the laser scan data, (taken at the same time but not used by SLAM here), to the walls of the hand made map we can assert that the robot remained well localized throughout this experiment.

Figure 5.10 shows a EKF SICK laser map made on a large data set outside a building using the ATRV robot equipped with the Crossbow inertial sensor. This data set cannot be solved using the EKF as we cannot impose the loop closing constraint consistently. To do so would violate the linearizations. A solution to this problem will be presented later.

Finally, figure 5.11 shows another EKF SICK laser map made on a very large data set outside the same building using the ATRV robot. The EKF algorithm is unable to impose the global constraint.

5.3 CEKF SLAM

The main idea of the compressed EKF, CEKF, is that the map can be separated into two parts, the part being observed and the unobserved part. We will refer to the part being observed as the visible map rather than local map to emphasis that it only contains recently seen features. The key insight is that while one makes updates to the visible map using the observations, one can accumulate the effect of these updates on the rest of the map in a set of matrices of order the size of the

visible map. Thus, the calculation burden of each iteration become nearly constant, until one needs to update the whole map.

The equations needed to do this were first presented in [19]. We let a signify the visible map and b the rest of the state vector. We can use matrices ψ and ϕ to do a global update of the covariance,

$$C_{ab}(t_3) = \phi(t_3 - 1)C_{ab}(t_2) \tag{5.15}$$

$$C_{bb}(t_3) = C_{bb}(t_2) - C_{ba}(t_2)\psi(t_3)C_{ab}(t_2). \tag{5.16}$$

Then use θ to update the state.

$$\mathbf{x}_b(t_3) = \mathbf{x}_b - C_{ba}(t_2)\theta(t_3). \tag{5.17}$$

These are the global updates done at a low frequency. The matrices ψ and ϕ are calculated iteratively. For the predict step we have,

$$\phi(t) = J_{aa}(t, t-1)\phi(t-1) \tag{5.18}$$

$$\psi(t) = \psi(t-1) \tag{5.19}$$

$$\theta(t) = \theta(t-1) \tag{5.20}$$

The update step is,

$$\phi(t) = (I - C_{aa}(t)\beta(t)\phi(t-1) \tag{5.21}$$

$$\psi(t) = \psi(t-1) + \phi^T(t-1)\beta(t)\phi(t-1) \tag{5.22}$$

$$\theta(t) = \theta(t-1) - \phi^T(t-2)J_{\eta a}^T(t-1)S^{-1}(t-1)\eta(t-1) \tag{5.23}$$

$$\beta(t) = J_{\eta a}^T(t)S^{-1}(t)J_{\eta a} \tag{5.24}$$

In [18] it is pointed out that as one observes a feature from the part of the map not being updated one can selectively update only that feature. In other words one can move a single feature up to the visible map without the need for a full global update. This is important since a full update has a much higher complexity, which in many cases challenges real-time operation. This would force the robot to stop and compute often. By only updating the features as needed the robot can operate non-stop.

The update matrices can be extended by adding row i=a+1 thus:

$$\phi_{a+1,a+1}(t) = \left\{ \begin{array}{cc} \phi_{aa}(t) & 0 \\ -C_{ia}(t_2) \cdot \psi_{aa}(t) & 1 \end{array} \right\}. \tag{5.25}$$

.

$$\psi_{a+1,a+1}(t) = \left\{ \begin{array}{cc} \psi_{aa}(t) & 0 \\ 0 & 0 \end{array} \right\}. \tag{5.26}$$

.

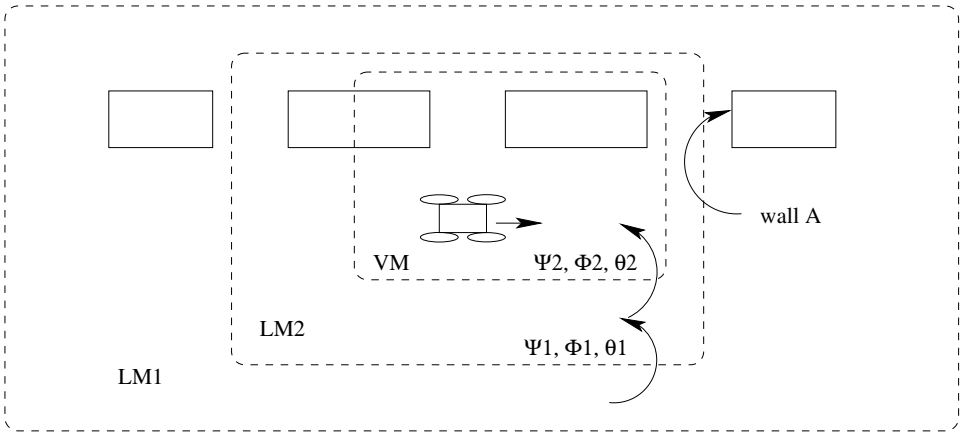

Figure 5.12: Here we illustrate the operation of the compressed filter. The dotted lines enclose each of three maps, LM1, LM2 and VM. VM is the visible map at the current time. LM1 ans LM2 are local maps at earlier times. The robot moves in the direction shown and wall A is observed. Since it is on LM1 it will need to be first added to LM2 and then to VM before it can be updated with the observation.

$$\theta_{a+1}(t) = \left\{ \begin{array}{c} \theta_a(t) \\ 0 \end{array} \right\}. \tag{5.27}$$

.

Here by $C_{i,a}(t_2)$ we mean the i^{th} row of the covariance matrix at iteration t_2, assuming the visible map was formed at iteration t_2. The subscript a indicates the subspace of the state space corresponding to the visible map. The covariance and the position of the added point will be calculated in the normal way for the compressed filter only we just calculate the i^{th} row. A careful examination of the update equations will show that equations (5.25) to (5.27) are what one would get if one had included the extra row from the formation of the visible map.

In our implementation a visible map begins with just the three pose coordinates as the state vector, $\mathbf{x}_r(t)$. As features are actually observed, they are put on the visible map. Once the visible map becomes larger than a specified size a new visible map is initialized with only the pose on it. The old visible map becomes a local map from which the current visible map can draw features. One can think of the visible map as being at the current time while the local map is at an earlier time. From that point on, as features from the older map are observed again they will have to be brought up to the current time using the update matrices for the current visible map.

Then as this new visible map becomes too large the process is repeated. There is now a chain of three maps, two local maps at earlier times and the current visible

map. Now observed features not on the visible map can be one of three types:

1. A new feature not on any local map will be simply added to the current visible map.
2. A feature from the previous, (second), local map will need to be brought up to date using the current update matrices.
3. A feature from the first local map will need to first be brought up to the time of the second local map and then up to the current time.

This process continues opening new visible maps and chaining them together, so that each map in this chain has a father and a son. When a feature is observed the visible map is asked for the feature. If the feature is not on the visible map, and thus not up to date, the visible map asks his father for the feature. This call then will travel down the chain until it gets to the map that has the feature. Then the updates are done for that feature back up the chain, with each map updating the feature and then turning it over to its son, who does the same and so on.

In figure 5.12 we have illustrated the idea. There we show three maps chained together by the update matrices ψi, ϕi and θi. The first map, $LM1$, is at a time t_1 the second $LM2$ at t_2, while the visible map, VM, is at the current time, t_3. As the robot continues down the street, wall A will come into view. At that point the feature matcher will ask VM for wall A. As VM does not have such a wall it will ask its father, $LM2$ for the wall. $LM2$ will similarly ask $LM1$. $LM1$ will update the wall with $\psi 1$, $\phi 1$ and $\theta 1$ before giving the wall to $LM2$, and so on.

It is this feature update that limits the size of the map or rather the chain. The size of the global map does not actually ever affect the complexity of the calculation unless a global update is required for some reason. Global updates are not required for pose estimation or local navigation. However if this chain of maps gets to be too long, the observation of features from a map far down or at the bottom of the chain could lead to significantly longer update times. Thus it may be necessary to stop the robot and do a global update from time to time to collapse the chain.

Experiments

We were fortunate to have access to a military urban warfare training facility outside of Stockholm. There we have a mock town with 18 mock buildings arranged along several streets. There are various vehicles and debris lying in the streets as well. The road surfaces are unpaved complete with large ruts, holes and rocks. The road is nowhere level and the slopes can be steep enough for the robot to start to slide sideways. The robot used was the ATRV robot equipped with a SICK LMS291 Laser scanner and a crossbow DMU-FOG inertial sensor. We used the fused inertial and odometry dead reckoning as described earlier.

To tune and test our SLAM program we used a few large data sets taken by driving the robot by hand. We drove the robot through a large nearly closed loop

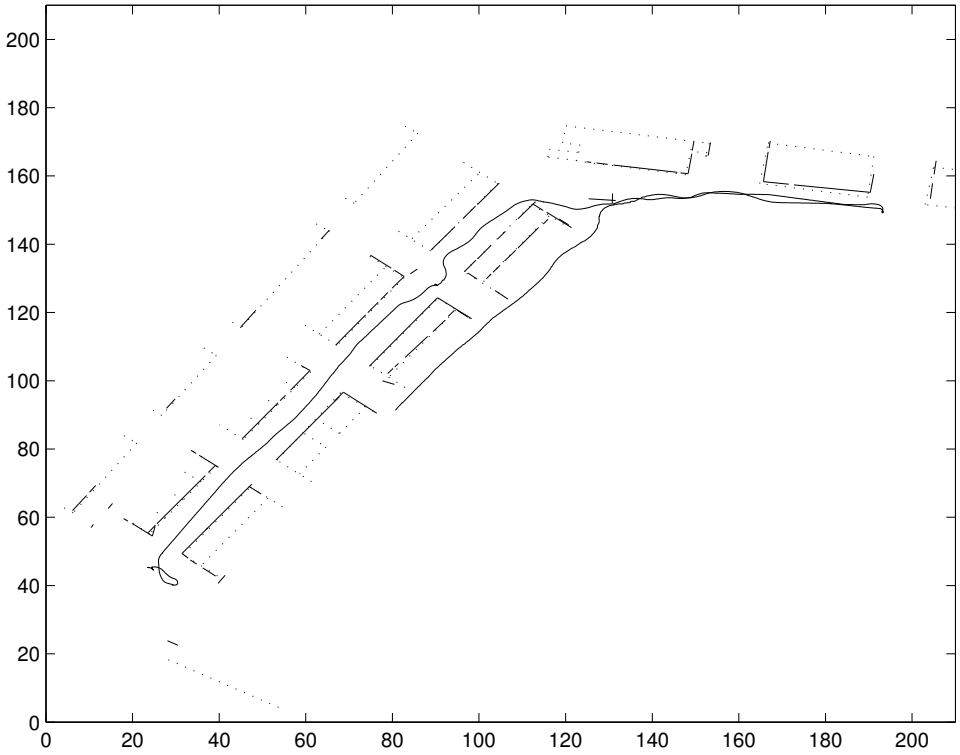

Figure 5.13: This is an example of a typical SLAM test at the mock town. The terrain behind the buildings where we start is very rough with the pitch and roll angles reaching maximums of 30, respective 10 degrees. The results show that we can build maps over the difficult terrain. The scales are in meters. The ground truth is shown as faint dotted lines for the survey map of the buildings.

in the area and the results are shown in Figure 5.13. This is a typical result from our SLAM tests in the area. The dotted lines show the survey maps of the area and the solid lines are the result of the SLAM.

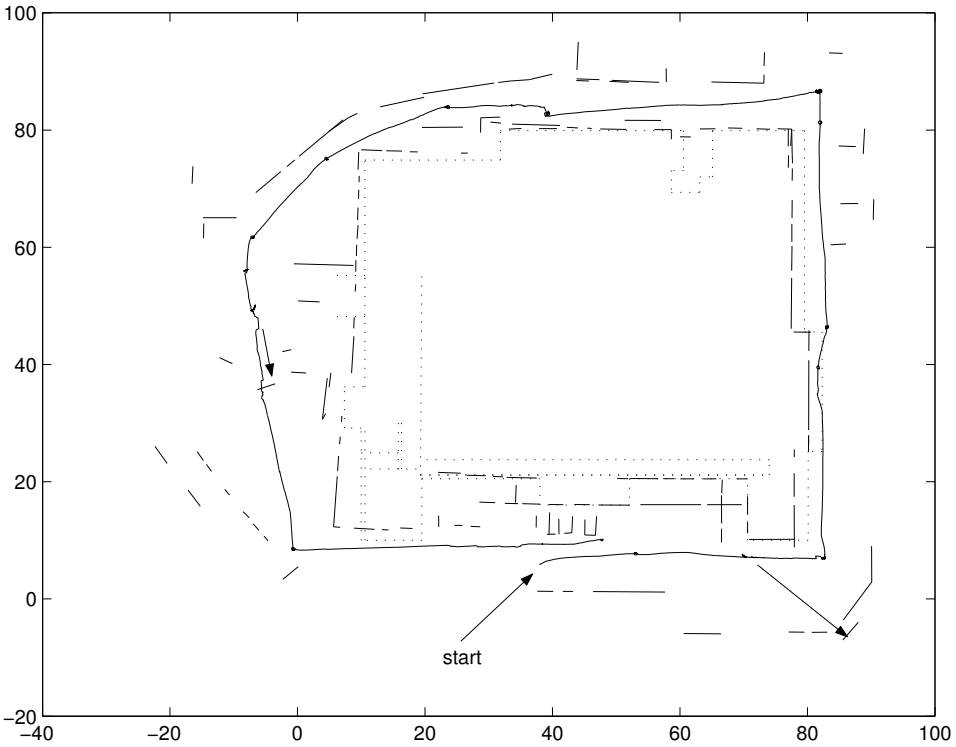

Figure 5.14: This is an example of autonomous exploration and map making. The dotted lines represent the actual building. The solid lines are the walls, fences and cars that the robot built into a map. The robot path is shown as the continuous curve encircling the building. This is also the real-time data saved at the time of the explore. Despite the errors the robot maintained a good enough pose estimate to carry out a complete exploration. Distances are in meters.

Up to this point the reader might be thinking that the SLAM problem is solved. It is true that on this data set the CEKF works very well. The reason is that this data sets had relatively easy data association and nice uniform dense features. It is particularly important that the robot has information sufficient to localize itself on the visible map. That means there needs to be walls in two linearly independent directions. In the next data set we have difficult data associations and long stretches with walls only in one direction. We also have a large loop in the map. This proved

to be beyond the limit of what a EKF/CEKF can handle.

We drove the robot outside, Figure 5.14. This building is about 70 m square with fences around most of it. The robot was taken out to a spot in the parking lot between the fence and the building. The robot was then told to explore a large rectangular region chosen to enclose the entire building. The robot then drove the path shown while simultaneously calculating its path and the map as shown. At two points human intervention was required. Once the robot was heading for a patch of grass that would be torn up by its wheels. A human stood in its way to prevent that. At another point the robot did not see a low bench with its sonar and drove too close to get away. It was driven by hand a few feet and commanded to resume exploring.

This is the same data as we showed earlier in the EKF SLAM section (see figure 5.10). That map was made using our current EKF program as opposed to this older implementation. We have gotten a bit better at filtering out some of the bad decisions made here. Figure 5.14 was made in real time at the time of the explore using the older implementation. That is why the program makes more erroneous decisions but these are very good illustrations of the real ambiguities in data filtering.

As one can see, the map has two points where errors occurred. These are shown by the arrows starting from the robot and pointing to the feature that caused the trouble. The first point was just before turning the first corner. There its position estimate acquired an error of by about 2 m. This turned out to be due to a fence that the robot could see through and was partially hidden by vegetation. This was sometimes seen as being closer and sometimes further away. We could eliminate the poor observations by making the feature threshold a bit tighter but this caused some good measurements to be lost resulting in a worse map.

The second place is similar and occurred before the last turn. There the robot was traveling down a slope and the laser scanner was seeing the ground. The ground looked much like a wall. Once the ground was put on the map as a wall the robot tried to localize with it. As the robot moved down the slope the wall seemed to move as well which was interpreted as pose errors. Making matters worse was the fact that the robot also slide sideways quite significantly while turning on this slope.

As can be seen the map in this example failed to close on return to the starting location. This was due to the problems we described. We then began working on the problem of being able to reconsider bad or rushed data associations like those that caused the first large error. We also realized that errors such as sliding sideways may still occur. Therefore, we would like to be able to somehow correct this map with the loop closing information. This was the motivation behind graphical SLAM.

This test clearly illustrates some of the challenges with EKF based systems:

- Too early commitment to data associations.
- Unmodeled drift and biases lead to serious inconsistency.
- Unimodal uncertainty model cannot capture the true stochastic map.

- Loop closing cannot be done consistently when large errors are present.

These issues form the basis for the work presented in the following chapter.

Chapter 6

Graphical SLAM

The graphical SLAM method is based on the same sort of graph that represents the Extended Information Filter, EIF [41], with some changes. The EIF graph represents the non-zero information matrix elements by edges in the graph between nodes that represent the state.

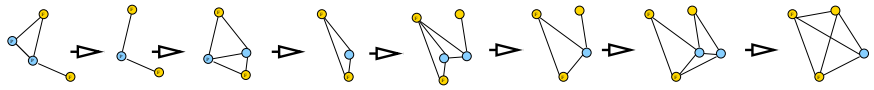

Figure 6.1: Here we illustrate the Extended Information Filter as a graph. The non-zero elements of the information matrix can be represented as edges in a graph. The nodes then correspond to the state components, pose and feature. The prior robot pose is marginalized out of the linear system producing a simpler graph. As new poses are added the older ones are eliminated which creates edges between previously unconnected feature nodes. As this proceeds the features all become connected to one another. However the links between distant features are weak.

In EIF SLAM the predict step will marginalize out the robot pose at the previous iteration and only keep the current pose in the graph see figure 6.1. When doing that the graph will gain edges between all nodes attached to the eliminated pose node. This includes the current pose node. This then leads to a dense graph/information matrix. The dense matrix makes solving for the expected state difficult.

Our method differs in two main areas. First we do not linearize the system to give a constant information matrix. Instead we recalculate the information matrix, (or Hessian of our graph energy), at each estimated state. So when the state changes so does this matrix.

The second main difference is that we do not marginalize out all the previous pose nodes which then gives us a sparse graph for which it is easier to calculate the expected state. This sparseness is not the result of any explicit sparsification step

but rather is a result of not doing the marginalization of pose nodes. Thus, we do not need to consider the consistency issues of SEIF.

We eventually do linearize and marginalize out poses from the state but we leave some poses in to maintain this sparseness. This also helps in other ways like giving each mapped area its own frame of reference. Such a frame is useful if other information is being collected along with the map an needs to move with it [62].

6.1 The SLAM Graph

The starting point for building our graph is equation (2.7) which we repeat here:

$$h_{MAP} = argmax_h\{\sum_i \ln P(\mathbf{v}_{di}|h) + \sum_i \ln P(\mathbf{v}_{fi}|h) + \sum_{j=1}^{N_f} \Lambda(n_j(h) - 1)\} \quad (6.1)$$

The SLAM problem is the problem of finding the maximum of the sum of three separate types of terms. Let us examine the first type of term. Define E_{di}, the dead-reckoning energy:

$$E_{di} = -\ln P(\mathbf{v}_{di}|h) = -\ln P(\mathbf{v}_{di}|\mathbf{x}_{r,i-1}, \mathbf{x}_{r,i}) = \frac{\xi_i k_i \xi_i}{2} \quad (6.2)$$

Here we assume an independent Gaussian model and drop an additive constant. The k_i are the inverse covariance matrices of the dead reckoning estimates and

$$\xi_i = T(\mathbf{x}_{r,i}|\mathbf{x}_{r,i-1}) - \delta_i. \quad (6.3)$$

Here T denotes the non-linear transformation to the robot frame at $\mathbf{x}_{r,i-1}$. The term δ_i is the estimated motion in the robot frame. We note that the EIF, (and EKF), approach would be to now linearize ξ_i and absorb all the Jacobian terms into the constant k_i. We will take a different approach, keeping this non-linear expression and introducing a new type of node to store this function, an energy node. The energy node will calculate this nonlinear E_{di} based on the current value of the pose states $\mathbf{x}_{r,i}$ and $\mathbf{x}_{r,i-1}$. These pose states will be stored in state nodes as before in the EIF graph. So instead of constant matrix elements represented by edges between two state nodes we have edges from state nodes to energy nodes. The energy nodes then can represent any general function of the attached states.

