2007

Vision based localization of mobile robots

Jason Mooberry

Vision Based Localization of Mobile Robots

Jason Mooberry

August 2007

Zach Butler 8/23/07

Roxanne Canosa 8/23/07

Richard Zanibbi 8/23/07

Golisano College of Computing & Information Sciences
Rochester Institute of Technology
Rochester, NY 14623

Table of Contents

Table of Figures

Abstract

Mobile robotics is an active and exciting sub-field of Computer Science. Its importance is easily witnessed in a variety of undertakings from DARPA's Grand Challenge to NASA's Mars exploration program. The field is relatively young, and still many challenges face roboticists across the board. One important area of research is localization, which concerns itself with granting a robot the ability to discover and continually update an internal representation of its position. Vision based sensor systems have been investigated [8,22,27], but to much lesser extent than other popular techniques [4,6,7,9,10]. A custom mobile platform has been constructed on top of which a monocular vision based localization system has been implemented. The rigorous gathering of empirical data across a large group of parameters germane to the problem has led to various findings about monocular vision based localization and the fitness of the custom robot platform.

The localization component is based on a probabilistic technique called Monte-Carlo Localization (MCL) that tolerates a variety of different sensors and effectors, and has further proven to be adept at localization in diverse circumstances. Both a motion model and sensor model that drive the particle filter at the algorithm's core have been carefully derived. The sensor model employs a simple correlation process that leverages color histograms and edge detection to filter robot pose estimations via the on board vision. This algorithm relies on image matching to tune position estimates based on a priori knowledge of its environment in the form of a feature library. It is believed that leveraging different computationally inexpensive features can lead to efficient and robust localization with MCL. The central goal of this thesis is to implement and arrive at such a conclusion through the gathering of empirical data.

Section 1 presents a brief introduction to mobile robot localization and robot architectures, while section 2 covers MCL itself in more depth. Section 3 elaborates on the localization strategy, modeling and implementation that forms the basis of the trials that are presented toward the end of that section. Section 4 presents a revised implementation that attempts to address shortcomings identified during localization trials. Finally in section 5, conclusions are drawn about the effectiveness of the localization implementation and a path to improved localization with monocular vision is posited.

1 Introduction

A brief treatment of some relevant work in the area of localization is given below in section 1.1. An overview of the various approaches to robotic architectures follows in section 1.2 to provide perspective on the different methodologies available to an implementation.

1.1 Localization

Localization is the science of divining a robot's position relative to some internal map. The map representation is usually dictated by the available sensing hardware the robot has at its disposal. The field of localization can be summarized by three general problem areas [10]. The first, position tracking, involves maintaining relative positional knowledge based on an initial position. This is the easiest of the three and concerns itself solely with combating the incremental build up of errors throughout a robot's travels in its environment. The second is harder and is referred to as global localization. Global localization is the problem of determining robot position with zero confidence in its initial position. That is, no a priori knowledge other than a map. The third, and most difficult of the three, is the so termed kidnapped-robot problem. Here, the robot may be moved to another location on its map without warning, despite any confidence it may have acquired since initially starting its localization task.

As one might expect, a variety of methods have been put forth in an attempt to address the various problems associated with localization. Three of the most popular are Kalman filter based localization [11], Markov Localization[5] and Monte-Carlo Localization[7]. At their core, each of these maintain probabilistic approximations of the robot's pose. In short, this approximation is formed recursively by incorporating odometry data (from a motion model) that is corrected by correlating sensor readings (from a sensor model) that are cohesive with a priori map data [7]. These models attempt to quantify the discrepancies between actual and ideal readings. The kinematics of the mobile robot, (that is, how effector activity corresponds to movement), generally are insufficient in describing actual robot motion. This is due to the sizable difference between the odometry readings and the actual distance traversed, caused by wheel slippage, varying terrain, etc. The motion model accommodates these errors by taking on a probabilistic nature, typically Gaussian. The sensor model, in a similar fashion, captures errors specific to the type of sensor that is being employed, though it is not often Gaussian.

1.1.1 Localization with Kalman Filters

The Kalman filter is frequently employed as the update step in the position tracking chore. Uncertainty in the robot's motion is modeled as a unimodal

Gaussian distribution. The central idea is to find the position on a preprogrammed map that most likely matches the sensor readings being taken at a given time [17,10]. This requires a highly accurate sensor model to prevent mismatches from occurring. However, the Gaussian belief assumption is restrictive, as the pose uncertainty is represented as a uniform distribution about a single point. In actuality, the robot is not likely to be able to correctly guess its position when presented with ambiguous sensor readings. Hence Kalman filters fail to localize when multiple possible poses must be tracked. Additionally, the robot's position must be known when the algorithm initializes (otherwise the matching cannot proceed), which has serious implications for real robots.

Multi-hypothesis Kalman Filters use multiple Gaussians (multi-modal) to represent pose uncertainty[7]. Consequently, the algorithm can track multiple possible robot positions and is capable of solving the global localization problem. Though this approach also suffers severely if the positional uncertainty cannot be modeled by uniform Gaussian distributions (which it frequently cannot be).