We can now look at the last two terms in equation (6.1). They give the energy of the feature measurements,

$$E_{fi} = -\ln P(\mathbf{v}_{fi}|h) - \kappa\Lambda = -\ln P(\mathbf{v}_{fi}|\mathbf{x}_r, \mathbf{x}_f) - \kappa\Lambda \quad (6.4)$$

where κ is 0 for a new feature or the dimension of the measurement if the measurement is matched to an existing feature. The Λ term gives some preference for using existing features to explain new measurements. Thus if the energy, $E_{fi} < 0$, the match can be considered to be correct.

We can now represent equation (6.1) by a graph where the *state nodes* represent the states, $\mathbf{x}_r$ and $\mathbf{x}_f$. Each term of the sum over measurement likelihoods corresponds to an *energy node*. The data association hypothesis is represented by edges connecting the state nodes to energy nodes. One can test various combinations of data associations by attaching edges between nodes in the graph.

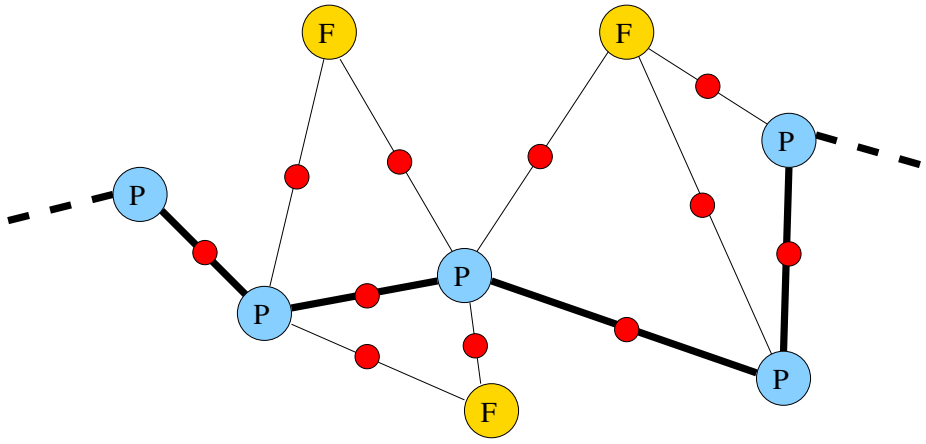

Figure 6.2: This shows how a graph is built up. The lighter (blue) pose nodes are labeled P, the feature nodes (yellow) F. These are the two types of state nodes. The darker (red) smaller circles are the energy nodes representing measurements. We sometimes focus attention to the subgraph of just pose nodes and the energy nodes connecting them. This subgraph is shown with heavier edges here. Th energy nodes between the pose nodes are from dead-reckoning and the ones between feature and pose nodes are from feature measurements.

The total energy is composed of two terms, one is related to the uncertainty in dead-reckoning and another from feature measurements, as in equations 6.4 and 6.2.

$$E = \sum_i E_{di} + \sum_i E_{fi}. \tag{6.5}$$

We now want to find the data association h_a and values for the state nodes, $\{\mathbf{x}_r\}$ and $\{\mathbf{x}_f\}$, that minimizes the total energy.

Nodes and edges can be added immediately when observing a new feature. In the M-Space model, new features will not have an effect on the map, ($E_{fi} = 0$), until they collect enough information to initialize some of the M-Space dimensions. At such a time all the edges of the feature can be tested again to determine if they lower the energy, ($E_{fi} < 0$). Thus the measurements are not added in any final way

but can be removed later. The advantage is here that association can be added or removed at any time, by inserting or deleting arcs from the graph.

We evaluate the feature energies by first defining the innovation function, $\eta(\mathbf{v}, \mathbf{x}_o)$, as in equation (4.10). If the η are Gaussian with variance $C_{\eta\eta}$ and mean 0, then the energy is given by:

$$E_f = \frac{1}{2}\eta(\mathbf{x}_o)^T C_{\eta\eta}^{-1} \cdot \eta(\mathbf{x}_o) - \Lambda\kappa, \tag{6.6}$$

This is similar to the dead-reckoning energy except for the κ term. Putting this together, we can write the gradient G_s for the energy with respect to the state s as.

$$G_s = \eta^T C_{\eta\eta}^{-1} \begin{pmatrix} J_{\eta o} J_{or} & J_{\eta o} J_{of} \widetilde{B}_f \end{pmatrix} = \eta^T C_{\eta\eta}^{-1} (D_o \eta D^o) \tag{6.7}$$

For the dead reckoning we get a very similar result with the B matrix equal to the identity and with the robot pose coordinates replacing the feature coordinates. We note that the s subscript will refer to the state coordinates, which are specifically $\delta\mathbf{x}_r$ and $\delta\mathbf{x}_p$. We can also calculate the Hessian of the energy.

$$H_{ss} = \begin{pmatrix} J_{or}^T J_{\eta o}^T \\ \widetilde{B}_f^T J_{of}^T J_{\eta o}^T \end{pmatrix} C_{\eta\eta}^{-1} \begin{pmatrix} J_{\eta o} J_{or} & J_{\eta o} J_{of} \widetilde{B}_f \end{pmatrix} = (D_o \eta D^o)^T C_{\eta\eta}^{-1} (D_o \eta D^o) \tag{6.8}$$

These quantities, E, H_{ss} and G_s, are calculated by the energy nodes based on the state nodes. They are the fundamental quantities of the graphical SLAM method. The Hessian is essentially the information matrix while the gradient sets the expectation value of the state given this data association. That is for the linearized system around the current state. The nonlinear system has many local minima. This is related to the multi-modal probability distribution that underlies the model. One finds that for instance two separate robot paths can be reasonable given the measurements. From then on the distribution will have a mode associated with each of these choices for the path. Later as more measurements are added to the system modes will become more isolated from one another as the later hypothesizes will not be compatible between modes.

We evaluate these terms each time we try to estimate the MAP state. Thus we make no approximations. Unfortunately, we also cannot find a closed form solution. That is why we must relax the graph to approach the minimum energy solution. Fortunately, in many cases, this can be done more quickly than finding the exact solution to the linearized system. For instance for very large systems where we are exploring new areas we can relax the graph locally and our complexity will not depend on the size of the system.

6.2 Graph Relaxation

For the purpose of our explanation we consider a single state node and its attached energy nodes. These energy nodes calculate the non-linear energy function and its derivatives at the current state. We sum the contributions to the gradient and Hessian from each attached energy node and use them to relax the state node:

$$H_{ss}\Delta\mathbf{x}_s = -G_s \tag{6.9}$$

We repeat this for all the nodes in some section of the graph starting with the set of state nodes attached to added energy nodes. If we had a quadratic energy this would amount to a Gauss-Seidel iteration for this subset of the full system.

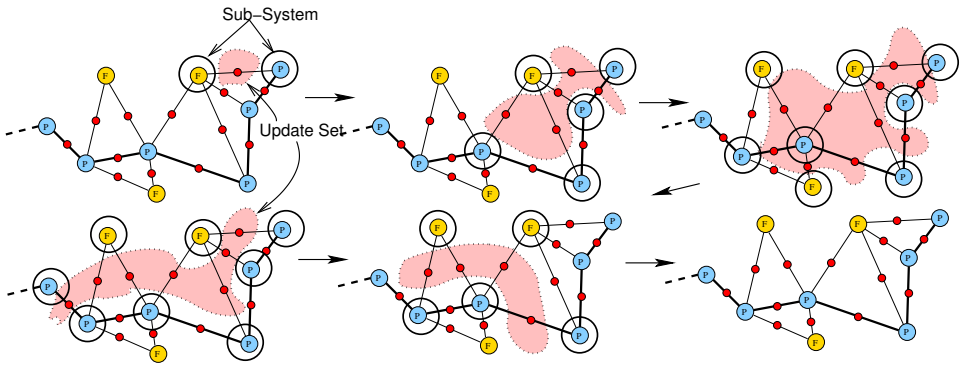

Figure 6.3: This figure illustrates 6 iterations of the relaxation algorithm. The update set starts containing the newly added energy node shown enclosed in the shaded region. The sub-system is shown by ringing in each of its state nodes. On the next iteration the update set has grown to include those energy nodes with significant change after relaxing the sub-system in the first iteration.

To clarify, the basic graph relaxation, (see figure 6.3), after adding a new connection to an energy node is:

1. Set an *update set* of energy nodes = the newly added energy node.
2. Form a set of state nodes, *sub-system*, out of all state nodes attached to an energy node of the *update set*
3. Relax the state nodes of the *sub-system* one at a time, Eq. 6.9.
4. Clear the *update set*
5. For all energy nodes attached to the *sub-system*; If the energy change was significant add to the *update set*.

6. If the *update set* is not empty goto 2.
   Else done.

For a significant change to an energy node we first require that the magnitude of the energy change be greater than an nominal small amount, 0.01. Most energy nodes will be eliminated by that test. Next we test the ratio of the magnitude of the change in energy to the energy after relaxation. If this relative change is larger than a threshold, (we used 5 percent), we consider the change significant.

This way of organizing the updates requires no extra work in testing for significant change as the quantities tested have already been calculated. The update will terminate quickly if little tension is added to the graph but can also spread quickly though the graph to where relaxation is needed.

In practice one needs to limit the number of times we run through the graph relaxation steps. It is better to relax a little each iteration than to spend too much time relaxing on any single iteration. In this way the cumulative effect will be to drive the system toward the minimum while assuring real-time constraints are met. Also this tend to low pass filter some of the noise from individual measurements. We also check that equation (6.9) actually reduces the energy. If not we reduce the step size until it does.

We have one other method to do relaxation on the graph which takes advantage of the special structure of the graph. The graph is a chain of pose nodes linked together with each pose node attached to some feature nodes. If a stress is added to the graph at one end of this chain, it can be difficult to propagate the stress along the chain. The stress tends to be absorbed mostly at the end of the chain and then become so small that it never distributes itself optimally along the chain.

To solve this we form a sub-system out of just a number of pose nodes going back from the current pose node. This number will vary between 20 to 300 depending on the current feature measurements. This subsystem includes no features. The Hessian, H, of the subsystem is block tri-diagonal with 3 x 3 blocks. This system can be solved easily with care given to the numerical stability. We don't solve the block tridiagonal system $Hx = -G$ directly but rather $H'x' = -G'$. Where $H' = P^T H P$, $x = Px'$ and $G' = P^T G$. P is given by,

$$P = \begin{pmatrix} I & 0 & 0 & ... \\ I & I & 0 & ... \\ I & I & I & ... \\ ... & ... & ... & ... \end{pmatrix} \tag{6.10}$$

and I is the block identity matrix. This can be thought of as changing from the global pose variable to incremental ones. We actually are using the difference in the coordinates of the poses which are not the same as the coordinates of the relative poses. For small movements between nodes this is approximately the same. By doing this we make H' strongly block diagonally dominant. This is so because of the form of the dead-reckoning energy term which is always present and is the only contributor to the off-diagonal blocks in the original sub-system. There is 100% fill

produced by P but by being careful to exploit the special structure of the original tridiagonal system the complexity remains linear in the number of nodes in the subsystem. Periodically, say every 25 iterations, we relax the graph in this way. We then relax the features attached to the subsystem. The subsystem used is the *tail* of the graph to be defined later.

The graph relaxation takes a time that depends on the amount of stress added to the graph. It does not depend on the size of the graph as the calculation remains local. While the robot is exploring new areas the amount of stress added is very small and updates are fast. If the robot turns around and backtracks there can be some stresses that will cause a longer update time. Closing loops can create even more stress and can cause the updates to take more time than one sensing cycle to complete. We therefore maintain a buffer for the input and allow the program to run asynchronously with the data coming in.

6.3 Feature Matching

The basic criteria for feature matching is that the energy should be minimized. The advantage with the graphical method is that the match can be reconsidered at any time. As each new measurement node is added to the graph and after we update the path of the robot, we can remove and then add all the dense information for a feature. This is done so as to have the information added along a continuous and correct robot path. We then try to extend the feature's p-dim. If this succeeds, (ie. we can initialize some part of the feature), we will automatically start to use all the measurements that we attached to the feature node. That is to say the measurement nodes will begin to calculate energies using the newly initialized p-dim. At this time we check each measurement node attached to the feature node to determine if the energy is greater that Λ times the p-dim. If so we remove the measurement node and its edges from the graph.

For initialized features we check the energy change when adding a new measurement node to the feature. If this change after relaxing the graph is greater that Λ times the p-dim we remove the node and its edges and restore the state.

Notice that we can add the features to the graph after the first observation. While the p-dim is 0 the energy returned by these measurement nodes will be identically 0. As soon as the p-dim grows the information stored in these nodes will be used to calculate the energy.

If the feature is never initialized it will eventually be removed from the graph. Periodically the features are checked to see if they can be deleted. A feature can be deleted if it has 0 M-Space dimension and was last observed at at a point that is considered too far from the current robot position, a few meters.[1]

[1] The features are maintained by a manager object, the feature bank, and no other object stores pointers to features, including the feature nodes. They instead store the key associated with the feature and ask the bank for a pointer to the feature as needed. When the feature is deleted it notifies the bank. If any object then asks for the deleted feature it receives a null pointer. The

Experiments

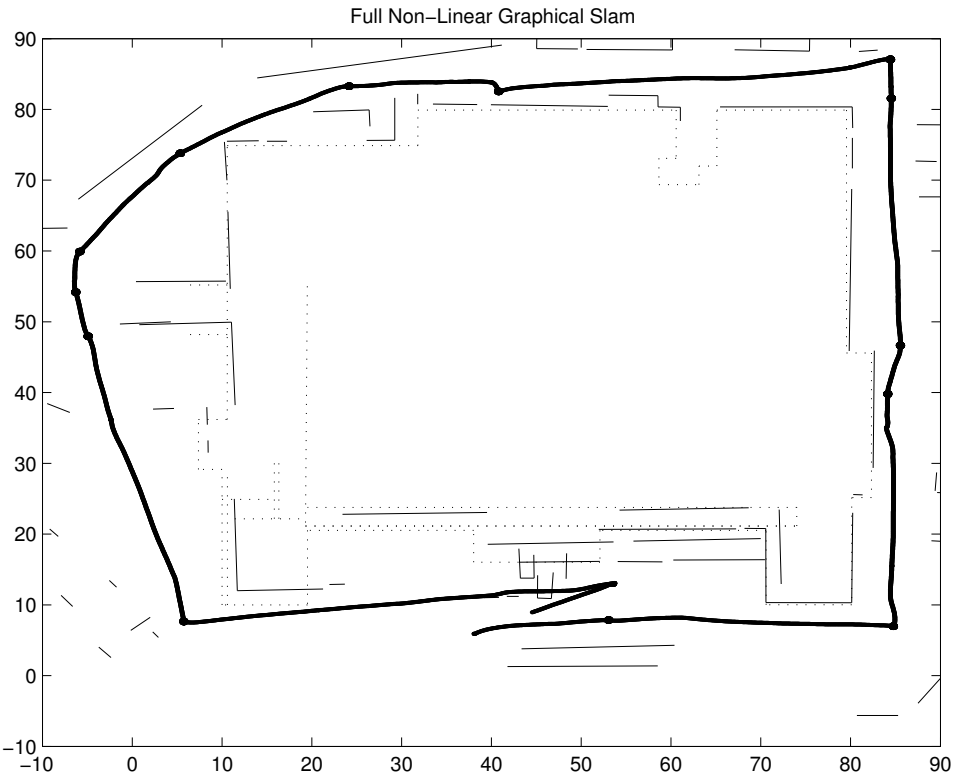

Figure 6.4: Here is a map made using the exact nonlinear energy for the system. The robot slid sideways at one point which caused the error in closing the loop. We see that the data association problems are now not present.

In figure 6.4 we show a map made using the ideas described so far. We can now see how the data set that the CEKF could not solve is handled by the graphical method. The robot drove autonomously around a building in a counter clockwise sense. There are 7,500 pose nodes along the path and 11,975 feature measurements. The graph updates took 225 seconds total or 30 ms on average. This is not counting

features may be deleted for any number of reasons and the other objects may not be informed. So if a feature node is found to point to a deleted feature it too will be removed from the graph and deleted as will any edges and energy nodes associated with the feature node. The entire list of feature nodes is periodically checked for this situation. Also, if one tries to calculate an energy using a deleted feature the feature node will ask for the feature to calculate its current state. At that point the calculation will fail and the node will be removed.

the feature extraction which is done by a separate module. The time between scans was 200 ms so the average utilization was 15%. This was on a 550 MHz Pentium III processor.

We see that the data association problems that caused problems for the CEKF implementation are no longer a problem. This was true without any special tuning or filtering. The false matches were simply removed as a natural consequence of the energy increase they eventually caused. This eliminated most of the problems that the CEKF implementation had with this data. The map appears to have fewer and smaller errors than the EKF and CEKF maps made with this data shown in figures 5.10 and 5.14.

The robot slid sideways while turning near the middle of the forth side of the rectangular path. This caused a large error and the robot failed to close the loop. If we now simply match the correct features from the end and beginning of the loop and relax the graph as described, we will find that only the ends of the path change. The stress is very quickly absorbed and will not propagate around the graph. We found that there are simple too many dimensions in this system to solve it efficiently so in the next section we will simplify the graph to solve the constraints more easily.

6.4 Graph Reduction by Star Nodes

In this section we will show how we can reduce the number of pose nodes while maintaining the local invariances and symmetries. The extent of the approximations will be to assume that the local map's relative geometry can be approximated linearly.

Refer to figure 6.5. We have selected a set of energy nodes to be combined into a new kind of energy node, a star node. The energy of the star node will be an approximation of the sum of the energies of all the energy nodes in the set. Therefore the energy of the star depends on the states of all the attached state nodes.

If some state node only has edges to one star node, it can be marginalized out of the energy formula for the star node. This is then a simplification of the graph. This step will create a direct coupling between all the remaining state nodes of the star, so that the star Hessian will be totally filled. The stars are sparsely connected to the rest of the graph so that the global Hessian remains sparse.

The energy for the star node is initially the Taylor expansion around $\bar{\mathbf{x}}_\mathbf{s}$:

$$E = E(\bar{\mathbf{x}}_\mathbf{s}) + G_s(\bar{\mathbf{x}}_\mathbf{s}) \cdot \begin{pmatrix} \delta\mathbf{x}_b \\ \delta\mathbf{x}_q \end{pmatrix} + \begin{pmatrix} \delta\mathbf{x}_b^T & \delta\mathbf{x}_q^T \end{pmatrix} H_{ss}(\bar{\mathbf{x}}_\mathbf{s}) \cdot \begin{pmatrix} \delta\mathbf{x}_b \\ \delta\mathbf{x}_q \end{pmatrix} \tag{6.11}$$

Here we have split the state perturbation vector $\delta\mathbf{x}_s$ into $\delta\mathbf{x}_b$ and $\delta\mathbf{x}_q$. The $\mathbf{x}_b$ is a pose node coordinate which is to become the 'base' pose of the star and $\delta\mathbf{x}_q$ are the other pose and feature coordinates. We now transform all coordinates to the base frame so that they become $\delta\mathbf{x}_o = T(\delta\mathbf{x}_q|\mathbf{x}_b)$. We can write the following identities:

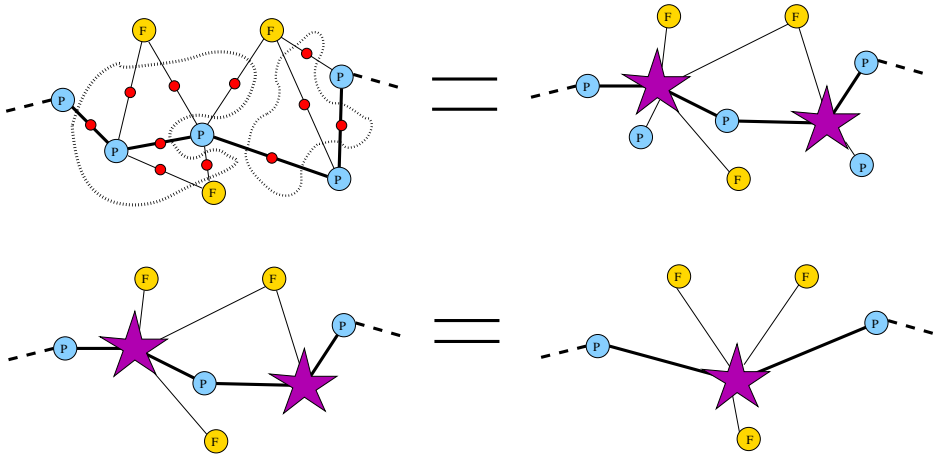

Figure 6.5: This shows how a graph is reduced. Starting from the basic graph we linearize all the energy nodes (enclosed by the dotted curves) attached to every other pose node into a star. We then eliminate the pose nodes which have only one edge. We then repeat with the pose node between two stars to merge them.

$$I = \begin{pmatrix} I & 0 \\ -BJ_{of}^{-1}J_{ob} & BJ_{of}^{-1} \end{pmatrix} \begin{pmatrix} I & 0 \\ J_{ob} & J_{of}\widetilde{B} \end{pmatrix} \tag{6.12}$$

$$\begin{pmatrix} \delta\mathbf{x}_b \\ \delta\mathbf{x}_o \end{pmatrix} = \begin{pmatrix} I & 0 \\ J_{ob} & J_{of}\widetilde{B} \end{pmatrix} \cdot \begin{pmatrix} \delta\mathbf{x}_b \\ \delta\mathbf{x}_q \end{pmatrix} \tag{6.13}$$

$$G_s \begin{pmatrix} \delta\mathbf{x}_b \\ \delta\mathbf{x}_q \end{pmatrix} = G_s \begin{pmatrix} I & 0 \\ -BJ_{of}^{-1}J_{ob} & BJ_{of}^{-1} \end{pmatrix} \begin{pmatrix} \delta\mathbf{x}_b \\ \delta\mathbf{x}_o \end{pmatrix}. \tag{6.14}$$

The B^2 and J_{of} matrices in equation (6.14) are block diagonal matrices with blocks for each state node that is connected to the set of energy nodes. The J_{ob} is a column of blocks. As a result of the invariance of the original energy nodes under transformations of all the state nodes to a new frame, the matrix multiplying $\delta\mathbf{x}_b$ above must vanish identically. Thus we can always find the full G_s from the G_q and the symmetry matrices:

$$G_s = \begin{pmatrix} G_b & G_q \end{pmatrix} = G_q \begin{pmatrix} BJ_{of}^{-1} \end{pmatrix} \begin{pmatrix} J_{ob} & J_{of}\widetilde{B} \end{pmatrix} \tag{6.15}$$

We define $Q = BJ_{of}^{-1}$ and $\widetilde{Q} = (J_{ob}, J_{of}\widetilde{B})$. The invariance relation is then written more compactly as:

[2] For the pose nodes we can set $B = I$, the identity matrix, and $\mathbf{x}_f = \mathbf{x}_r$

$$G_s(\bar{\mathbf{x}}_s) = \{G_q(\bar{\mathbf{x}}_s)Q(\bar{\mathbf{x}}_s)\}\widetilde{Q}(\bar{\mathbf{x}}_s). \tag{6.16}$$

We see that the $\widetilde{Q} = (J_{ob}, J_{of}\widetilde{B}) = D^o$ where the base frame now acts as a robot or sensor frame for all the measurements. Even those taken from other robot poses. The $\{G_q(\bar{\mathbf{x}}_s)Q(\bar{\mathbf{x}}_s)\} = D_oE$ in the same way. So q derivatives times Q are relative derivatives.

$$G_s(\bar{\mathbf{x}}_s) = D_oE(\bar{\mathbf{x}}_s)D^o. \tag{6.17}$$

We can now do the same for the Hessian,

$$H_{ss}(\bar{\mathbf{x}}_s) = \widetilde{Q}(\bar{\mathbf{x}}_s)^T\{Q(\bar{\mathbf{x}}_s)^T H_{qq}(\bar{\mathbf{x}}_s)Q(\bar{\mathbf{x}}_s)\}\widetilde{Q}(\bar{\mathbf{x}}_s). \tag{6.18}$$

We now take advantage of our freedom to chose the linearization point. We chose this so that the gradient vanishes. This means that the star node is linearized around the state that gives it the lowest energy.

The H_{qq} matrix is the Hessian at the linearization point without the rows and columns of the base node. The Hessian might not have full rank as it includes only part of the measurements for those feature nodes attached to it. We want to have an explicitly stable representation so we do a singular value decomposition of H_{qq}.

$$H_{qq} = \sum_{k=1}^{m} \mathbf{U}_k \lambda_k \mathbf{U}_k^T \tag{6.19}$$

Here k runs over the m positive eigenvalues λ and $\mathbf{U}_k$ are the eigenvectors. Define the projection into the eigenspace:

$$\bar{u}_k = \mathbf{U}_k^T Q\bar{\mathbf{x}}_o, \quad u_k = \mathbf{U}_k^T Q\mathbf{x}_o, \quad \Delta u_k = u_k - \bar{u}_k \tag{6.20}$$

We can now write the energy for the star node in a compact invariant and stable form,

$$E = E(\mathbf{u}) + \sum_{k=1}^{m} \frac{\lambda_k}{2}(\Delta u_k)^2 \tag{6.21}$$

We can also calculate the derivatives of the energy around any new state. So for example the gradient at $\mathbf{x}_s$ becomes

$$G_s(\mathbf{x}_s) = \sum_{k=1}^{m} \frac{\Delta u_k(\mathbf{x}_s))\lambda_k}{2} U_k^T Q(\bar{\mathbf{x}}_s)\widetilde{Q}(\mathbf{x}_s). \tag{6.22}$$

The Hessian is

$$H_{ss}(\mathbf{x}_s) = \sum_{k=1}^{m} \widetilde{Q}^T(\mathbf{x}_s)Q^T(\bar{\mathbf{x}}_s)U_k \frac{\lambda_k}{2} U_k^T Q(\bar{\mathbf{x}}_s)\widetilde{Q}(\mathbf{x}_s) \tag{6.23}$$

We have not included any term for the derivatives of $\widetilde{Q}(\mathbf{x}_s)$. There are two motivations for this. First such a term will be a product with $\Delta u_k(\mathbf{x}_s)$ and thus be relatively small and zero at the star equilibrium point. Second including such a term will allow the possibility that the Hessian could have negative eigenvalues, something that complicates our graph relaxation. Without it the Hessian is positive semi-definite. Computations with and without including these derivatives have been performed for the empirical evaluation. The differences are too small to warrant the extra work.

The eigenvalues and eigenvectors are saved along with $Q(\bar{\mathbf{x}}_s)$ and $\bar{u}_k$. These can then be used to find the energy for any new state.

6.5 Star Formation

We now address the problem of how to collect the information into star nodes in a structured way. We will define four sections of the graph in terms of sets of nodes. We want to reduce the number of nodes in the graph by forming stars. At the same time we need to limit the dimensionality of the stars or the calculations needed to form them will be too complex. The difficult step is solving for the eigenvectors.

As we have said we will not form stars until the state nodes are sufficiently mature. In practice this means we maintain a sliding window including the last N_{tail} pose nodes and try to form stars only beyond this window. The number of nodes we go back, N_{tail}, will vary depending on the context as discussed later. We select a mature pose node and all its attached energy nodes are collected into a star as shown in figure 6.5. We continue selecting pose nodes and combining the energy nodes into stars but we always leave one pose node between each of the stars as in figure 6.5. The stars formed by combining the ordinary energy nodes is refereed to as a level 0 star. Two level zero stars can be combined into a level 1 star by eliminating the pose node between them.

We continue in this way combining stars at the same level to form stars of one higher level. We stop when some criteria on the size is reached. That might be the distance between the two poses attached to the star, the dimension of the star or the level of the star.

We now define the four sections of the graph, figure 6.6. One section is the *tail*, the current robot pose node and all the pose nodes before it back to until we come to a star node. While making the graph the robot appears to be dragging a tail of pose nodes behind it back to this star node. In reality the pose nodes are being created at one end and eliminated by forming stars at the other and are not moving.

Another section is the *formation set*. It is here that stars are still being built up level by level. The formation set is a simple chain of alternating pose and star nodes.

As stars in the *formation set* get to large to grow anymore they are moved out into the *loop set*. It is the *loop set* that is compared to older sections of the graph

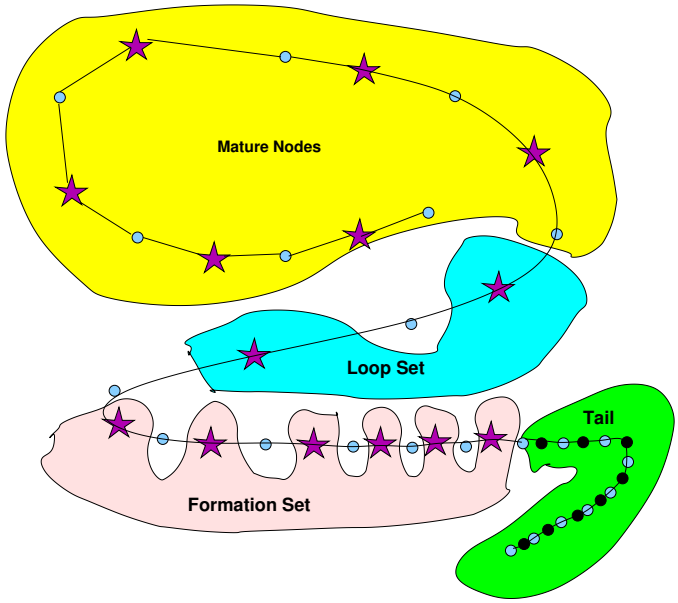

Figure 6.6: This figure illustrates the four sections of the graph. No feature nodes are shown for clarity. The tail is the newest section of the graph where no star nodes have yet been formed. The formation set is where the stars are being built up until they become too large to grow any more. They are then moved to the loop set which is matched to the mature nodes to find possible loops in the graph. After searching for loops the stars from the loop set are moved to the mature nodes.

to find potential loops in the robot path. These loops then represent constraints on the graph. Stars are removed from the *loop set* and put into the forth set, the *mature nodes* as described later.

As we have said the length of the *tail*, N_{tail}, will vary. It depends on the current features being observed. While we are still collecting information for initializing the features attached to a pose node we will not move it from the *tail* to the $formation$ *set*. That is so that we can adjust the $dense\ information$

Stars in the *loop set* and from the *mature nodes* that share feature nodes can be combined in order to build in the topological constraint in the graph. This should only be done if the features they have in common are sufficient to define the transformation between the base frames.

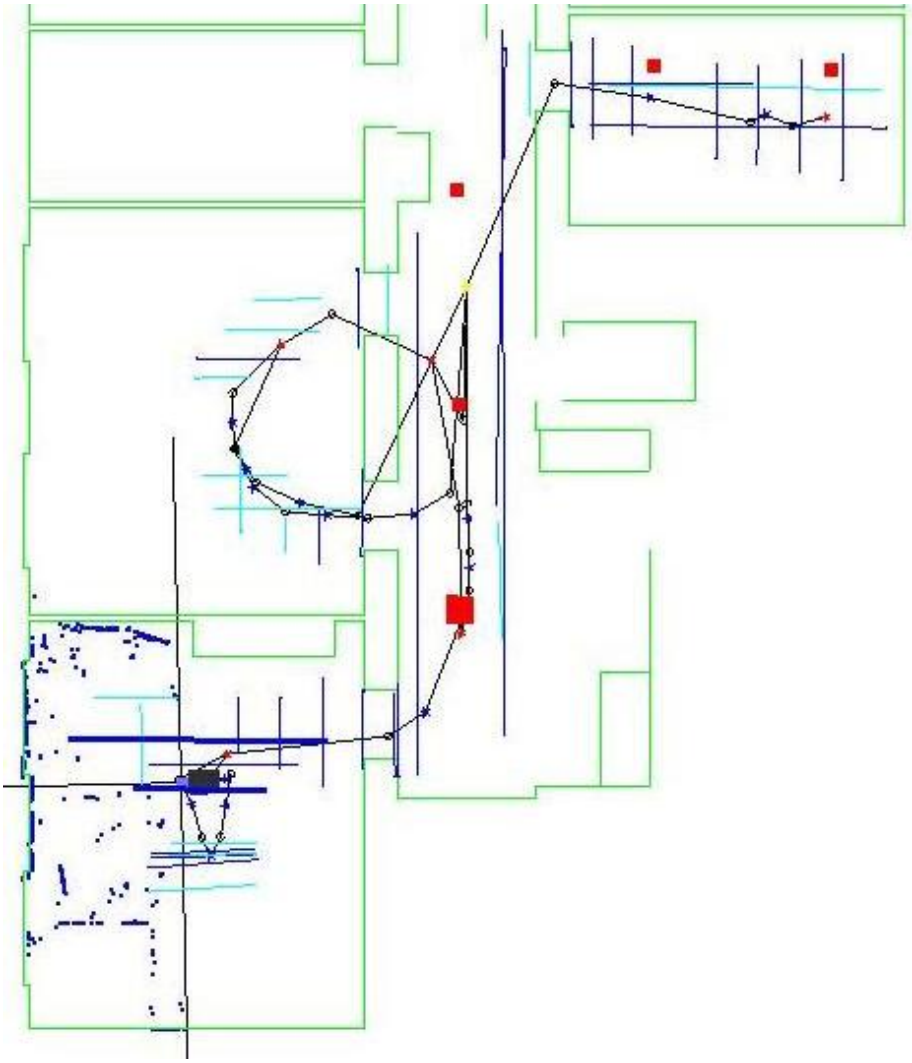

Figure 6.7: Here is the map made using the star nodes and vision features indoors. The size of the mapped area is about 13 by 15 meters. The robot path stars and ends in the room at the lower left. The raw laser scan data is displayed as a check on the final localization accuracy. By comparing it to the hand made map shown by the lighter (green) lines, one can assert that the final location was accurate to a less than 5 cm.

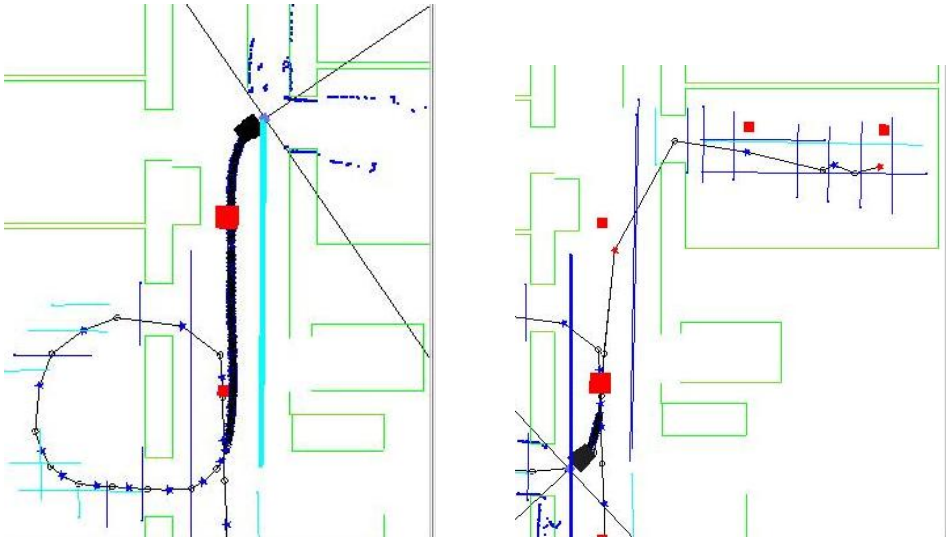

Figure 6.8: Here we show the map at two different stages of being made. On the left we see that the tail has been allowed to grow to the entire length of the ceiling line that the robot is about to pass under. At this time that line is still 1D (light blue), orientation only. It has been used to keep the robot oriented in the corridor. When the robot passes under the line it can be fully initialized to 3D (dark blue). As the tail extends to the first observation of this line all of the observations can now be used. To the right we show the situation on a return to the corridor. As the line is now fully initialized there is no need to have the tail long.

Experiments

Figure 6.7 show a map made using only vision features and the star nodes. Here we see how at some places the stars were formed at points where the robot crossed its previous path. Some of these stars connect the two section of the path so that they attach more than two pose nodes. In particular we see one star node that has consolidated all the measurements form the paths in the corridor leading from the larger room on the left to the smaller room on the top right corner and back. This lead to the long edge connecting a pose node in the doorway of the smaller room to the star node just outside the larger room between its two doors.

Figure 6.8 illustrates the advantage of varying the length of the tail. Here a line on the ceiling was observed for a rather long distance before the robot was able to triangulate the height of the line. By keeping all the pose nodes from the first observation up to the current time no information was lost. When the line is finally initialized all the energy nodes along the tail can use the full 3D M-Space feature. At that time the tail can be made much shorter as shown on the right.

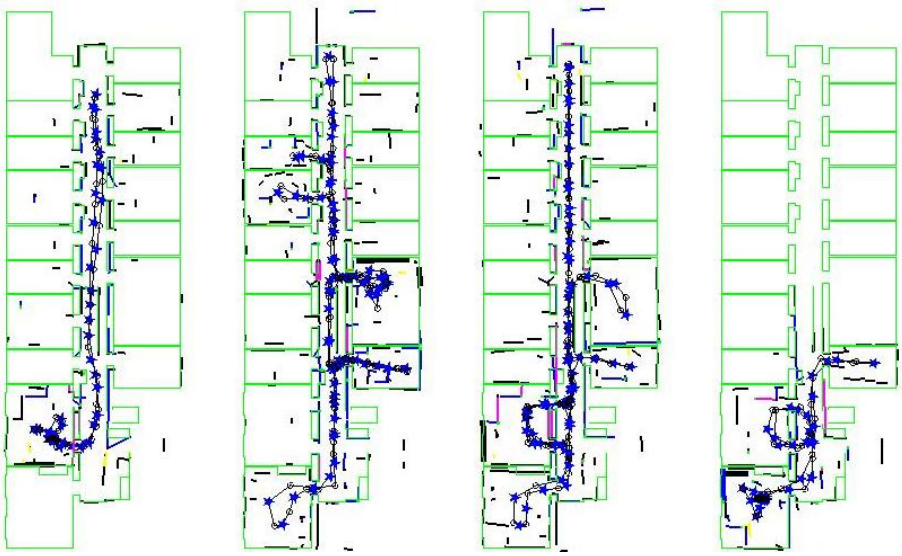

Figure 6.9: Four Graphical SLAM maps made by from left to right a PeopleBot, Pioneer, PowerBot and a custom built robot. The number of walls in the maps were 155, 219, 192 and 78 respectively. The lighter (green) lines represent the hand made map while the darker (black) lines are the SLAM map. The size of this building is 13 by 39 meters.

We also evaluated the Graphical SLAM implementation on the four data sets taken in our lab with the four different robots. For this test we made no changes to any of the parameters for odometry model or feature initialization and matching from the ones used in the earlier EKF tests (see figure 5.2). The only parameters we picked were the ones particular to the Graphical SLAM algorithm. The resulting maps are shown in figure 6.9.

These data sets were rather well suited to the EKF algorithm as the robot is able to stay well localized the entire time and there are no large loops in the paths. This resulted in no large corrections to the robot pose and thus the linearizations were done reasonably consistently. The graphical SLAM algorithm made good maps as well but had more trouble. The backtracking of these paths caused the tension in the graph to increase. This made relaxation harder. The EKF with its closed form global solution did not suffer from this problem.

In addition the data association here was relatively easy. This meant that the graphical method's more flexible data association and feature initialization was not a significant advantage here.

A final point was the order of the experiments. We have tuned all the M-Space feature parameters for matching, tracking and initialization of features as well as

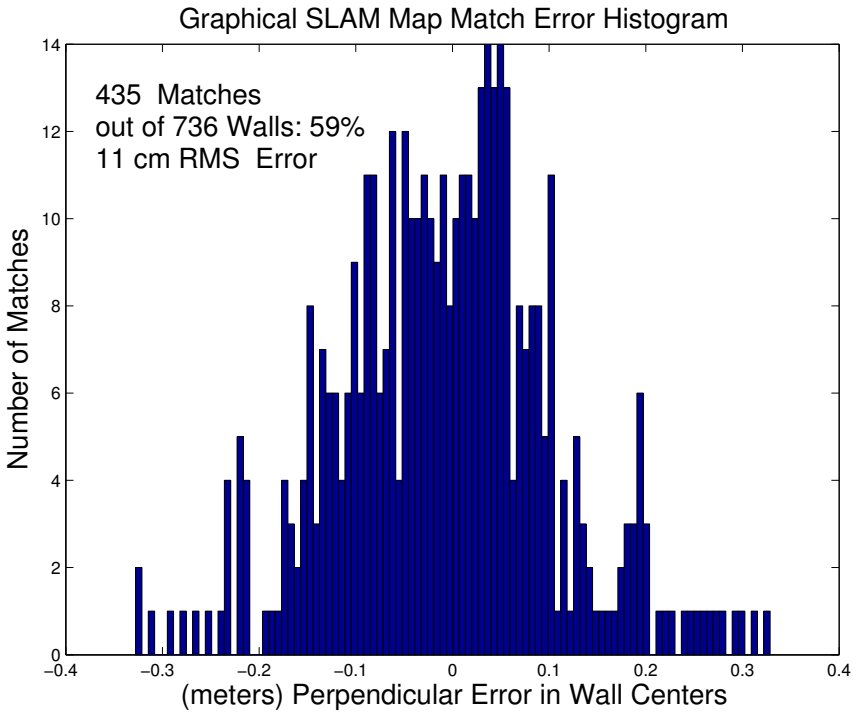

Figure 6.10: Here we see the distribution of the differences between the location of the wall centers after matching the maps pairwise. The walls were matched using a simple matching of all walls within 50 cm and 6 degrees. We matched all 6 pairs of maps and cumulated the results in this histogram. 59% of the walls matched. The RMS Error in the perpendicular distance of the wall centers was 11 cm.

the parameters of the odometry models using the EKF program. These then were naturally well tuned to give good results for that implementation. We did not re-tune these parameters for the Graphical SLAM evaluations. We feel that this make the comparison more meaningful than having it depend on a large amount of tuning of parameters. It does favor the EKF implementation somewhat.