1.1.2 Markov Localization

The core idea of Markov localization is to use discretizations of belief (normalized histograms, essentially), such as occupancy grids [5,6]. This is in contrast to the unimodal Gaussians that are used in Kalman filters. Since the representation of the probability is only restricted by the quantization of the world space, an arbitrary number of robot positions can be tracked. Consequently the global localization problem can be solved, however, variants differ in sensor update implementations and belief representations. Though in some ways an improvement over Kalman filter based localization, Markov localization can be computationally infeasible according to how the world space is quantized. Should high positional accuracy be required and the world space be large, difficulties are certain to arise.

1.1.3 Monte Carlo Localization

Monte Carlo Localization models the probability density as a discrete distribution of weighted pose samples (in lieu of a continuous representation) and relies on a recursive Bayesian particle filter to refine those pose estimations [6,7,10]. The algorithm does so by degrading samples that are not cohesive with sensor data while reinforcing samples that are. At each iteration the particles are regenerated based on the current weightings, lower weighted particles are less likely to have a sample represent them in the next population than are those with a higher weight. Thus driving the particles to converge on the actual position of the robot over time. MCL overcomes many limitations of similar algorithms. It scales well based on the available computation power and does not require highly accurate sensors, while also not restricting the form of the probabilistic

representation. The MCL algorithm's potency has led to it being adopted widely and adapted to a variety of different sensor types.

1.1.4 Vision Based Localization

Most localization approaches discussed gather distance data from sensors in one way or another to drive localization updates. With vision as a primary sensor this data is generally obtained by way of stereoscopic imaging [20] or is augmented by other sensors such as laser range-finders [22]. Despite interesting work that calculates depth information by tracking features [18], distance information is unlikely to be available with monocular vision. The few probabilistic localization implementations that rely solely on monocular vision to sense leverage feature matching to drive position updates [8].

Particularly relevant to the problem at hand is work done with integrating MCL and an invariant feature histogram matching algorithm [8] (an invariant feature is one that is recognizable despite the target feature having undergone various transformations, such as rotation, scaling and zoom). This algorithm forms the core of the MCL update step, matching database images to vision data through histograms of invariant features. These histograms then in turn drive the pose corrections by way of a normalized intersection operator. The particle update step in MCL uses the invariant feature matching steps described above to compare a sampled and processed image with database images. A visibility region is applied to reduce the complexity of the matching task. Given the particle's position on an auxiliary map, a region growing algorithm is applied to determine which pictures in the database could provide reasonable matches. Images taken outside this area, or with the incorrect orientation are disallowed in the comparison.

1.2 Architecture

Four main architectures [16] have historically been employed in building robotic systems: deliberative, reactive, behavioral and hybrid. These are briefly discussed in the following sections.

1.2.1 Deliberative

Deliberative architectures generally follow a sense-plan-act methodology. This usually requires that a complex internal representation of the environment be maintained. As new sensor information arrives, it is integrated into this representation. Decisions to act are then made based upon the updated world model, after any amount of deliberation. In its pure form, deliberative planning prevents a robotic system from being robust to changes in its environment. Clearly, attempting to determine a plan that works for all eventualities in a dynamic world is problematic. Additionally, the bottleneck generated by

integrating sensory information into a world model coupled with the decision making process prohibits rapid responses.

1.2.2 Reactive

Reactive architectures are the antithesis of the older, deliberative paradigm. The central argument is that the world is its own best model, and that intelligent behavior arises out of the dynamical relationship between a robot and that world. The intent is to purge the robotic system of any internal representations and rely entirely on low level responses to build functionality. The system is less likely to be deceived by a priori knowledge that is not cohesive with the sensed environment. Though this architecture can be applied to build a responsive system, it is difficult to infuse macro level behavior.

1.2.3 Behavioral

This approach dictates that everything in the system be decomposed at the behavior level, instead of at the functional level [1, 12]. That is, reasoning, path planning, perception, etc... are the responsibility of each behavior in the system to implement (or not) as they see fit. Contrast this with having a single global module that performs each of the aforementioned tasks. This grants a large degree of robustness to the architecture, as behaviors are generally not dependent on each other. The problem with this approach is in engineering high level behavior, as the system manifests an emergent quality that can be altogether different from that which is intended.

1.2.4 Hybrid

Hybrid architectures [2] attempt to combine the best of the reactive and deliberative paradigms to make provisions for high level planning and simultaneously maintaining responsiveness in dynamic environments. Systems employing this architecture rely on varying degrees each of reactive and deliberative techniques. Hybrid systems are the dominant approach for most current research efforts.

2 Monte Carlo Localization

Monte Carlo Localization (MCL) is a particularly efficient, scalable and flexible localization solution. It has proven itself to be a simpler and more robust technique than Kalman filter based localization or Markov localization [7, 10].

The generic update function for MCL is given by the following function.

$$Bel(x) = p(o_t|x_t) \int p(x_t|x_{t-1}, a_{t-1}) Bel(x_{t-1}) dx_{t-1}$$

Where *Bel* represents the belief in a given pose, *o* the sensor readings/observations, *x* the pose and *a* the motion or action of the robot. Refer to [7] for an in depth explanation and derivation. The term $p(x_t|x_{t-1}, a_{t-1})$ describes the probability that the pose is correct given the motion history and is frequently referred to as the motion model. The term $p(o_t|x_t)$ describes the probability that the current sensor reading was taken from a given pose, which is termed the sensor model. This function, given the previous belief, the previous position, the action undertaken, and the sensor readings, generates the confidence (or *belief*) that *x* is the correct pose of the robot. This update function drives the convergence of the posterior pose estimates on the actual robot position. In MCL the probability density function (PDF) is represented by a mass of particles, which the update function is applied to recursively. Note the initial position is typically modeled by a uniform distribution over all possible poses. A high level diagram of the iterative process is shown in Figure 1.

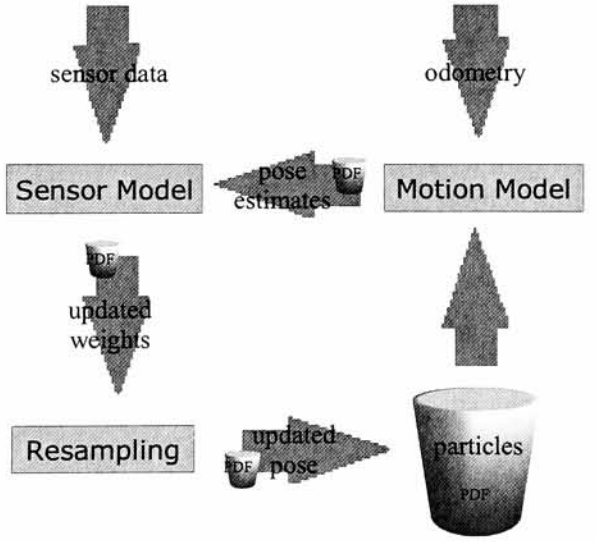


Figure 1: MCL operation

2.1 Motion Model

Recall the term $p(x_t|x_{t-1}, a_{t-1})$ describes the probability of the robot's position, given previous position and the action undertaken. To accurately portray this a model must be developed for the robot that can suitably describe the motions of the platform that occur as a result of any willful action. A thoughtful and correct application of forward kinematics is not in and of itself sufficient to represent this motion, as real world robots are acted upon by all manner of external forces that can be impossible to model. Robots are often subject to imperfections in their own constituent components as well. Consequently an important part of this model is the approximation of errors. Typically the fastest way to arrive at such an estimation is to study the robot's perceived motion in a variety of environments. Subsequent coaxing of the 'error picture' allows the model created through kinematics to be revised to generate position estimates. The robot used for testing is a four wheeled, skid steered rover whose macro level motions are composed of straight line translations and rotations about a single point. Data was gathered for both types of movement and is also presented in the following sections.

2.2 Sensor Model

The sensor model facilitates the distillation of accurate position information from the available position data, and is represented by the $p(o_t|x_t)$ term in the belief equation. Many previous applications of MCL leverage sensors that provide environment feature data and distance information implicitly [4, 5, 6, 8]. This simplifies both the matching problem and the construction of a map which describes the environment the robot will perceive. However these sensors are frequently expensive and, with few exceptions, are only concerned with data from two dimensions. Vision based sensors on the other hand yield an incredible amount of raw data with which positional information may be derived, though this richness of information comes at a cost. Adapting MCL for use in vision based localization presents several challenges that must be overcome. A significant amount of processing must be performed on data yielded by the vision sensor to derive both a map and a sensor model. Additionally a suitable frame of reference must be settled on, which is typically tied intimately to the sensor model. Vision sensors are also more susceptible to subtle changes in the environment, such as lighting and rotation or scaling of the reference with respect to the robot. Further attention is given to these issues in sections 3 and 4.

The MCL algorithm scales nicely as the computational intensity associated is a direct function of the number of particles being maintained. A critical mass of particles is typically required for the algorithm to solve the overall localization problem. As such, it is necessary to fine-tune the particle count based on empirical results obtained from the custom robot platform.

3 Vision Based MCL

For all of the reasons stated above in the localization survey, MCL is the preferred technique to arrive at a successful localization implementation. Particularly benefits relevant to this effort are the scalability, neutral stance with respect to sensing hardware and general robustness in the face of errors. MCL has been applied with monocular vision as a primary sensor in only a handful of cases [8,22,27]. The documented applications that exist have relied on a single feature matching technique to correlate vision input to a database. The final goal of this research will be to implement and maximize the efficiency of MCL on the low cost robot platform through the use of multiple computationally inexpensive invariant features and present the resulting empirical evidence. This section describes the initial development of such a vision-based MCL implementation.