The result was that the 4 graphical SLAM maps compared almost as well with one another as the EKF ones did, (compare figures 6.10 and 5.3). The RMS error when matching these 6 pairs of maps was 11.1 cm. This was about 10% worse then the EKF maps. Additionally fewer walls could be matched between these maps(compare figures 6.11 and 5.4). We were able to match only 59% of the walls here while 72% of the EKF walls could be matched between maps.

This was a good illustration of when the EKF algorithm works very well. It also

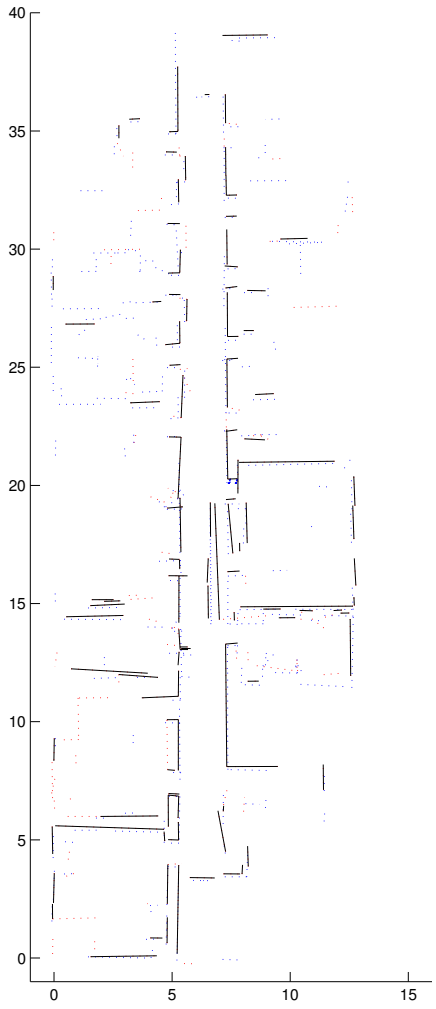

Figure 6.11: This illustrates the extent of the matched sections between the two largest maps. The walls of the original two maps are shown dotted while the matched walls are in solid black. Distances are shown in meters.

is a good illustration of how the M-Space representation allows us to keep most of the implementation and its parameters the same in order to better compare two SLAM algorithms. Later we will look at some more challenging situations in which the Graphical method outperforms the EKF solution.

6.6 Solving the Topological Constraints

Now we return to the problem of closing the loop. We assume here that we have found a set of features attached to the loop set stars that match other features from an older section of the graph. This set must be sufficient to define the transformation between the two sections.

We combine star nodes from the loop set until we have a single star with enough features matched to the older section to define the transformation. We do the same for the star nodes in the older section. The result is a pair of star nodes that can be combined to build the constrain into the graph. However, we do not combine them yet. We must first adjust them to agree.

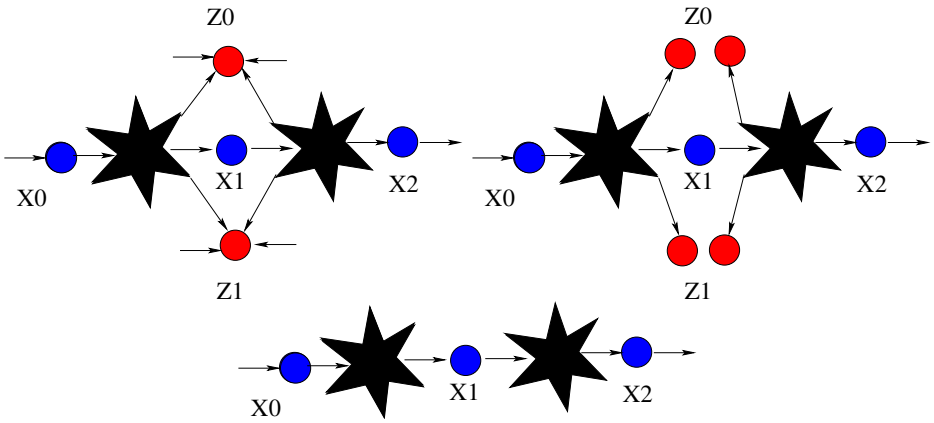

Figure 6.12: This shows how a graph is simplified temporarily in order to more easily calculate a solution to the loop constraint.

The tail is cut off from the rest of the graph. Then we solve the constraint for an approximate system. Figure 6.12 illustrates the approximate system that we use. We want to eliminate the off diagonal terms of the Hessian matrix for the total energy of the graph. We notice that if the feature nodes that two stars have in common are instead considered to be different features then the features can all be marginalized away and the graph simplified to only pose and star nodes. The Hessian will then be block diagonal with mostly 3 x 3 blocks. This is the system we solve the constraint equation for.

Having done that we restore the graph to its original feature match hypothesis and fine tune the solution by relaxation. We do that by reattaching the edges one at a time starting with the strongest edges, the ones with the most measurements. The tail is added as a last step, also one node and edge at at time.

It is only now that the pair of star nodes that define the loop can be merged into a single star node. After doing this there will be a cycle in the graph that identifies the loop constraint.

Let us look more closely at the solution of the approximate system. We need to marginalize out the feature nodes from the star node energy. We will show that this amounts to taking H^{-1} for the star and ignoring the feature part. We have the eigenvector decomposition of each star node's energy so that it is then simply to write down the matrix in question.

The $\mathbf{x}_o$ variables of star j were the $\mathbf{x}_f$, (and $\mathbf{x}_r$), relative to the base pose node of the star, (see equation 6.13). The perturbation in these, $\delta\mathbf{x}_o$, was defined relative to the star equilibrium point. Thus $\delta\mathbf{x}_o = 0$ implies that $G_j = 0$. We need to define some notation, the Hessian with respect the $\mathbf{x}_o$ variables of the j^{th} star and the innovation of the relative coordinates of star j:

$$H_j \equiv Q^T H_{qq} Q, \quad \mathbf{q}_j \equiv \delta\mathbf{x}_o \tag{6.24}$$

The constraint for all closed loops in the graph can be written as:

$$\sum_{j \in stars} A_j \mathbf{q}_j = \mathbf{c}. \tag{6.25}$$

The A_j matrices and the $\mathbf{c}$ vector are found by linearizing the relative transformations around the loops. The A_j have only 0's in the columns for the feature states in $\mathbf{q}_i$. The sum over j includes all the star nodes. We now consider a cost function with this constraint built in by using a Lagrange multiplier, γ.

$$\sum_{j \in stars} \mathbf{q}_j^T H_j \mathbf{q}_j + \gamma(\sum_{j \in stars} A_j \mathbf{q}_j - \mathbf{c}). \tag{6.26}$$

This then gives us these equations that are easily solved:

$$\sum_{j \in stars} \begin{pmatrix} H_j & A_j^T \\ A_j & 0 \end{pmatrix} \begin{pmatrix} \mathbf{q}_j \\ \gamma \end{pmatrix} = \begin{pmatrix} \mathbf{0} \\ \mathbf{c} \end{pmatrix}. \tag{6.27}$$

$$\mathbf{q}_j = -(H_j^{-1} A_j^T)\gamma. \tag{6.28}$$

$$\gamma = -\left(\sum_{j \in stars} A_j H_j^{-1} A_j^T \right)^{-1} \mathbf{c}. \tag{6.29}$$

The feature parts of $\mathbf{q}_j$ are not needed as the features will be set during the fine tuning step. We notice that H_j^{-1} is multiplied by A_j and so the feature parts of

H_j^{-1} are not needed. This coarse calculation is very fast. The fine tuning can take a significantly long time, (10's of seconds on a 550 MHz Pentium III). How much time depends on the details of the map and the path the robot took.

The $\mathbf{q}_j$ variables are the perturbation of the the relative pose across a star node. The state that this perturbation is relative to is the star equilibrium position. We could alternatively use the relative pose before imposing the new constraint instead of the star equilibrum position. These poses have been relaxed with the full system before our approximation and therefore may give a better starting point for the fine tuning step. We found the best result when we took a weighted average between these two choices. The weight depends on how much of an adjustment is needed to close the loop. If the graph needs only a small adjustment from the current state to close then we give the current state a high weight.

Experiments

In figure 6.13 we show the same map as shown in figure 6.4 now made using the the graph reduction by star nodes [24, 89]. We formed stars up to level 7, (127 pose node reductions per star). The calculation time on average was comparable to an EKF on the same data set. The same loop closing problem as on the other maps of this data appears but now we can close the loop by the method described. The result is shown in the lower half of figure 6.13. We see that the map is substantially improved in the areas that had the most errors and became slightly worse in the upper two corners which were nearly perfect in the upper map.

At this point we have succeeded in solving this difficult data set. The combination of flexible data association, local linearizations and being able to use the topological constraints consistently were the key elements that allowed this.

This was a successful test of our loop closing ideas but here we entered the loop matching constraint in by hand. In the next section we show how this can be detected automatically. There we will look at an even harder data set.

6.7 Finding Loops Automatically

The data association problem when the robot pose error is too large to disambiguate single matches can be solved by considering multiple matches simultaneously [20, 28]. We also will require multiple matches before accepting a loop closing.

The criteria we will use is based on the energy increase from making the match being balanced against a threshold that is based on the same matching term Λ as we had in equation (2.5) plus a new term based on considering the likelihood of seeing the features matched to and not seeing the features that we fail to match to.

We prevent the normal data association from trying to match features that are close in metric distance but far in graph distance. The graph distance is here the distance between two features measured along the edges of the graph. This distance is taken along a path that does not pass through any feature nodes. The edges have

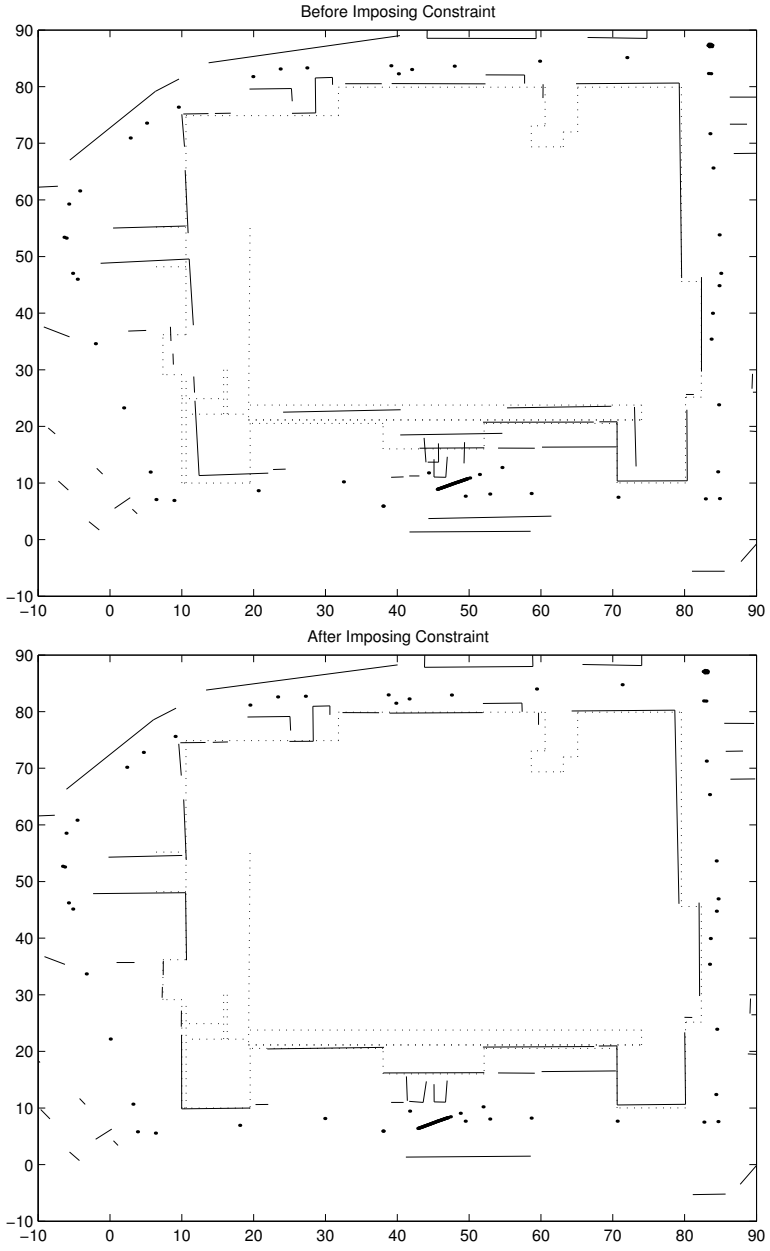

Figure 6.13: Here is the map made using the star nodes. After completing the loop we imposed the constraint and fine tuned the graph resulting in the figure on the bottom. The size of this area is about 100 by 100 meters.

a distance that approximates the strength of the edge. It is only the features close to the current pose node in graph distance that are matched to each iteration.

This then prevents the program from closing loops incorrectly by matching to features that are only weakly coupled to the current pose. In order to close these loops we look at all the features with edges to the stars of the loop set and try to find multiple matches. We call this set of features the *loop features*. If we cannot find any loop matches we move the oldest stars from the loop set to the mature nodes. We leave some small number of star nodes in the loop set to be tested again with newer stars the next time the loop closing test is done.

Before we can apply the test we need to eliminate the impossible matches from consideration. The metric information on the features is used for this with a tolerance that grows with graph distance from the loop features.

We form a list of *match features* for the i^{th} loop feature that are not attached to the tail or loop set. We then remove features from this list by applying a series of tests. The first[3] test for the j^{th} match feature is

$$({}_i\mathbf{x}_f - {}_j\mathbf{x}_f)^T B_i^T W_{ij} B_i ({}_i\mathbf{x}_f - {}_j\mathbf{x}_f) < 1 \tag{6.30}$$

Where ${}_i\mathbf{x}_f$ are the coordinates of the i^{th} loop feature, ${}_j\mathbf{x}_f$ are the coordinates of the corresponding j^{th} match feature and the B_i matrix is evaluated at the current estimate of ${}_i\mathbf{x}_f$ and W_{ij} is a weight matrix that depends on the graph distance between the two features[4]. This graph distance is approximated by the difference in the distance from the current pose node for the two features. The weight W grows as the graph distance between the features becomes smaller. We used a W that was proportional to the reciprocal of the graph distance. After this step every loop features i has a list of match features j that could be matched to it.

We now form pairs of features from those *loop features* that have at least one match feature. These pairs are tested to see if they define a transformation, (ie. for walls: is the angle between the pair of walls large enough). If it is this pair will be the base pair of a set of hypotheses. We have one hypothesis for each distinct match between the pair and the *match features*. Each of these hypotheses then defines a transformation based on the pair. All the features matched to the loop features are transformed with this transformation. Then each match between loop feature and match feature is tested by applying a threshold to $r_{ij} d_{ij}$:

$$d_{ij} = ({}_i\mathbf{x}_f - {}_j\mathbf{x}_o)^T B_i^T H_i B_i ({}_i\mathbf{x}_f - {}_j\mathbf{x}_o). \tag{6.31}$$

$$r_{ij} = \frac{m}{trH_i} + \frac{Ma_{ij}}{M + a_{ij} trH_i} \tag{6.32}$$

[3]Before testing with equation (6.30) we apply a few very broad tests that limit us to features in the general area of the loop set.

[4]We leave out some smaller modification to B_i that are needed to account for different M-Spaces of the two features. These are projection and scaling matrices to the common M-Space of the i and j features.

$$a_{ij} = \frac{trH_j}{trH_j + trH_i} \tag{6.33}$$

Here H_i, H_j are the Hessians of the energy with respect to feature i, j respectively and ${}_j\mathbf{x}_o = T({}_j\mathbf{x}_f)$ is the transformed matched feature coordinate vector. We also have two constants, M is a large constant and m a small one. This is an estimate of the increase in energy caused by matching the two features. The a_{ij} account for the reduction in energy from the relative movement of features i and j after relaxation, while M accounts for the effect of the changes to the rest of the the graph state nodes. The M estimates the coupling of the two features to their local maps. We set M to 10^6 which corresponds to mm accuracy. We see that M limits the size of H preventing an unreasonably tight test. To avoid a too loose test m is set to 100 (10cm).

The increase in energy from making the match, that the products $r_{ij}d_{ij}$ estimate, is balanced against the gain from matching the features from the Λ term of equation (6.6). Considering only these two terms we found that obvious mistaken matches were made as in figure 6.14

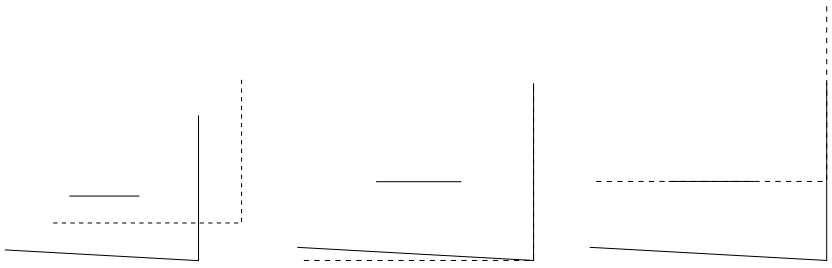

Figure 6.14: This illustrates how an obvious mistake can be made by considering the change in energy alone. The solid and dashed lines represent loop features and match features, respectively. We see that the match in the middle has a higher energy than the one on the right. But the middle match is more likely when we consider the unlikeliness of not having seen the long solid line on the return to the area.

Therefore we need to consider the likelihood of not having seen the same feature when the robot was in this area previously. We define L_h, a part of the energy of this hypothesis.

$$L_h = -\sum_{i \in matched} \log(1-(1-p_i)^{ni}) - \sum_{i \in unmatched} \log(1-p_i)^{ni} \tag{6.34}$$

where p_i is the probability of observing a feature i if we try to and ni is the number of times we try. Thus $(1-p_i)^{ni}$ is the probability of not observing it and $1-(1-p_i)^{ni}$ of observing it at least once.

The ni is the number of times we tried to 'see' the looped feature from the $match\ features$ part of the graph. The ni and pi are estimated by maintaining 2 counters for every feature. The first counter is incremented each time the feature should be seen from the current robot pose. The other counter is incremented when the feature is matched to a measurement. p_i is then the ratio of these and n_i is the maximum of the first counter over all potential matches j.

Equation (6.34) makes it harder to not match important very visible features compared with not matching hard to see features. Furthermore, it tends to match important features to important features. There are no parameters to set in deciding our thresholds. We simply count. We can now form our cost function by combining all these terms, equation (6.34) and equation (6.31) with equation (6.6)

$$Cost = L_h + \sum_{i \in matched} (r_{ij} d_{ij} - \Lambda(dim)_i) \tag{6.35}$$

Where $(dim)_i$ is the dimension of the feature. From this we derive thresholds for individual feature terms to produce the smallest cost. There must also be some minimum number of matches before we accept the loop closure. This simple formula was sufficient for the data sets we worked on.

There are of course situations in which it will fail. Situations where the features near the overlapping region are too sparse and/or do not provide sufficient constraints on the pose to close the loop. There could also be disturbances in the measurements just in these overlapping regions that caused the estimated maps to be different. Another problem is as we have already pointed out that as a result of repetitive patterns in the environment two different areas might look very similar.