3.1 Research Platform

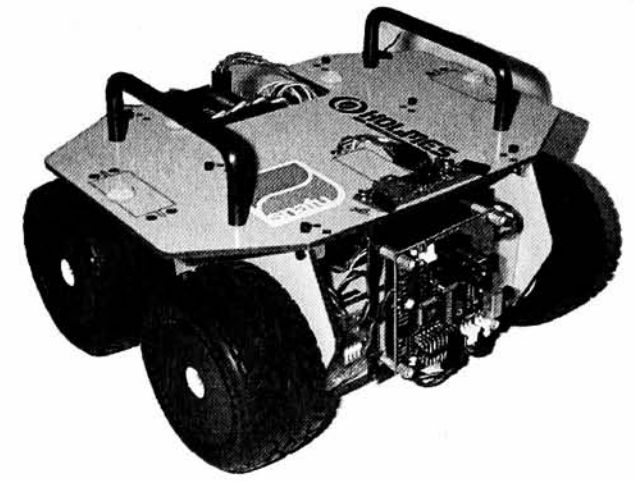


Figure 2: Maurice the robot

All experimentation was conducted on Maurice the robot (Figure 2), a custom platform built specifically for this research. Briefly, Maurice is a skid-steered, four wheel drive rover with built-in odometry and infrared proximity detectors running the BusyBox distribution of the Linux operating system. The primary sensor is a CCD camera with on-board video processing. Testing and experimentation are facilitated by CompactFlash storage and Ethernet connectivity. Appendices A and B contain more information about the hardware and software development, respectively.

3.2 Motion Estimation

The motion model that is employed is relatively simple. The shaft encoders are the only source of odometric information available on the robot, consequently the model only need be concerned with these inputs. Note the custom robot platform is best suited to navigating about and delivering accurate odometry in an environment that consists of a single plane. Consequently the Cartesian coordinate system has been selected to represent robot position.

3.2.1 Environment

The robot will operate in an arena roughly 220 square feet in area. A graphical depiction of this area is shown in Figure 3, with obstacles clearly visible.

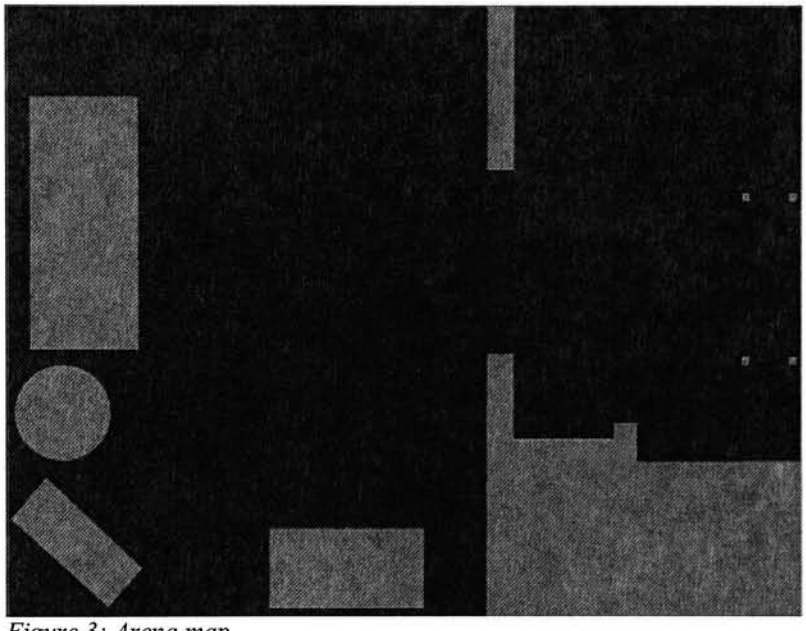


Figure 3: Arena map

~220 sq ft (17' x 13')

3.2.2 Rotation Error Modeling

Through observations of the robot motions in general, it was clear that a substantial difference exists between observed motion and that indicated by the on board odometry. Witnessed error in the platform's motions are attributable to the cumulative effects from a variety of sources. Some likely sources include wheel slippage, off-axis motor encoders and noise on quadrature decoder interrupt lines.

The rotational errors observed are presented in Figure 4. Results have been grouped for left and right turns in the same plot, which correspond to to the left and right clusters respectively in the graph.

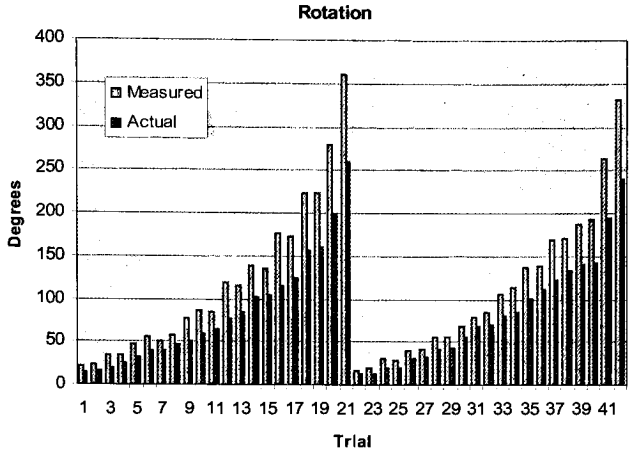


Figure 4: Degrees rotated, as measured by the robot and actual

While error data is consistent for both left and right rotations, it is clear that a significant error component is in general present in rotational motion. Further analysis of the trial data is shown in Figure 5, where the apparent proportionality of the error value to the amount of rotational motion is confirmed.

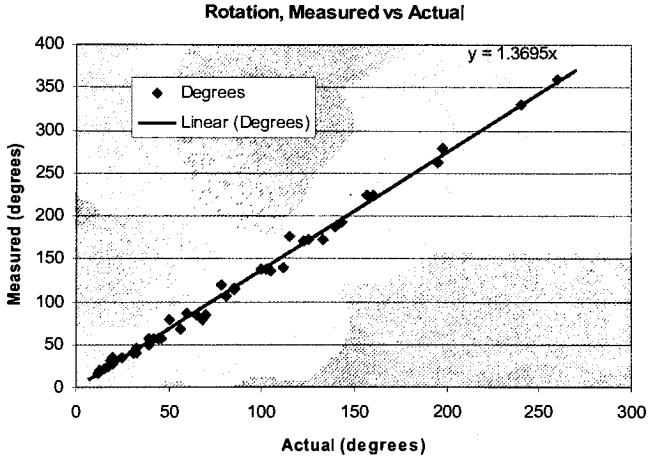


Figure 5: Error as a function of actual rotation

It is presumed that the error constant, C yielded by this analysis, is largely attributable to wheel slippage, an unavoidable problem in skid steered vehicles. The slope of the sample data yields the constant value 1.3695, and so the actual correction constant is the inverse, 0.73. Incorporation of this value back into the sample data produces the corrected measurements, which are shown in Figure 6.

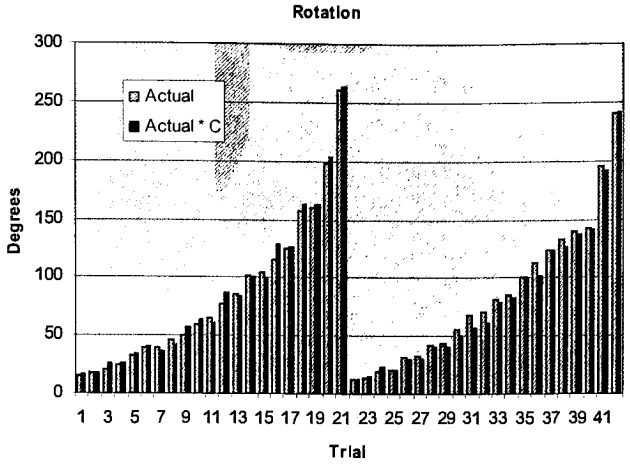


Figure 6: Application of skid-steer correction constant

The strong errors observed are largely eliminated after compensation for the linear bias, however a non-trivial error component remains as illustrated below in Figure 7, with the outliers indicating errors of almost 15°. Capturing this noise is critical to the generation of reasonable guesses during particle filter operation.

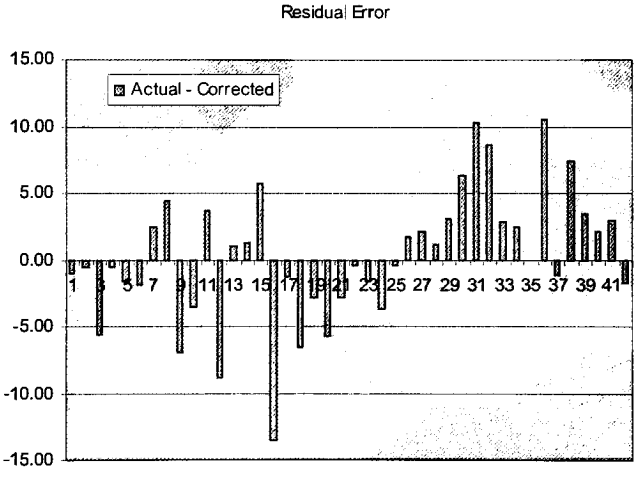


Figure 7: Error remaining after incorporation of skid-steer constant

A histogram of this residual component reveals a Gaussian distribution (plotted in Figure 8) with a mean very close to zero and a standard deviation of about 5°.

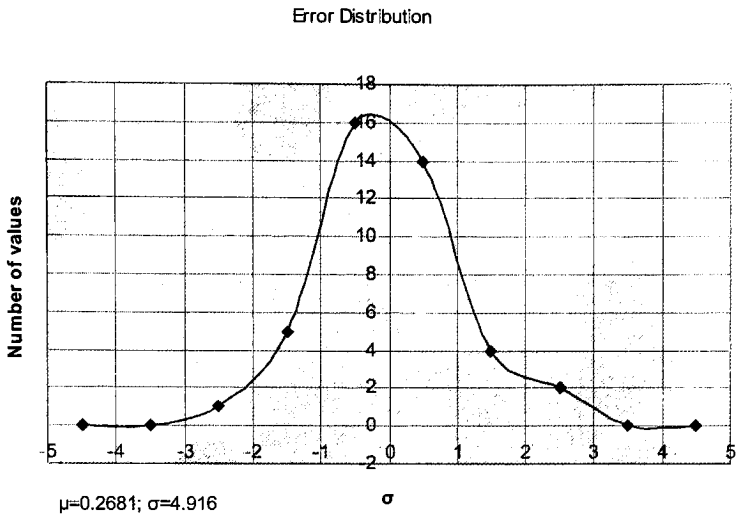


Figure 8: Histogram of residual error

The rotational position for this platform in the motion model is consequently computed as $\theta_t = \theta_{t-1} + \theta' \cdot C + randN(0.2681, 4.916)$ where θ' is the perceived rotation, *randN(mean, stdev)* yields a random number from the normal distribution defined by its arguments and C is the rotational correction constant determined previously.