Experiments

We again use our ATRV robot equipped with a SICK laser scanner and a 6-axis inertial sensor. Our automatic loop closing was easily able to find the simple loop shown earlier. We then tried a more challenging robot path around and through our building with three loops in it. The walls of the building are now partially hidden from the laser scanner by scaffolding and fencing around the building as there was renovation work being done. This lead to even more sparse feature measurements. In addition, we did not correct for the obvious and significant bias in the inertial and odometry angle estimates, (see figure 3.6). In figure 6.15 we show the first loop closing point. The loop was quite far from being closed but after the constraint is imposed the map is at least topologically correct. We can see that the star node just inside the building now has four pose nodes attached to it. It is this explicit loop in the path that is used to set up the constraint matrices for this and subsequent loop closing calculations as in equation 6.25.

In figure 6.16 we show the robot closing the second loop. Now the map is becoming fairly close to the hand made map shown by the dotted lines. When the second loop was discovered the first loop had been built into the graph by forming a star node with edges to 4 pose nodes. This first constraint is included in the

constraint equation having been found by looking for cycles in the graph. The two loop constraints are solved simultaneously. As the adjustment needed to close the new loop is much smaller than it was after discovering the first loop, a higher weight is given to the current robot path relative to the star equilibrium path when setting the coarse adjustment. This helps to speed up the fine tuning step.

In figure 6.17 we show the robot closing the third loop. Here there is a slight problem as the two map sections do not completely fit together. There was apparently some errors when passing through the outer door that cause the part of the map inside the building to not line up with the parts outside. Thus the match was made for the inside walls but some of the outside walls could not be consistently matched up. This shows that closing loops with relatively sparse features is problematic. We needed the features inside before we could be sure of the match but they were too weakly coupled to the features outside to get a match spanning both areas. Nevertheless the map is reasonable and the robot returns to its parking place in the lab room correctly as shown in figure 6.18. Here we forced a final global update to remove extra tension from the map.

One final evaluation of the automatic loop detection and correction using Graphical SLAM was made on the KTH campus. The robot took 58 minutes to travel the path. A large map containing 667 walls was made of an area 170 by 260 meters. The task of SLAM was made more difficult by the fact that the SICK laser scanner was blinded by the sun at several points and required rebooting. While this was happening the robot was being driven by hand. The result is several stretches of 2-3 meters with no laser data at all. Three loops were detected automatically. The first loop was closed in an open courtyard about 40 m square, figure 6.19. .

In figures 6.20 through 6.23 we show how 2 more loops were automatically detected and closed by the algorithm. Unfortunately the final loop from the start section to the finish section was not detected. This was because the stars needed to close the loop were still part of the formation set at the termination of the experiment. They had not yet been moved into the loop set.

In figures 6.24 through 6.27 we show a second experiment on the same data with different parameters. The differences are shown in table 6.1. The differences can be summarized by saying we were more selective in which walls to initialize, the tail was shortened and the stars somewhat larger. The maps were improved by this but the main error around the entire path, while smaller, is still present. This error might be corrected if the final loop were detected. As we said above automatic detection for that loop is not done as the region of overlap was not sufficiently mature when the program terminated. The overall better result from the second test is most likely due to using fewer more selective features. This confirms a rule of thumb in SLAM that a few good features are better than many features of mixed quality.

Both tests produced good results and comfirm that the algorithm can correctly detect and impose loop constraints. We need to point out that on the campus data the loop detection began to take significant time in those places where there were many combinations to check. Also the fine tuning around the loops was more time

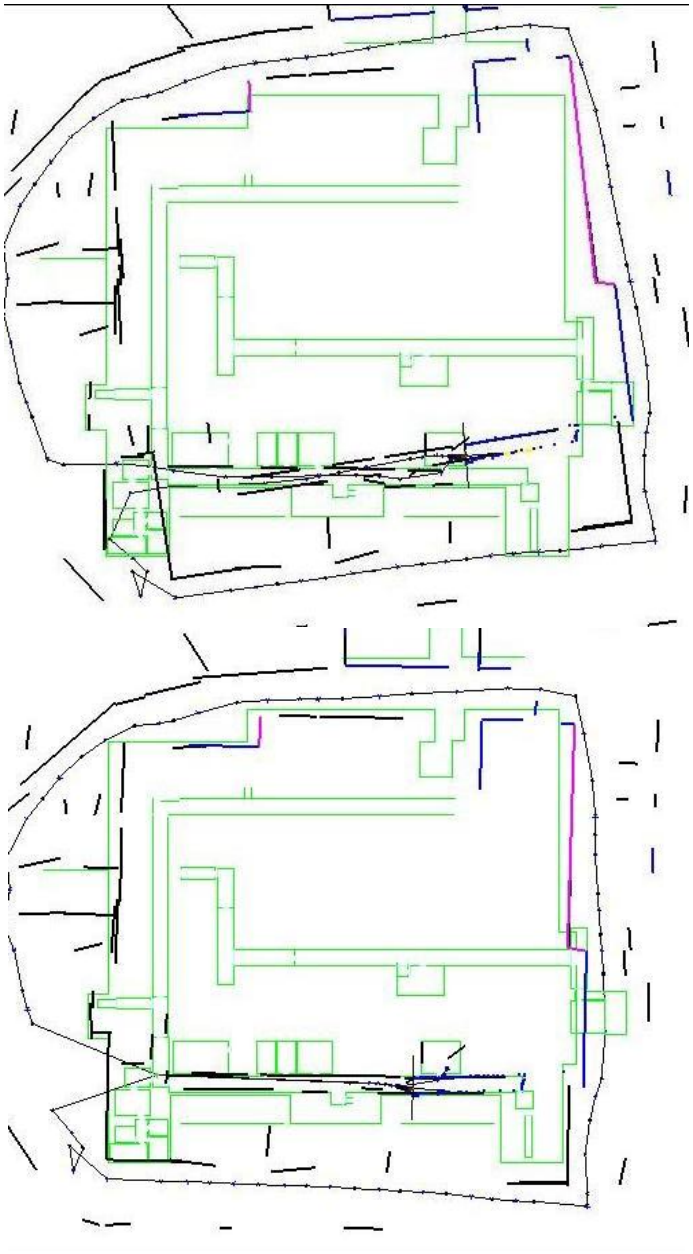

Figure 6.15: The 1^{st} loop closing, just prior, top and after, bottom. The loop closing is done on a section of the graph with enough mature features to make multiple matches possible. The match is made between features near the bottom left corner both indoor and outdoor walls.

Figure 6.16: The 2^{nd} loop closing, just prior, top and after, bottom. the outdoor section in the bottom right corner of the building is used to set the constraint.

Figure 6.17: The 3^{nd} loop closing, just prior, top and after, bottom. Here the constraint involves the same section of the map as the first constraint.

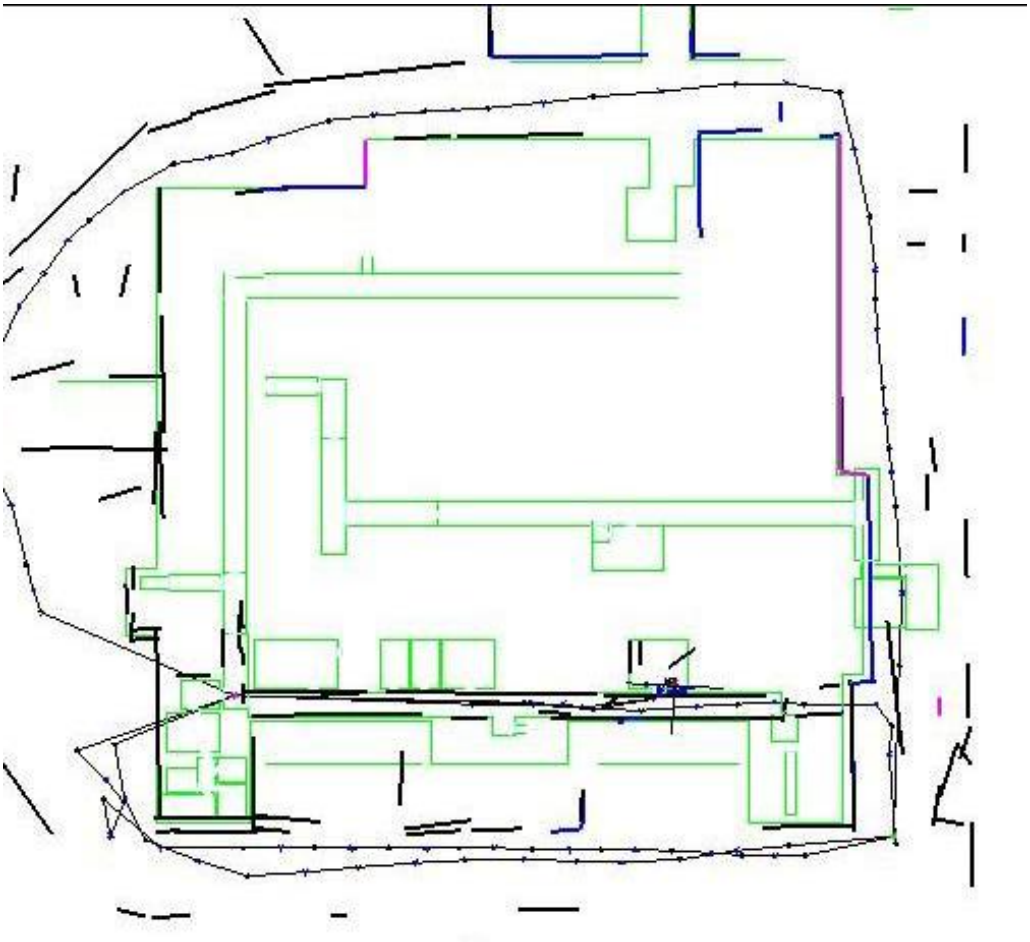

Figure 6.18: The final map with the robot parked back in its lab.

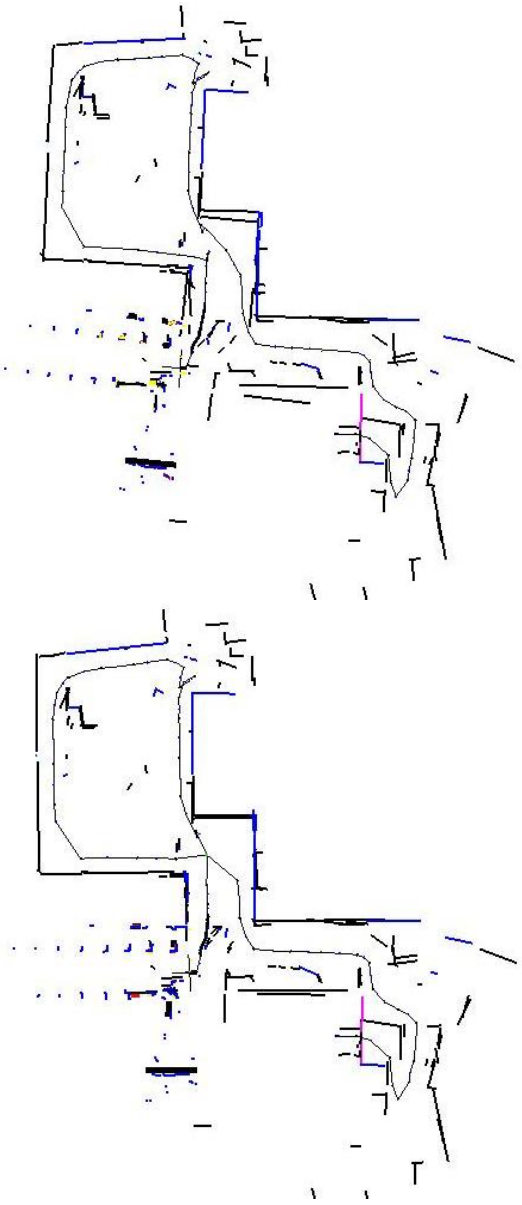

Figure 6.19: This is a test of our automatic loop closing using our ATRV robot on the KTH campus. Here the robot is leaving a courtyard and the loop around the courtyard has some errors in it. This can be seen by the duplication of walls near the entry/exit point from the courtyard in the top figure. In the bottom figure the robot has recognized this loop and added a constraint to the graph. One can see a star has been formed with four connected pose nodes. This star builds the constraint into the pose only subsystem

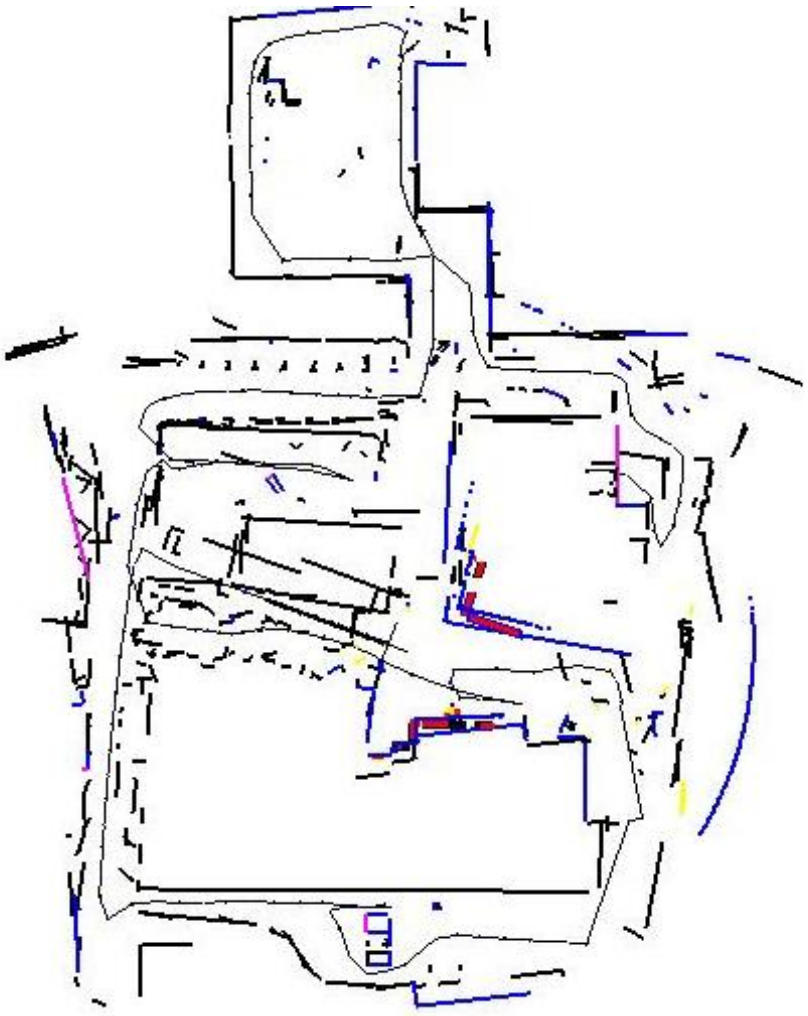

Figure 6.20: Here the robot is near the center of the map and has completed a loop around the large building in the bottom part of the figure. The loop has errors as shown by the duplication of walls.

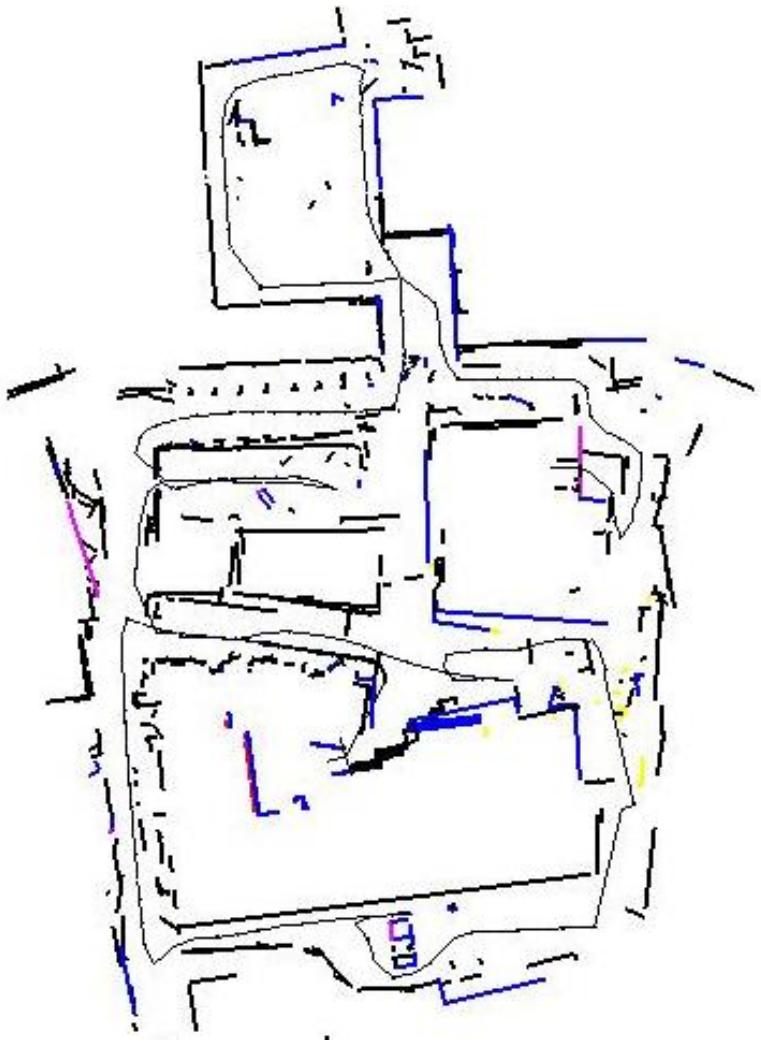

Figure 6.21: Here the loop from figure 6.20 has been automatically detected and corrected. Now there are two loop constraints on the graph. Notice that there is a large error on the lower right as the robot was not able to close the central loop as there was insufficient overlap with the previous path.

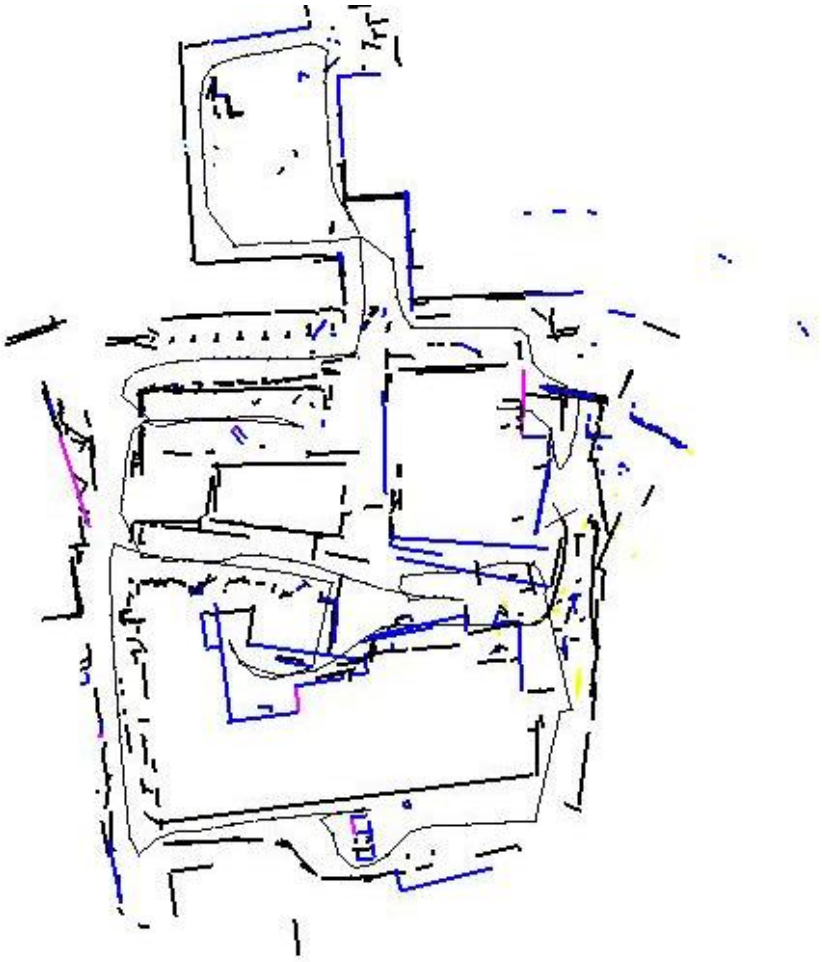

Figure 6.22: Here the robot is again back near where it started its path, to the right center. A third loop has been detected in the area between the two buildings. The loop between the start and end of the path has not been detected here. That loop has not been looked for yet as the end section of the overlap is not yet considered mature enough.