3.2.3 Translation Error Modeling

Translations are constrained by the wheeled platform to those that take the robot forward or back along its current trajectory. These straight line movements are generally more well behaved than their rotational counterparts as is evidenced by the data shown below in Figure 9. This data has been capture for both types of flooring found in the arena where tests will be conducted. For convenience this data has been consolidated into a single plot. The left cluster illustrates trials conducted on carpet, the right, wood.

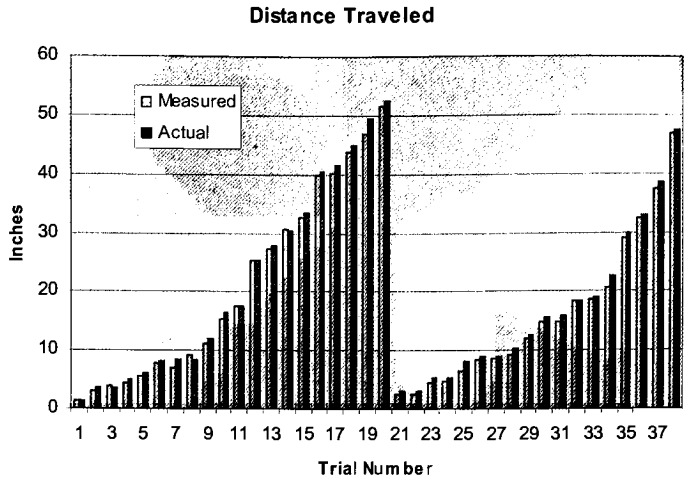


Figure 9: Straight line motion errors

Errors based on flooring type are negligible with respect to straight line motion, but significant when heading is taken into account (shown in Figure 10). These results led to the discovery of a slightly lethargic motor on the front left wheel of the robot, whose influence is greatly exaggerated on carpet. This is attributed to the higher friction constant which prevents it from skidding to keep up with the other three motors. The net effect is a significant negative heading error whenever the robot is on this surface.

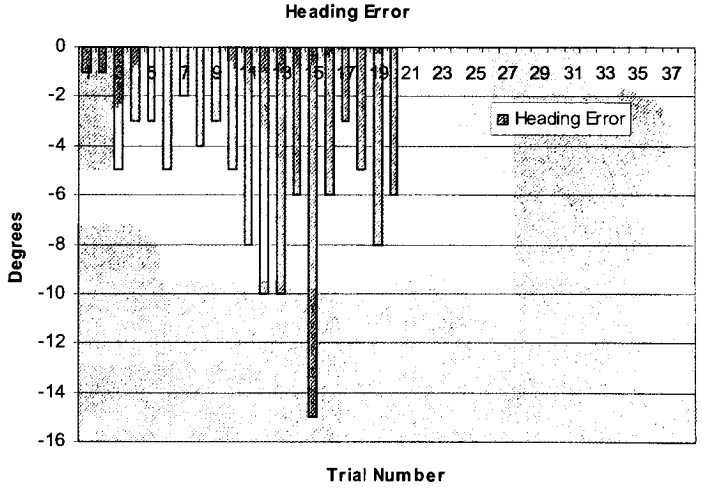


Figure 10: Heading error resulting from straight line movement

Notice that the performance of the MCL algorithm should not suffer due to this extra noise in position estimation. The algorithm itself thrives on limited amounts of noise that allow it to explore all poses in the search space, as long as samples can be generated that reflect the observed noise distributions. For simple turns, which are effected by the motors turning in opposite directions, the heading error is much less pronounced and can be computed with only the encoder error factored in.

The actual motion model employed is in some ways crude. The encoder advances for a given movement (forward, reverse, right, left) are tracked for each motor. At the completion of a movement, the error is then integrated into each reading, yielding a guess as to the actual position of each motor. This is a scientific approach (noting that the encoder positions can be expected to exhibit normally distributed error), however estimating the heading error in straight motions is decidedly not so. Since slight errors in the encoders may occur at any time during the robot's movement, vastly different effects can be observed on its resulting orientation. It would be a difficult and time consuming endeavor to generate guesses about orientation errors based on real-time data. Instead, a rotational error distribution is experimentally determined and applied to yield heading errors after each respective motion is completed.

While the encoder readings are fairly accurate, displacement errors are still present and so need to be accounted for to bring the motion model in line with the actual platform behavior. These errors are modeled as gaussian based on the interpolation of data from the various trials, shown in Figure 11 below.

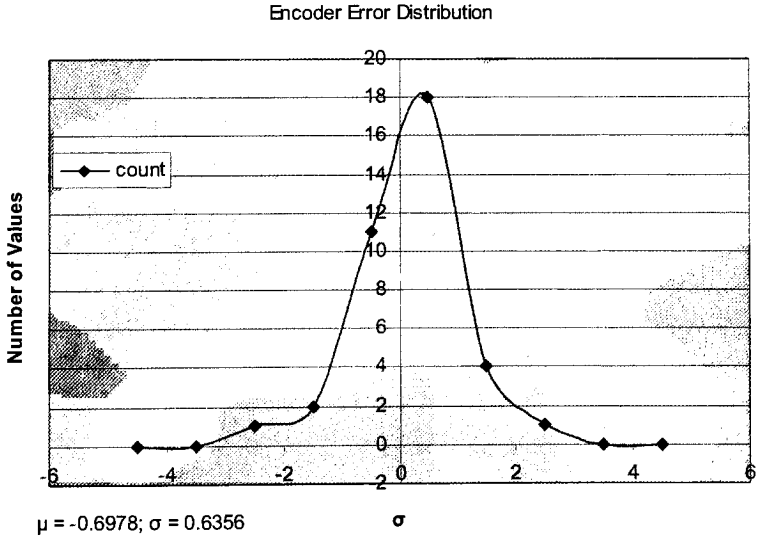


Figure 11: Error distribution

It is worthwhile to note that the error sampling based on these models is only successful because it is performed many times (once for each particle in the density representation). Hence it approximates the actual probability density error function.

3.2.4 Application of Motion Model

To reiterate, implementation of the error model on the robot platform grants improved position estimation and a means to generate sample robot positions for use in the particle filter. This coupled with a fairly naïve kinematics application completes the motion model. A sample run of the motion model alone applied to the generic position tracking task is shown below in Figure 12. In this sample the robot traversed a total distance of 9' 8". A mass of 500 particles is used to estimate the probability density. The robot begins in the upper left corner and moves incrementally along an arbitrary path toward the bottom right. At each increment the particle population is drawn.

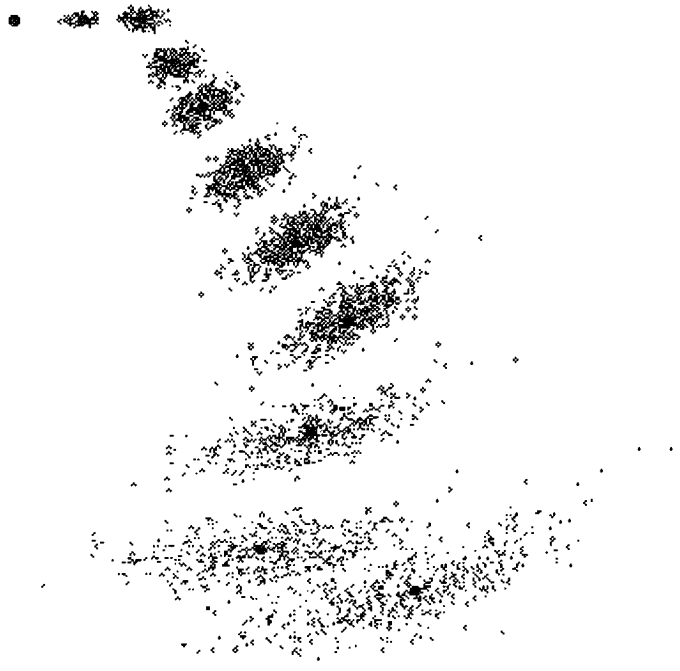


Figure 12: Motion model applied to generic position tracking task

Actual robot position shown as solid red circle

Clearly, errors compound rapidly when only the motion model is applied. This helps illustrate the critical importance of reining in position estimates by sensor feedback.

3.3 Sensor Model

Localization cannot be performed meaningfully without a frame of reference. This frame of reference, or map, takes different shapes depending on the available sensing equipment and sensor model employed. Since the sensor model employed here is based on various features extracted from color images, it is appropriate to build a single common image library from whence all feature information can later be extracted. While it is possible that each feature will have a preference for the number, quality, orientation or resolution of these source images, keeping this initial population constant should facilitate a more consistent performance comparison. A library consisting of 624 images was constructed by positioning the robot at 18" intervals (both x and y displacements) across the test arena, eight orientations each (45° increments). Features are extracted from the vision sensor and correlated to a feature database generated from this library. This operation yields match probabilities which drive the sensor update computation by revising the particle weights in the PDF representation.

In other vision based implementations [8,22,27] occupancy grids or similar mapping strategies are used to track physical location in addition to the map supplied by the feature reference. This is in a sense redundant, as the image library already retains positional information for the environment, presuming coverage is uniform. The work described here is concerned only with the ground truth provided by the feature library.

3.3.1 Naïve Correlation with Color Histograms

Color histograms are a suitable metric for differentiating pose estimates due to their rotational and translational invariance. They are also easy to generate and take comparatively little space to store. Another convenient property is the ease with which a human observer can gauge the similarity of a histogram pair by inspecting a plot of the bins. Consequently a sensor model based on color histograms can be simpler than other methods to debug.

Much like the motion model, the sensor model must be built to accurately reflect how the environment in which the robot localizes is perceived by the camera. The notion of comparing color histograms themselves is in of itself simple, however the peculiarities again of the platform, the environment and how the two interact must be well understood to permit the extraction of meaningful values to drive the particle update. Noise that exists in the system must be modeled to maximize the likelihood that histogram match quality reflects the probability that a sample image corresponds to a given position in the map. The map was implemented as a population of color histograms generated from the base image set. A variety of readings were taken from the camera with the lens covered to establish a noise floor. This was then incorporated into the sensor model to minimize the impact of noise on the histogram comparison. One of the

sample images used to generate the noise floor and the resulting histogram are shown below in Figures 13 and 14, respectively.