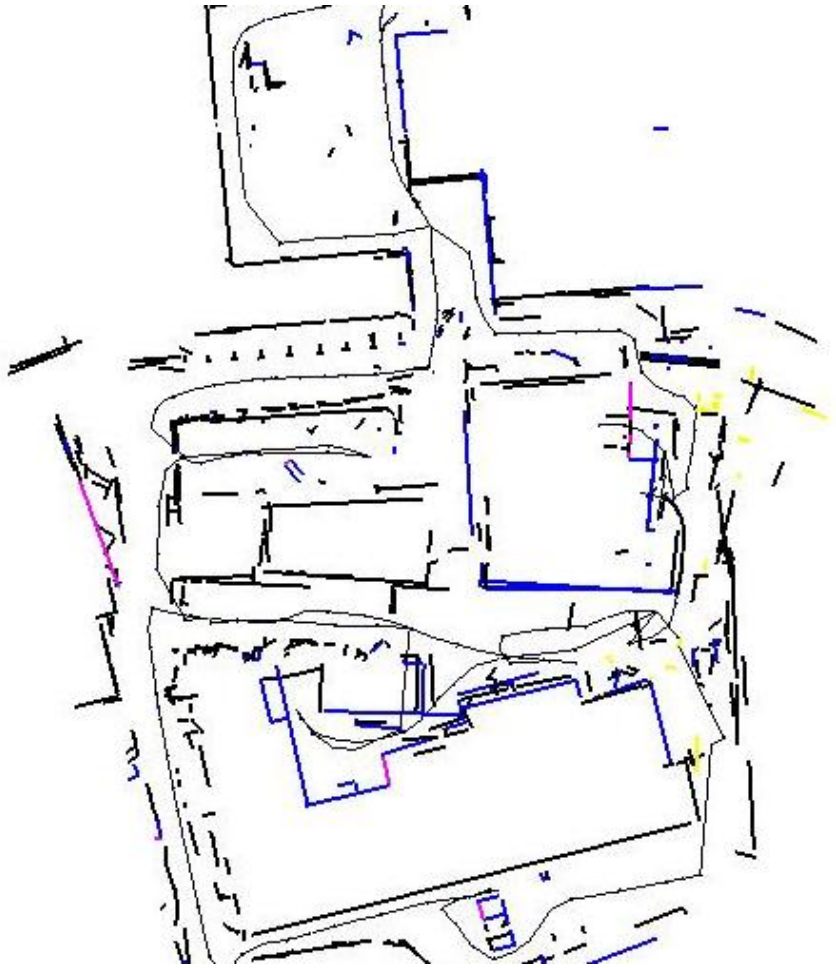

Figure 6.23: Here the loop from figure 6.22 has been automatically detected and corrected. Now there are three loop constraints on the graph.

Figure 6.24: By varying some of the parameters, (see table 6.1), we were able to run through the data again producing a somewhat better map just prior to closing the second loop.

consuming on this large data set. The loop closing would need to be implemented as a separate process running on a different computer for real time operation.

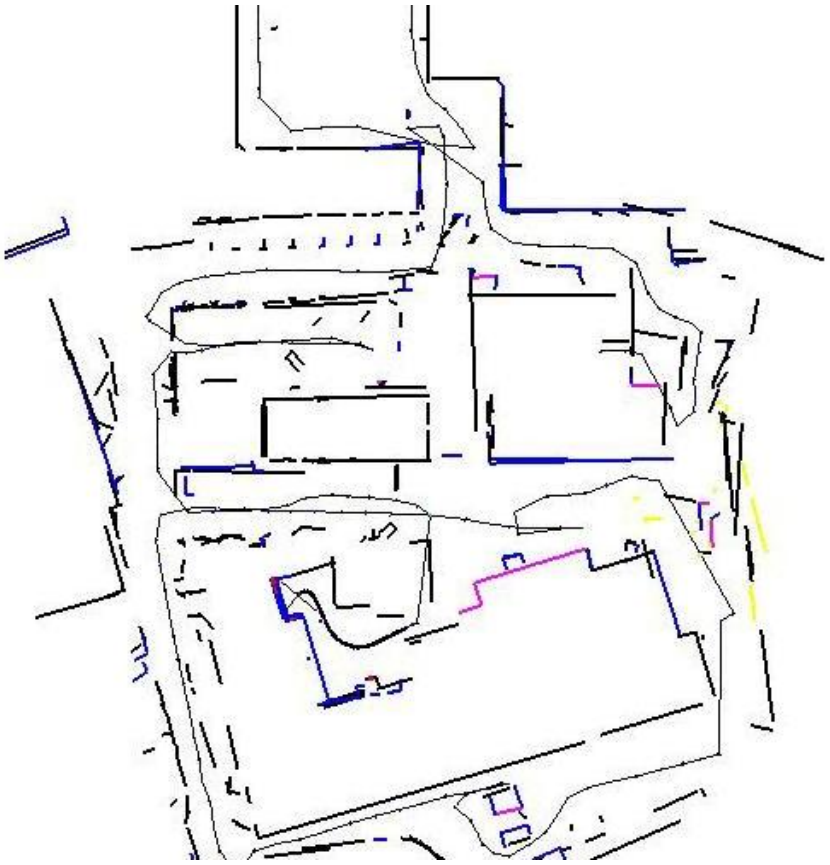

Figure 6.25: Here the loop from figure 6.24 has been automatically detected and corrected. Now there are two loop constraints on the graph.

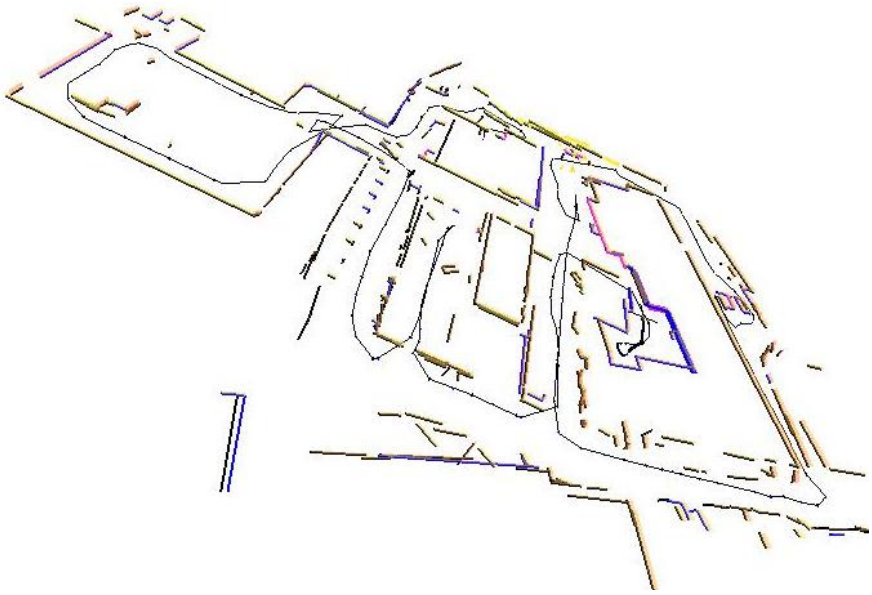

Figure 6.26: This shows the map from from figure 6.25 shown from a different angle.

Figure 6.27: This shows the final map from for test 2 (see table 6.1). This can be compared to test 1, figure 6.23.

Parameter	Test 1	Test 2
Λ	4	16
Wall Initialization: Length Threshold	.5	1.5
Wall Initialization: Number of scan points	75	100
Minimum Tail Length	100	50
Maximum Star Dimension	50	60
Minimum Number of Wall Matches for Loop Detection	4	5

Table 6.1: We show the differences between the parameters of the Graphical SLAM with automatic loop detection between the two trials on the campus data. The Λ parameter is the feature matching parameter as in equation 6.4. The wall initialization refers to the thresholds for extending the M-Space dimension of walls from 0 to 2 dimensions. There is a threshold on the length and number of scan points. The minimum tail length is the part of the graph that will be always left with the exact calculation of measurement energies. The automatic loop detection has only one parameter which is the minimum number of matches required to match two sections.

Part III

Insights

Chapter 7

Summary and Discussion

The problem of simultaneous localization and mapping (SLAM) is fundamentally a challenge in estimation, however there are several different issues that must be considered in the design of real systems.

- Consistent integration of data that are not fully observable.
- Data fusion in the presence of noisy data-associations.
- Maintenance of a consistent map in the presence of non-linearities in the measurement models.
- Methods that enable introduction of topological constraints, (such as loops) into a map and the corresponding map update.
- Handling of computational complexity to ensure real-time operation.

Summary

The first issue is one of observability. The individual measurements used to build up the map can have symmetries from dimensions of the features not measured. These symmetries should be explicitly maintained in order to maintain better consistency in the map estimates.

The M-Space feature representation has the advantage of being able to partially initialize the feature dimensions. This allows us to use the information from partially observed features. We can also share feature coordinates between two features so two walls can both share the same point for a common corner.

Several excellent methods for SLAM are based on the Extended Kalman Filter (EKF). These include the Compressed Extended Kalman Filter which we have implemented to study its strengths and weaknesses. The strengths include the relatively simplicity of the algorithm and the improved, (over the standard EKF), computational complexity. The main weaknesses are the gradual loss of consistency due to the linearizations and the related problem of imposing of topological

constraints. It also suffers from the early commitment to data associations and the need to quickly initialize the features.

We formulate the estimation problem as an energy optimization problem where the energy function reflects the structure of the information being gathered by taking measurements from robot poses along a path. This structure leads us to a graphical representation of the energy. The graph structure is given by the connections between poses along the path of the robot. This tends to form a sparse structure. The sparseness can lead to constant time updates of the map if exploited. The sparseness is destroyed if one choses to marginalize out all robot poses except the current pose. By instead allowing a few poses to remain we maintain sparseness while simplifying the graph.

By making changes to the connections of edges in our graph we can merge features or change data associations easily at any time. We can turn off certain edges in order to simplify calculations and then turn them back on to fine tune the map. We can wait to initialize feature dimensions as long as we like. When we decide to do the initialization, we still can use the very first piece of information. This kind of flexibility is a great advantage of the graphical method.

The M-Space allows us to factorize the energy of our graph into a part that is a result of the symmetries, translational and rotational invariances and a part that encapsulates the information actually measured. By maintaining this explicate factorization we eliminate many of the problems of linearization and numerical computation errors.

The consistency of the map will deteriorate as a result of linearization errors. We can minimize this problem by re-linearizing the less mature measurements each time they are used for updates and linearizing in a local frame when we finally do permanent linearizations. This local frame is the base frame of our star nodes.

The relaxation of the graph will take a time that is independent of the size of the graph. This is true since the calculation propagates only to the parts of the graph that have significant stresses. Far from the nodes being added to the graph the effect of the new nodes will be small since the stress will be small.

The graphical relaxation works very well when the robot is exploring new areas. Then the stress of new nodes being added is small and easy to relax. When going back to areas previously visited the stress will necessarily be greater and harder to relax. If the robot is continuously driven back and forth over the mapped area the graph can become intertwined and relaxation can become difficult. If there is any bias in the sensors it will worsen this effect.

When closing loops the graph can be used to form a simple system which can be used to approximate the full system. This approximation takes account of the constraint of common features for pose nodes that have a common star, but not those separated by more than one star node. This approximation is very close to the true system if such constraints are not very important. Here again having an intertwined graph can make the approximation less good. If the constraints implied by common features between stars are enough to define a transformation the stars can be merged. This then builds in the constraint in the approximate system and

will help maintain the quality of the approximation. If this is not possible poor results can occur when using the approximation.

Using the graph approximation we can impose multiple constraints on the graph with little calculation. These constraints all correspond to cycles in the graph when traversing it through pose and star nodes only. Each independent cycle gives one loop constraint.

Having solved the approximate system the exact system is then used to fine tune the map. It is this fine tuning that is time consuming. If the coarse adjustment was not good the system will end up in one of the many local minima. Also the time for the fine tuning is strongly dependent on the quality of the coarse adjustment.

The distance along the graph can be used as a measure of the correlation between state nodes. We therefore do not try to match individual features that are far in graph distance from our current pose node. This is how we avoid false matches. As we can later move these measurements to any other state nodes, we can fix this matching later by considering larger sets of matched features.

Two sections of the map close in metric distance and far in graph distance can be matched using an energy criteria. A key element of this criteria is the consideration of the likelihood of not seeing features in one section that we saw in the first section. Thus the match criteria depends on both the matched features and the unmatched features.

A nice feature of our loop closure criteria is that it depended on no new parameters. One simply has to count the number of matches to features and the number of iterations that the feature was visible and could have been observed by the sensor.

Discussion

We have shown a flexible way to make data associations that allows one to reconsider the decisions made in light of subsequent information. This goes a long way towards eliminating the problems of incorrect data associations.

The unwanted effects of linearization are generally the result of having done the linearizations around many different states. In EKF SLAM the linearizations are done around the best current state and then never recalculated. By re-linearizing all the currently relevant measurements continuously we can eliminate these effects. Furthermore we have shown that many measurements can be combined without approximating the most important non-linearities by using the star nodes. Thus we can significantly reduce the amount of calculation for the re-linearization.

The star nodes have also lead us to a strategy for handling topological constraints in SLAM. We can simplify our representation by simply ignoring the features attached to the star nodes. This becomes a reasonably good approximation to the true system and can be used to quickly adjust the position of the stars to solve the topological constraint. Then we can include the features again to fine tune the solution.

Some methods are provably consistent and convergent for linear Gaussian systems. For the general non-linear system no algorithm can be sure to avoid the

local minima of the SLAM problem. Therefore, discussions of convergence and consistency must be taken with a grain of salt.

We can say that the graph energy can be used to form correct relative inferences about states of the system. By this we mean we can say things like map 1 is three times more probable than map 2. Furthermore, since we have been very careful to minimize all approximations we are confident that we can make such statements with greater accuracy than for example the EKF, EIF or other methods that linearize in the global frame.

The issue of the local minima is both a problem and an advantage for our method. We accurately model most of these local minima. As a result we get stuck in them. Sub-map methods that do not try to impose global consistency between the features in different sub-maps have less of this structure in the model and thus have less problems. One can expect that the inferences made using these models are less accurate than using our model. On the other hand the local maps of these other models are individually more accurate locally than ours.

We found that in those cases that the EKF works well, it is very hard to beat in terms of producing nice maps easily. It is the hard situations that other methods become interesting.

The computational complexity of our method does not depend on the size of the map but does depend on the path of the robot. Paths that mostly are single passes through long sections of the environment with a few intersections and crossing points work very well. Paths with lots of backtracking and multiple paths through the same areas lead to more tension in the graph and are harder to relax.

We have evaluated our method on several different platforms.

Chapter 8

Ideas for Future Work

While the present work has addressed many of the fundamental problems in SLAM, there are still many problems that could be addressed in future work. Some of them are outlined below.

Idea 1: Star Multi-grid

In the course of the present work we found that the limitation on the size of the star nodes sometimes prevented simplifying the graph. It also made working with very dense features, (seeing more than 12 features in one 'local area'), impossible. Thus it would be nice if we could make the complexity of forming very large stars manageable in real-time. The main limiting step is the eigenvalue problem, equation 6.19.

The other benefit might be a more efficient large scale (or global) update. The current approach of relaxing locally works well sometimes but not always. Particularly when the graph often crosses back over itself and contains many medium to large scale constraints, the graph can become difficult to relax.

We might investigate eliminating the eigenvalue problem from the procedure for making a star node. In that case, one could make very large star nodes. Even as large as the whole graph. This would then be essentially an Extended Information Filter.

By saving the the energy nodes that were combined into each star we could imagine being able to go back and forth between different levels of stars. So that the star for the whole graph, (except for the tail), would be the coarsest level. We could open that star up and go down to any finer level we wanted and relax there. We would then re-linearize and merge into the coarser stars again. This might be a way to do multi-grid like relaxation that is based on the SLAM problem structure rather than forcing the SLAM problem into a multi-grid format designed for finite element problems.

Idea 2: Free Stars

The main difficulty with the graphical method is trying to relax the graph in the situation of multiple passes through the same area. As the graph becomes intertwined it becomes harder to avoid local minima. One way to avoid problems with the intertwined graphs is to let the features seen from different stars be considered to be different features permanently. Thus we have several representation of the same physical landmark. This is the approach taken in some other sub-map methods. The graph updates would always be very quick. Even loop closing will be possible in real time.

The duplication of features is a problem only when presenting the map to other agents that want to use the map. We are primarily interested in the map for localization and thus care less about this duplication.

An improvement of this separated star node idea is to let the star base nodes be nodes in a second layer graph. Where the energy nodes in that graph are set by somehow gathering the effects of all the features that really are the same between two star nodes. So that we would have a sort of special pose node only graph. One would set the pose nodes by relaxing this second graph and then the features of each star would simply move to the position given by the new pose nodes attached to their star. In this way the feature locations would be indirectly pushed toward being consistent with the matching feature locations of other stars. This is similar to the Atlas framework [20].

Idea 3: Sensor Grids

Another possibility is making SLAM graphs with more than one robot. This introduces a new type of energy node connecting two robots. With such a team SLAM approach it is possible to make maps or just stay localized with fewer or no environmental features at all. This is because the robots themselves can act as movable features for one another. With such a scheme a team of robots with relatively poor sensors could by working together to make good maps or just not get lost.

There is something natural about using the graphical method for networks of sensors whether on mobile robots or stationary. In the future we will begin to have sensor networks throughout our environment and the issues around understanding the information these networks provide is going to be an important research area.

Idea 4: Model Non-White and Non-Gaussian Noise

Another idea is to use a graph to represent time correlated errors such as those from GPS. The GPS sensor gives a triangulation fix on the robot based on the time of fight estimates of signals from orbiting satellites. These estimates are typically accurate to tens of meters. By using corrections available via a FM radio signal one can get better than 1 meter accuracy. The main problems with GPS are that the errors are not Gaussian and are in fact time correlated [90, 91].

Typically, one observes the GPS pose as continuous for a time and then a large jump of up to tens of meters followed by continuous readings. The reason is satellites coming in and out of view, reflections of the signals and multiple paths for the signals. This type of data is incompatible to most of the SLAM estimation techniques. One proposed solution is given in [91].

These types of errors could be modeled in a graph by having the GPS measurements be measurements of a GPS feature. The GPS feature has an edge to a non-Gaussian energy node that models the probability distribution of the GPS system in the absence of changes to the satellite reception. The other edges go to Gaussian energy nodes connected to the robot pose nodes where GPS readings were received. Thus the location of the GPS feature will be estimated by relaxation. When a jump occurs a new GPS feature will be initialized. The new GPS feature might be using different satellites or have some other multi-path problems than the previous GPS feature. This could be a way to deal with these wonderful but difficult GPS measurements.

Chapter 9

Conclusions

We have addressed some of the practical issues for implementing SLAM. These issues are:

- observability of the model,
- the data associations,
- the handing of non-linearities with regard to final consistency,
- the enforcing of topological constraints on the map,
- and the real-time computational issues of complexity.

The observability of the features is allowed to change in our M-Space representation. This observability is an explicit and generic part of the representation. By using the M-Space we have a formalism that allows the generic handling of both this observability issue and the centrally important coordinate transformations. We are then able to formulate algorithms that automatically account for these potentially troublesome issues.

The approach we advocate for data association is to increase the flexibility in making associations. Thus, one can both wait longer to make the association and then at any time remove the association if subsequent information contradicts the earlier decision. We do this in our graphical framework by simply moving or removing edges from a graph.

The problems encountered with linearization are mostly due to not knowing the true underlying state of the system. If that were known one could safely linearize about that state. Instead, the best one can do is linearize around the current best estimate of that state. As a result of the changes to that best estimate many methods for SLAM develop inconsistencies for larger maps.

With our graphical method the linearizations are also done around the best estimate of the state but we also recalculate the previous linearizations as needed to assure that all linearization is done around the same, best estimate of the state. We

have developed a way to combine measurements together without fully linearizing them, so that the important non-linearities are still calculated exactly.

We have shown how our graphical SLAM method can be used to both discover and solve multiple topological constraints on the map. We have done extensive tests with real data to confirm the validity of our approach. Our experience with these tests was that the search for and imposition of the topological constraints added about 10 to 20% overhead to the SLAM program. This depends on how often loops are formed. This overhead comes at the time the robot returns to a previously explored region. The criteria we use for loop detection proved to be very effective and should be generally useful for closing loops even for other SLAM approaches. It seems to be important to consider the chance of not seeing features as well as measuring the similarity of the matched features.

While the robot is exploring new areas SLAM is done quite quickly and the calculation time is independent of map size. For small maps the time and quality is comparable to our EKF implementation. For medium size maps the quality of the EKF solution is seen to be inferior to the graphical SLAM one. Most likely a result of the linearizations. The more complete use of early feature measurements, (before feature initialization) is another possible reason.

One expects that this method will be substantially faster and produce better maps than and EKF for really large data sets. This is due to the better treatment of non-linearities and the fact that the updates are all local until global constraints are imposed.

Part IV

Appendices

Appendix A

Notation

We often signify blocks of a larger matrix by giving the subscripts of a set of row/columns. So J_{of} would be a block of the Jacobian of $\mathbf{x}_o(\mathbf{x}_b, \mathbf{x}_f)$ corresponding to the feature coordinates.

Subscripts:

- $_r$ is the robot pose which can include parts to go from the center of the robot to the sensor frame,
- $_f$ is the feature parameters,
- $_p$ is the measured subspace perturbations,
- $_s$ is the state,
- $_b$ is the base or reference pose of a star node,
- $_q$ is the rest of a star node coordinates other than the base,
- $_o$ is a feature point or robot pose relative to a robot pose,
- $_d$ is dead-reckoning.

Vectors:

- $\mathbf{x}$ is a coordinate vector,
- $\mathbf{v}$ is some measurement,
- $\mathbf{U}_k$ is an eigenvectors of H_{qq},
- λ_k is a eigenvalue of H_{qq},
- $\eta(\mathbf{v}, \mathbf{x}_o)$ is the innovation function,
- u_k is $\mathbf{U}_k^T Q \mathbf{x}_o$,
- $\bar{\mathbf{x}}$ is a linearization point,

$\bar{u}_k$ is $\mathbf{U}_k^T Q \bar{\mathbf{x}}_o$.

Matrices:

H is the Hessian matrix of the Energy,

G is the gradient vector,

J is a Jacobian matrix,

B is the symmetry matrix,

$\widetilde{B}$ is the dual symmetry matrix,

C is a covariance matrix,

I is an identity matrix,

Q is BJ_{of}^{-1},

$\widetilde{Q}$ is $(J_{ob}, J_{of}\widetilde{B})$.

Other Quantities:

E is the Energy,

h_a is the data association between measurements and features,

h is $(\{\mathbf{x}_r\}, \{\mathbf{x}_f\}, h_a)$,

Λ is the match parameter,

N_f is the number of features,

n_j are the number of measurements of feature j,

R is a rotation matrix, (either 2 or 3 dimensions),

θ is the rotation angle around the z axis,

D^o is $\{J_{or}, J_{of}\widetilde{B}\}$ which projects the incremental changes to some relative base frame.

D_o the relative derivative, equal to $\frac{\partial}{\partial \mathbf{x}_o}$. This is the derivative with respect to coordinates in some relative base frame. $D_o = \frac{\partial}{\partial \mathbf{x}_q} BJ_{of}^{-1}$ where q and f include the entire state except the base pose.

Definitions

- ATLAS Framework. A sub-map graph method for SLAM [20].
- Compressed Extended Kalman Filter, CEKF. An optimization of the calculation of the EKF where high frequency local updates and low frequency global updates are done [19].

- Data Association. The match between measurements and the elements of the model that gave rise to them.
- Dead-reckoning. An estimate of the robot pose from adding up the small changes from the incremental estimates of the motion.
- Dense Information. The raw data used by the feature prior to initialization to estimate its parameters.
- Energy Node. A node that can calculate an energy and its derivatives based on the value of the state for all attached state nodes.
- Extended Kalman Filter, EKF. This is a general estimation method which when applied to the SLAM problem becomes an iterative method that relies on first linearizing the measurements and the using a Kalman filter to estimate the state.
- Expectation Maximization, EM. A two phase SLAM method that searches the space of all hypotheses.
- FastSLAM. A method of doing SLAM where possible paths are particles in a particle filter [45, 46, 43, 47, 48, 49].
- Features. Geometric landmarks designed to provide localization information on the robot pose.
- Feature Node. A state node that represents a feature.
- Formation Set. A set of star nodes that are being combined pair wise to form larger stars.
- Inconsistent SLAM. A SLAM method that produces incorrect error estimates for the state.
- Innovation. A function of the measurement and state that has expectation value of zero and often assumed to have white Gaussian noise from the measurement uncertainty.
- Loop Set. The set of star nodes that is being checked to see if it matches older regions of the graph in order to close loops.
- Loop Features. The set of feature nodes with edges to star nodes in the Loop Set.
- Mature Nodes. The Stars that are older than the loop set stars.
- Maximum A Posteriori, MAP, Hypothesis. The most probable hypothesis given the measurement data.

- Maximum Likelihood Hypothesis. The hypothesis that produces the highest probability for the measurement data.
- Measurement Space, M-Space. The measured part of the parameter space of the feature.
- P-dimension, P-dim. The dimension of the M-Space.
- Pose. The position and orientation.
- Pose Node. A state node that represents a pose the robot had while taking measurements.
- Simultaneous Localization and Mapping, SLAM. The problem of having a robot make a map of an area while using the map to localize itself.
- Sparse Extended Information Filter, SEIF. A method of doing SLAM using the inverse of the covariance matrix or Information matrix. By doing a sparsification step on this matrix constant time SLAM can be achieved.
- State Node. A node of the graph that contains the state information on some element of the model. Examples are pose nodes and feature nodes.
- Star Node. An energy node that is formed by combining multiple measurements linearized in a local frame.
- Tail. A section of the graph consisting of the pose nodes from the current robot pose node back to the formation set.

Appendix B

Robots

Here we briefly describe some of the robot used in the experiments.

Pioneer

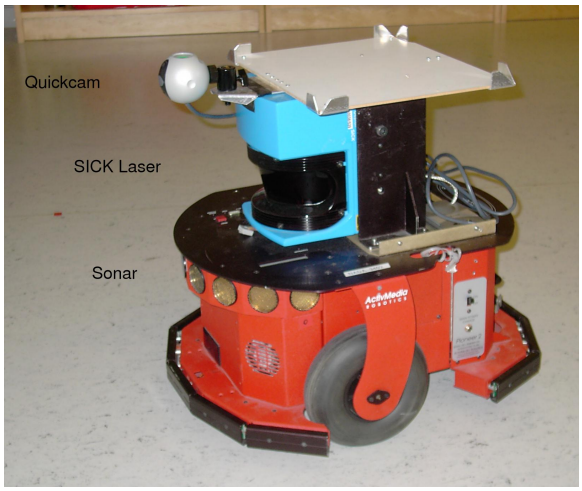

Figure B.1: Here is our ActivMedia Pioneer Robot. Goofy.

In Figure B.1 we show our ActivMedia Pioneer robot named Goofy. This robot is designed as a research robot for indoor environments. For detection of objects in the environment it is equipped with a SICK LMS Laser scanner and a set of sonar range sensors. It also has an QuickCam camera mounted pointing upwards. It has a wide-angle lens, 283 pixel focal length for 320x240 images. This camera is used

in one of our experiments. We observed significant vibration of the camera when mounted as shown.

ATRV

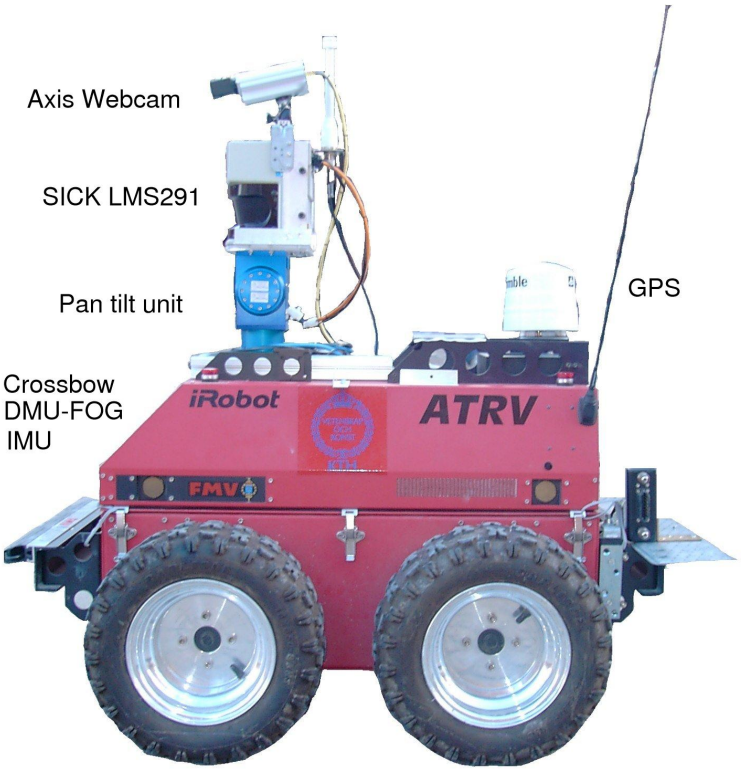

Figure B.2: Here is our ATRV Robot, Pluto.

In Figure B.2 we show our iRobot ATRV robot named Pluto. This robot is designed for outdoor use on roads and easier terrain. For detection of objects in the environment it is equipped with a SICK LMS 291 Laser scanner (range 82 m) and a set of 12 sonar range sensors, (range 8 m). It also has an Axis Network Camera. The laser and camera are mounted on a pan tilt device. The robot has an IMU unit, (Crossbow DMU-FOG) and a GPS unit to help with motion and location estimation.

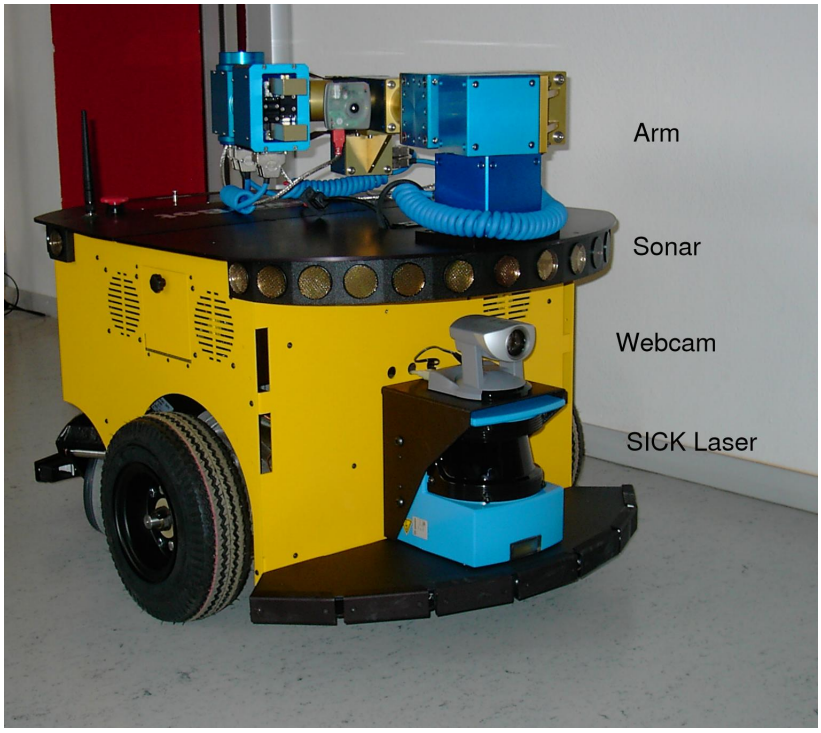

Figure B.3: Here is Dumbo. Our ActivMedia PowerBot Robot.

PowerBot

In Figure B.3 we show our ActivMedia PowerBot robot named Dumbo. This robot is designed as a research robot for indoor environments. For detection of objects in the environment it is equipped with a SICK LMS Laser scanner and a set of sonar range sensors. It also has an web camera and an arm. It is primarily used for experiments in active perception.

PeopleBot

In Figure B.4 we show our ActivMedia PeopleBot robot named Minnie. This robot is designed as a research robot for indoor environments. For detection of objects in the environment it is equipped with a SICK LMS Laser scanner and two sets of sonar range sensors. It also has an web camera mounted on a pan-tilt device. It is primarily used for experiments on robot-human interaction.

Figure B.4: Here is our ActivMedia PeopleBot Robot, Minnie.

Appendix C

Details of Hough Line Extraction

The SICK Laser scanner gives range data in a 180 degree arc in a 2D plane. These ranges are spaced either 1 or 1/2 degree apart. We need to extract straight lines form this raw data. For this we use the range weighted Hough transform. The details of this algorithm are given here. The other parts of the line extraction were explained in the section 4.1.

Figure C.1 illustrates the basic idea of the two stage range weighted Hough Transform. Each scan point fills cells in a coarse accumulator with a weight equal to its range value. The cells filled are those for which the point is consistent with the cells ρ and γ, the perpendicular distance of the line from the origin and the angle of the line normal with the x-axis. The coarse grid has cells 8 degrees per cells and a total of 32 ρ levels, a 32 x 45 grid.

The fine grid is only 8 x 8 which is a 1 degree and 1/256 of the maximum ρ is the fine resolution. The resulting set of points is tested to remove points that are far from the fit line. Finally if the remaining points pass tests on continuity, length and number of points, we remove these points from the accumulator and repeat the process until no new lines are found.

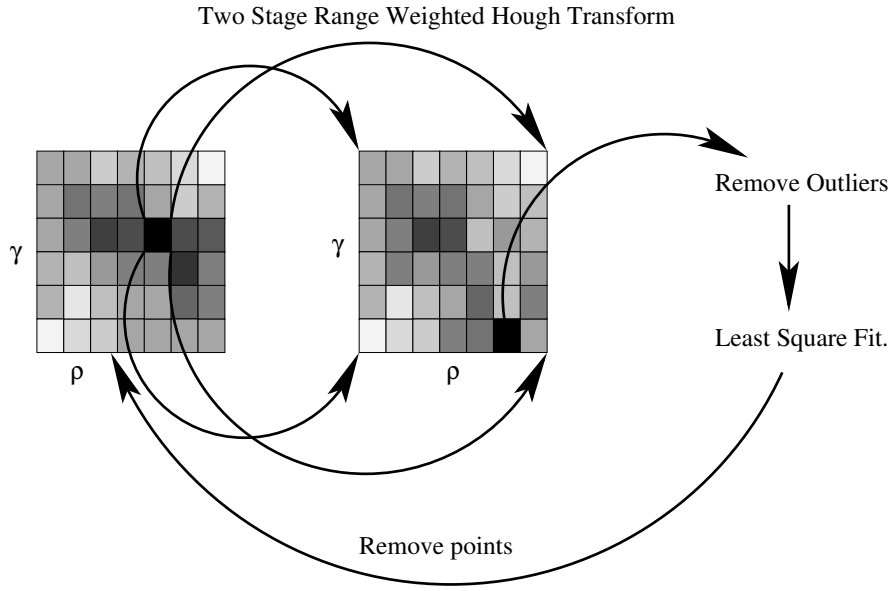

Figure C.1: The first accumulator has a coarse grid of ρ and γ. The maximum cell's contents is the used to fill a finer grid accumulator. A preliminary fit is done to a line. After removing outlining points far off the fit line, a final fit is done adjusting the points for finite beam width. The final set of scan ranges are removed from the accumulators and a new maximum is found.

Appendix D

Transformation Rules

A feature coordinate is $\mathbf{x}_f$ and a feature in the sensor frame is $\mathbf{x}_o = T(\mathbf{x}_f|\mathbf{x}_r)$. We can write the transformation rules:

$$\mathbf{x}_o^{3D} = R^{3D}(\mathbf{x}_f^{3D} - \mathbf{x}_r), \tag{D.1}$$

$$\mathbf{x}_o^{2D} = R^{2D}(\mathbf{x}_f^{2D} - \mathbf{x}_r), \tag{D.2}$$

$$\mathbf{x}_o^{S} = \mathbf{x}_f^{S}, \tag{D.3}$$

where R is the rotation matrix from the map frame to the sensor frame.

We define the 2-D points as having an x and y but extending to plus/minus infinity in the z direction. This then implies that the rotation of a 2-D point to a general frame will produce a line in the new frame that will not in general be vertical. We resolve this by taking the z of the transformed point to be the same as it was in the original frame. This may sound strange, but it is what is needed for the important case of a wall or vertical pole being observed by a 2D laser scanner. If the sensor is rotated it will still see a line on the wall but the end points might be at different heights. R^{2D} can then be written terms of the Euler angles θ, ϕ and ψ as,

$$R^{2D} = \begin{pmatrix} \frac{\cos\theta + \sin\theta\sin\phi\tan\psi}{\cos\phi} & \frac{\sin\theta - \cos\theta\sin\phi\tan\psi}{\cos\phi} \\ \frac{-\sin\theta}{\cos\psi} & \frac{\cos\theta}{\cos\psi} \end{pmatrix}$$

For the 3D rotations we get the normal rotation matrix,

$$R^{3D} = \begin{pmatrix} \cos\theta\cos\phi & \sin\theta\cos\phi & -\sin\phi \\ -\sin\theta\cos\psi + \cos\theta\sin\phi\sin\psi & \cos\theta\cos\psi + \sin\theta\sin\phi\sin\psi & \cos\phi\sin\psi \\ \sin\theta\sin\psi + \cos\theta\sin\phi\cos\psi & -\cos\theta\sin\psi + \sin\theta\sin\phi\cos\psi & \cos\phi\cos\psi \end{pmatrix}$$

The Jacobians are given by differentiating (D.1),(D.2)and (D.3).

$$J_{or} = \frac{\partial \mathbf{x}_o}{\partial \mathbf{x}_r} \tag{D.4}$$

$$J_{of} = \frac{\partial \mathbf{x}_o}{\partial \mathbf{x}_f} \tag{D.5}$$

So a robot pose might consider the variable part of the map to sensor transformation to be just $\mathbf{x}_r = (x, y, \theta)^T$. Then J_{or} would only contain columns for these. However the robot pose could also include the pitch, roll, height or sensor offset pose from the robot center point.

Appendix E

Test of Sufficiency

We need to test if a subset of features ${}_i\mathbf{x}_f$ are sufficient to fully define a transformation based on minimizing equation (6.35) to the matched j features. We start with the identity transformation with $\mathbf{x}_r = 0$ as our linearization point. ${}_j\mathbf{x}_o = T({}_j\mathbf{x}_f|\mathbf{x}_r)$. To find the correction we would need to solve:

$$\sum_{i \in matched} r_{ij} J_{or}^T B_i^T H_i B_i ({}_i\mathbf{x}_f - {}_j\mathbf{x}_o - J_{or}\delta\mathbf{x}_r +) = 0. \tag{E.1}$$

where the sum is over the subset of loop features that are matched. The r_{ij} are the weights from equation 6.32. So if the matrix multiplying $\delta\mathbf{x}_r$ is rank 3 the subset is sufficient.

Notice that here the transformation is applied to ${}_j\mathbf{x}_f$. When we actually close the loop we apply the inverse transform to ${}_i\mathbf{x}_f$. It is the need for the Hessian, H_i, that makes it easier to transform ${}_j\mathbf{x}_f$ in estimating the transform. We solve this iteratively re-linearizing each time until the adjustment to the transformation becomes small.

Appendix F

Graph Distances

We use a distance measure for an edge based on the Hessian of the edge's energy node. We consider the sub-matrix of the energy node's Hessian that correspond to the edge's state node. We calculate the eigenvalues and average the reciprocals of the non-zero eigenvalues. This is then the distance measure for the edge.

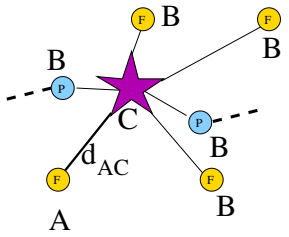

Figure F.1: Here we illustrate the calculation of the graph distance. .

$$d_{AC} = mean(\frac{1}{\lambda_i}) \tag{F.1}$$

where λ_i are the eigenvalues of H_{AA}. Where the energy of the star node is expanded about the linearization point by:

$$E_C = \bar{E} + \mathbf{G}_A \delta\mathbf{x}_A + \mathbf{G}_B \delta\mathbf{x}_B + \frac{1}{2}\delta\mathbf{x}_A^T H_{AA} \delta\mathbf{x}_A + \cdots \tag{F.2}$$

When doing the graph search we do not search paths that pass through feature nodes as the features normally are not sufficient to define the transformation. Thus the paths pass through pose nodes and star nodes along the pose-star and star-star edges. The feature distances are then found by following all the feature's edges to a pose or star node and taking the lowest distance.

The search is repeated each time a loop closing is search for, which happens at a relatively low frequency. During the time between such searches the distances are approximated locally as we add edges to the graph.

Bibliography

[1] A. Elfes, "Using occupancy grids for mobile robot perception and navigation," Computer, vol. 22, no. 6, pp. 46–57, 1989.

[2] J.-S. Gutmann and C. Schlegel, "AMOS: comparison of scan matching approaches for self-localization in indoor environments," in Proc. of the First Euromicro on Advanced Mobile Robot, 1996, pp. 61–67.

[3] D. Pagac, E. Nebot, and H. Durrant-Whyte, "An evidential approach to map-building for autonomous vehicles," IEEE Transaction on Robotics and Automation, vol. 14, no. 4, pp. 623–629, Aug. 1998.

[4] R. Kuc, "Forward model for sonar maps produced with the polaroid ranging module," in Proc. of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS2003), vol. 1, 2003, pp. 358–364.

[5] S. Thrun, "Learning occupancy grids with forward models," in Proc. of the IEEE International Conference on Intelligent Robotics and Systems (IROS01), 2001, pp. 1676–1681.

[6] F. Lu and E. Milios, "Optimal global pose estimation for consistent sensor data registration," in Proc. of the IEEE International Conference on Robotics and Automation (ICRA'95), 1995, pp. 93–100.

[7] J. Gutmann and K. Konolige, "Incremental mapping of large cyclic environments," in Proc. of the 1999 IEEE International Symposium on Computational Intelligence in Robotics and Automation, vol. 1, 1999, pp. 318–325.

[8] O. Bengtsson and A.-J. Baeveldt, "Robot localization based on scan-matching–estimating the covariance matrix for the idc algorithm," IEEE Transactions on Robotics and Automation, 2004.

[9] B. Kuipers, "Modeling spatial knowledge," Cognitive Science, vol. 2, pp. 129–153, 1978.

[10] R. A. Brooks, "Aspects of mobile robot visual map making," in Proc. of 2nd International Symposium on Robotics Reseach, Kyoto, Japan, Aug. 20-23, 1984, pp. 369–375.

[11] R. Chatila and J. P. Laumond, "Position referencing and consistent world modeling for mobile robots," in Proc. of the IEEE International Conference on Robotics and Automation (ICRA93), vol. 1, 1985, pp. 138–145.

[12] H. Choset and K. Nagatani, "Topological simultaneous localization and mapping (SLAM): Toward exact localization without explicit localization," IEEE Transactions on Robotics and Automation, vol. 17, no. 2, pp. 125–137, Apr. 2001.

[13] P. Moutarlier and R. Chatila, "Stochastic multisensory data fusion for mobile robot location and environmental modelling," in Proc. of the International Symposium on Robotics Research, 1990, pp. 85–94.

[14] J. Leonard and H. Durrant-Whyte, "Mobile robot localization by tracking geometric beacon," IEEE Transactions on Robotics and Automation, vol. 7, no. 3, pp. 376–382, June 1991.

[15] Z.Wang, S. Huang, and G. Dissanayake, "Decoupling localization and mapping in SLAM using compact relative maps," in Proc. of the IEEE International Conference on Intelligent Robots and Systems (IROS05), vol. 1, 2005.

[16] P. Newman, J. Leonard, J. Tardós, and J. Neira, "Explore and return: Experimental validation of real-time concurrent mapping and localization," in Proc. of the IEEE International Conference on Robotics and Automation (ICRA'02), Washinton, DC, USA, May 2002, pp. 1802–1809.

[17] S. Thrun, "Robotic mapping a survey," Tech. Rep. CMU-CS-02-111, Carnegie Mellon University, 2002.

[18] J. Folkesson and H. I. Christensen, "Outdoor exploration and SLAM using a compressed filter," in Proc. of the IEEE International Conference on Robotics and Automation (ICRA03), vol. 1, 2003, pp. 419–427.

[19] J. E. Guivant and E. M. Nebot, "Optimization of the simultaneous localization and map-building algorithm for real-time implementation," IEEE Transactions on Robotics and Automation, vol. 17, no. 3, pp. 242–257, June 2001.

[20] M. Bosse, P. Newman, J. Leonard, and et al., "An atlas framework for scalable mapping," in Proc. of the IEEE International Conference on Robotics and Automation (ICRA03), vol. 1, 2003, pp. 1899–1906.

[21] Castellanos, J. J. A., Neira, and J. D. Tardós, "Limits to the consistency of the EKF-based SLAM," in Intelligent Autonomous Vehicles (IAV-2004), M. I. Ribeiro and J. Santos-Victor, Eds., IFAC/EURON, IFAC/Elsevier, 2004.

[22] S. J. Julier and J. K. Uhlmann, "A counter examplee to the theory of simultaneous localization and map building," in Proc. of the IEEE International Conference on Robotics and Automation, May 2001, pp. 4238–4243.

[23] F. Lu and E. Milios, "Globally consistant range scan alignment for environmental mapping," Autonomous Robots, vol. 4, pp. 333–349, 1997.

[24] J. Folkesson and H. I. Christensen, "Graphical SLAM - a self-correcting map," in Proc. of the IEEE International Conference on Robotics and Automation (ICRA04), vol. 1, 2004.

[25] S. Thrun, D. Fox., and W. Burgard, "A probalistic approach to concurrent mapping and localization for mobile robots." Autonomous Robots, vol. 5, pp. 253–271, 1998.

[26] C. Martin and S. Thrun, "Real-time acquistion of compact volumetric 3D maps with mobile robots," in Proc. of the IEEE International Conference on Robotics and Automation (ICRA02), 2002, pp. 311–316.

[27] R. Rikoski, J. J. Leonard, and P. M. Newman, "Stocastic mapping frameworks," in Proc. of the IEEE International Conference on Robotics and Automation (ICRA02), 2002, pp. 426–432.

[28] J. Neira and J. Tardós, "Data association in stocastic mapping using the joint compatibility test," IEEE Transaction on Robotics and Automation, vol. 17, no. 6, pp. 890–897, Dec. 2001.

[29] J. Tardós, J. Neira, P. Newman, and J. Leonard, "Robust mapping and localization in indoor environments using sonar data," International Journal of Robotics Research,, vol. 4, 2002.

[30] J. Folkesson and H. I. Christensen, "Closing the loop with graphical SLAM," Submitted: IEEE Transactions on Robotics and Automation, 2005.

[31] M. E. Jefferies, W. Weng, J. T. baker, M. C. Cosgrove, and M. Mayo, "A hybrid approach to finding cycles inhybird maps," in Proc. of the Australian Conference on Robotics and Automation, vol. 1, 2003.

[32] B. Stewart, J. Ko, D. Fox, and K. Konolige, "The revisiting problem in mobnile robot map building: A hierarchical baysian approach," in Proc. of the Conference on Uncertainty in Atificial Intelligence, vol. 1, 2003.

[33] P. Buschka and A. Saffiotti, "Some notes on the use of hybrid maps for mobile robots," in Proc. of the IEEE International Conference on Intelligent Autonomous Systems, vol. 1, 2004.

[34] B. J. Kuipers, "Representing knowledge of large-scale space," MIT Artificial Intelligence Laboratory, Tech. Rep. TR-418 (revised version of Doctoral thesis May 1977, MIT Mathematical Department), July 1977.

[35] R. Smith, M. Self, and P. Cheeseman, "A stochastic map for uncertain spatial relationships," in 4th International Symposium on Robotics Research, 1987.

[36] J. A. Castellanos, J.M.Martinez, J. Neira, and J. D. Tardós, "Simultaneous map building and localization for mobile robots: A multisensor fusion approach," in Proc. of the IEEE International Conference on Robotics and Automation (ICRA98), vol. 1, 1998, pp. 1244–1250.

[37] J. A. Castellanos, J. Montiel, J. Neira, and J. D. Tardós, "The spmap: a probabilistic framework for simultaneous localization and map building," IEEE Transactions on Robotics and Automation, vol. 15, no. 5, pp. 948–952, Oct. 1999.

[38] G. Dissanayake, H. Durrant-Whyte, and T. Bailey, "A compuationally effecient solution to the simultaneous localization and map building (SLAM) problem," in Proc. of the IEEE International Conference on Robotics and Automation, Apr. 2000, pp. 1009–1014.

[39] J. E. Guivant, F. R. Masson, and E. M. Nebot, "Simultaneous localization and map building using natural features and absolute information," Robotics and Autonomous Systems, pp. 79–90, 2002.

[40] J. E. Guivant and E. M. Nebot, "Solving the computaional and memory requirements for feature-based simultaneous localization and mapping algorithms," IEEE Transactions on Robotics and Automation, vol. 19, no. 4, pp. 749–755, Aug. 2003.

[41] J. E. Guivant, E. M. Nebot, and S. Baker, "Localization and map building using laser range sensors in outdoor applications," Journal of Robotic Systems, pp. 565–583, 2000.

[42] Y. Lui and S. Thrun, "Results for outdoor-SLAM using sparse extended information filters," in Proc. of the IEEE International Conference on Robotics and Automation (ICRA03), vol. 1, 2003, pp. 1227–1233.

[43] S. Thrun, Y. Liu, D. Koller, A. Ng, Z.Ghahramani, and H. Durrant-White, "Simultaneous localization and mapping with sparse extended information filters," International Journal of Robotics Research, vol. 23, no. 8, pp. 690–717, 2004.

[44] M. A. Paskin, "Thin junction tree filters for simultaneous localization and mapping," in Proc. of the 18th Joint Conference on Artificial Intelligence (IJCAI-03), G. Gottlob and T. Walsh, Eds. San Francisco, CA: Morgan Kaufmann Publishers, 2003, pp. 1157–1164.

[45] M. Montemerlo, S. Thrun, D. Koller, and B. Wegbreit, "FastSLAM: A factored solution to the simultaneous localization and mapping problem," in Proc. of the National Conference on Artificial Intelligence (AAAI-02), Edmonton, Canada, 2002.

[46] M. Montemerlo and S. Thrun, "Simultaneous localization and mapping with unknown data association using FastSLAM," in Proc. of the IEEE International Conference on Robotics and Automation (ICRA03), vol. 1, 2003, pp. 1985–1991.

[47] J. Nieto, J. E. Guivant, and E. Nebot, "Real time data association for FastSLAM," in Proc. of the IEEE International Conference on Robotics and Automation (ICRA03), 2003.

[48] D. Hähnel, W. Burgard, D. Fox, and S. Thrun, "An efficient FastSLAM algorithm for generating maps of large-scale cyclic environments form raw laser range measurements," in Proc. of the IEEE International Conference on Intelligent Robots and Systems, vol. 1, 2003, pp. 206–211.

[49] C. Stachniss, D. Hähnel, and W. Burgard, "Exploration with active loop-clossing for FastSLAM," in Proc. of the IEEE International Conference on Intelligent Robots and Systems, vol. 1, 2004.

[50] A. Eliazar and R. Parr, "Dp-SLAM: Fast, robust simultateous localization and mapping without predetermined landmarks," in Proc. of the Joint Conference on Artificial Intelligence (IJCAI-03), 2003.

[51] R. Eustice and M. W. J. Leonard, "Sparse extended information filters: Insights into sparsification," in Proc. of the IEEE/RSJ International Conference on Intellegent Robots and Systems (IROS05), 2005.

[52] Castellanos, J. M. Devy, and J. D. Tardós, "Towards a topoligical representation of indood environments: A landmark-based approach," in IEEE Conf.on Intelligent Robots and Systems IROS99, 1999.

[53] J. Leonard and P. Newman, "Consistent, convergent and constant-time SLAM," in Proc. of the 18th Joint Conference on Artificial Intelligence (IJCAI-03). San Francisco, CA: Morgan Kaufmann Publishers, 2003.

[54] P. Newman, J. Leonard, and R. Rikoski, "Towards constant-time slam on an autonomous underwater vehicle using synthetic aperture sonar," in Proc. of the International Symposyum on Robotics Research (ISRR03), 2003.

[55] C. Estrada, J. Neira, and J. D. Tardós, "Hierarchical SLAM: real-time accurate mapping of large environments," IEEE Transactions on Robotics and Automation, 2004.

[56] M. Golfarelli, D. Maio, and S. Rizzi, "Elastic correction of dead-reckoning errors in map building," in Proc. of the IEEE International Conference on Intelligent Robots and Systems, vol. 1, 1998, pp. 905–911.

[57] T. Duckett, S. Marsland, and J. Shapiro, "Learning globally consistent maps by relaxation," in Proc. of the IEEE International Conference on Robotics and Automation (ICRA00), 2000.

[58] ——, "Fast, on-line learning of globally consistant maps," Autonomous Robots, vol. 12, pp. 287–300, 2002.

[59] U. Frese and T. Duckett, "A multigrid approach for accelerating relaxation-based SLAM," in "Proc. of the IJCAI-03 Workshop on Reasoning with Uncertainty in Robotics (avail at http://www.aass.oru.se/Agora/RUR03/"), 2003.

[60] H. J. Chang, C. G. Lee, Y.-H. Lu, and Y. C. Hu, "A computational efficient SLAM algorithm based on logarithmic-map partitioning," in Proc. of the IEEE International Conference on Intelligent Robots and Systems (IROS04), vol. 1, 2004, pp. 1041–1046.

[61] N. Tomatis, I. Nourbakhsh, and R. Siegwart, "Hybrid simultaneous localization and map building: a natural integration of topological an metric," Robotics and Autonomous Systems, pp. 3–14, 2003.

[62] J. Nieto, J. E. Guivant, and E. Nebot, "A novel hybrid map representation for denseSLAM in unstructured large environments," in Proc. of the IEEE International Conference on Robotics and Automation (ICRA03), 2004.

[63] F. Masson, J. Guivant, and E. Nebot, "Hybrid architecture for simultaneous localization and map building in large outdoor areas," in Proc. of the IEEE International Conference on Intelligent Robots and Systems(IROS02). IEEE, 2002.

[64] J. A. Castellanos, J. Neira, and J. D. Tardós, "Multisensor fusion for simultaneous localization and map building," IEEE Transactions on Robotics and Automation, vol. 17, no. 6, pp. 908–914, Dec. 2001.

[65] S. H. G. ten Hagen and B. J. Krose, "Towards global consistent pose estimation from images," in Proc. of the IEEE International Conference on Intellegent Robotics and Systems (IROS02), vol. 1, 2002, pp. 466–471.

[66] D. Burschka and G. D. Hager, "V-GPS(SLAM): Vision-based inertial system for mobile robots," in Proc. of the IEEE International Conference on Robotics and Automation (ICRA04), vol. 1, 2004, pp. 409–415.

[67] M. A. Garcia and A. Solanas, "3D simultaneous localization and model from stereo vision," in Proc. of the IEEE International Conference on Robotics and Automation, vol. 1, 2004, pp. 847–853.

[68] S. Se, D. G. Lowe, and J. Little, "Mobile robot localization and mapping with uncertainty using scale-invariant visual landmarks," International Journal of Robotics Research, vol. 21, no. 8, pp. 735–58, 2002.

[69] A. Costa, G. Kantor, and H. Choset, "Bearing-only landmark initialization with unknown data association," in Proc. of the IEEE International Conference on Robotics and Automation (ICRA04), vol. 1, 2004, pp. 1764–1770.

[70] S. Williams and I. Mahon, "Simultaneous localization and mapping on the Great Barrier Reef," in Proc. of the IEEE International Conference on Robotics and Automation (ICRA04), vol. 1, 2004, pp. 1771–1776.

[71] A. Davison, "Real-time simultaneous localization and mapping with a single camera," in Proc. of the IEEE International Conference on Computer Vison, 2003.

[72] A. Davison, Y. G. Cid, and N. Kita, "Real-time 3D SLAM with awide-angle vision," in Intelligent Autonomous Vehicles (IAV-2004), M. I. Ribeiro and J. Santos-Victor, Eds., IFAC/EURON, IFAC/Elsevier, 2004.

[73] L. Goncalves, E. D. Bernardo, D. Benson, M. Svedman, J. Ostrowski, N. Karlsson, and P. Pirjanian, "A visual front-end for simultaneous localization and mapping," in Proc. of the IEEE International Conference on Robotics and Automation (ICRA05), 2005.

[74] N. Karlsson, E. D. Bernardo, J. Ostrowski, L. Goncalves, P. Pirjanian, and M. Munich, "The vSLAM algorithm for robust localization and mapping," in Proc. of the IEEE International Conference on Robotics and Automation (ICRA05), 2005.

[75] D. Asmar, J. S. Zelek, and S. M. Abdallah, "SmartSLAM: localization and mapping accross multi-environments," in Proc. of the IEEE International Conference on Systems,Man and Cybernetics, vol. 2. IEEE, 2004, pp. 5240–5245.

[76] K. S. Chong and L. Kleeman, "Accrate odometry and error modeling for a mobile robot," in "Proc. of the IEEE International Conference on Robotics and Automation (ICRA'97)", 1997, pp. 2783–2788.

[77] M. G. Dissanayake, S. Sukkarieh, E. Nebot, H. Durrant-Whyte, and M. Corba, "The aiding of a low-cost strapdown inertial measurement unit using vehicle model constraints for land vehicle applications," IEEE Transactions on Robotics and Automation, vol. 17, no. 5, pp. 731–747, Oct. 2001.

[78] H. Bulata and M. Devy, "Incremental construction of a landmark-based and topological model of indoor environments by a mobile robot," in Proc. of the IEEE International Conference on Robotics and Automation (ICRA96), 1996.

[79] A. Martinelli, N. Tomatis, and R. Siegwart, "Open challenges in SLAM: An optimal solution based on shift and rotation invariants," in Proc. of the IEEE International Conference on Robotics and Automation (ICRA04). IEEE, 2004.

[80] H.-W. R. Madhavan, "Natural landmark-based autonomous vehicle navigation," Journal of Robotics and Autonomous Systems, vol. 1, pp. 79–95, 2004.

[81] A. Diosi and L. Kleeman, "Uncertainty of line segments extracted from static sick laser scans," in Australasian Conference on Robotics and Automation, 2003, pp. 1–6.

[82] ——, "Advanced sonar and laser range finder fusion for SLAM," in Proc. of the IEEE International Conference on Intellegent Robotics and Systems (IROS04), vol. 1, 2004, pp. 1854–1860.

[83] P. Jensfelt, "Approaches to mobile robot localization in indoor environments," Ph.D. dissertation, Signal, Sensors and Systems (S3), Royal Institute of Technology, SE-100 44 Stockholm, Sweden, http://www.cas.kth.se/~ patric/publications/phd.html, 2001.

[84] D. Sack and W. Burgard, "A comparison of methods for line extraction from range data," in Proc. ofthe IFAC Symposium on Intelligent Autonomous Vehicles (IAV2004), vol. 1, 2004.

[85] J. Leonard, R. Rikoski, paul Newman, and M. Bosse, "Mapping partially observable features from multiple uncertain vantage points," IJRR International Journal on Robotics Research, vol. 7, no. 3, pp. 943–975, Oct. 2002.

[86] J. Folkesson, P. Jensfelt, and H. I. Christensen, "Vision SLAM in the measurement subspace," in Proc. of the IEEE International Conference on Robotics and Automation (ICRA05), 2005.

[87] ——, "Graphical SLAM using vision and the measurement subspace," in Proc. of the IEEE/RSJ International Conference on Intellegent Robots and Systems (IROS05), 2005.

[88] J. W. Weingarten, G. Gruener, and R. Siegwart, "Probabilistic plane fitting in 3D and an application to robotic mapping," in Proc. of the IEEE International Conference on Robotics and Automation (ICRA'04), vol. 1, 2004, pp. 927–932.

[89] J. Folkesson and H. I. Christensen, "Robust SLAM," in Proc. ofthe IFAC Symposium on Intelligent Autonomous Vehicles (IAV2004), vol. 1, 2004.

[90] S. Sukkarieh, E. Nebot, H. Durrant-Whyte, and M. Corba, "A high integrity IMU/GPS navigation loop for autonomous land vehicle applications," IEEE Transactions on Robotics and Automation, vol. 15, no. 3, pp. 572–578, June 1999.

[91] J. E. Guivant and F. Masson, "Using absolute non-Gaussian non-white observations in Gaussian SLAM," in Proc. of the IEEE International Conference on Robotics and Automation (ICRA05), vol. 1, 2005, pp. 338–343.