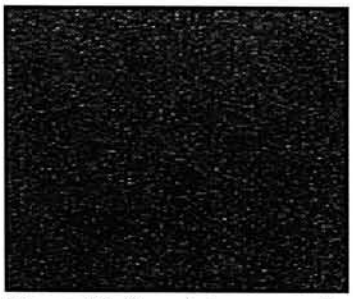


Figure 13: Sample image used to calculate noise bias

Pixel value intensity enhanced to illustrate variation

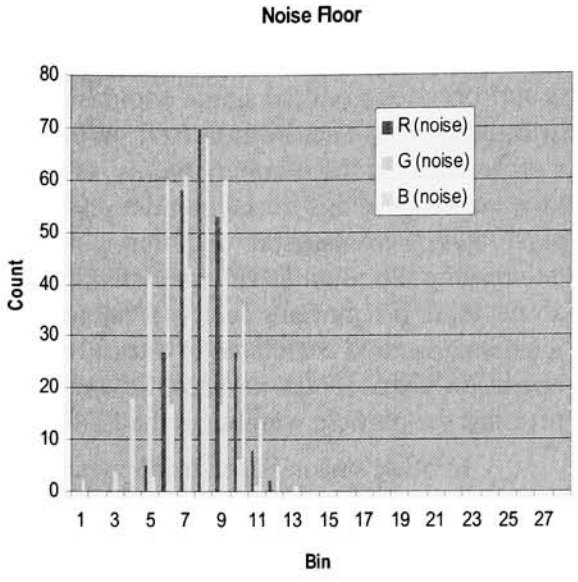


Figure 14: Color histogram noise floor

To maximize the relevance of the naïve image compare for these trials, it is necessary to understand how the environment appears in the color histogram space, particularly as perceived by the CMUCam. Figure 15 shows the maximum counts for each bin that occur across the entire color histogram library. Clearly the environment the robot is operating in induces low pixel values much more often than high. While well lit, the arena is frequently perceived as dark in areas that are carpeted by the camera, which is likely attributable to the fact that auto white-balancing is disabled. Understanding this allows a normalization constant to be derived for the naïve comparison that yields more actionable match data in the context of the base image set. Note the incandescent lighting in the arena has a strong red component, which is also evident in Figure 15. The sensitivity to lighting hue and intensity is one of the weaknesses of color histogramming as a means for comparing position.

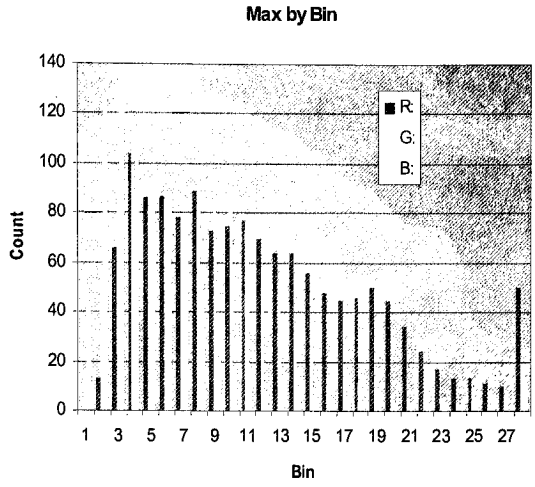


Figure 15: Maximum values by bin, across entire image library

Tailoring the histogram comparison in the manner described above limits its use generically. However bypassing this step would come at the cost of reduced sensitivity and more frequent errors in image matching. During localization, this would manifest as a tendency to lose good matches during the resampling step.

A simple differencing measure, the L_1-metric is suggested in [30] for histogram comparison. A heuristic adaptation of this measure, termed "naïve correlation" is performed to drive the update step of the particle filter as shown here, where N is the particle count, s the sample, r the reference image, n the noise floor and m the bin's relevance measure.

$$quality = \sum_{i=1}^{N} \sum_{b=1}^{bins} max((|s_{b,i} - r_{b,i}| - n_{b,i}), 0) / m_{b,i}$$

That is, for each bin in the sample, produce the difference between the corresponding bin of the reference, remove the noise bias and scale by perceived relevance.

Match quality for a sample image (Figure 16) is plotted against the entire image library in Figure 17. Note the step characteristic, which is attributable to the sizable differences in lighting between the two rooms in the arena. Additionally, a pronounced clustering of match qualities is evident between 0.3 and 0.7. This illustrates one of the difficulties in using color histograms solely to filter position estimates. The inability to disambiguate different positions strongly may result in positional uncertainty. Ideally sensor readings should correlate with a minimum of positions.

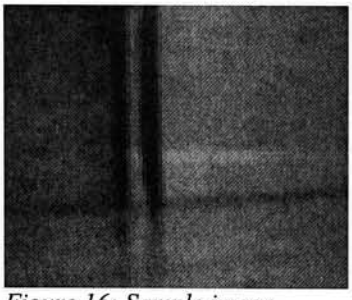


Figure 16: Sample image

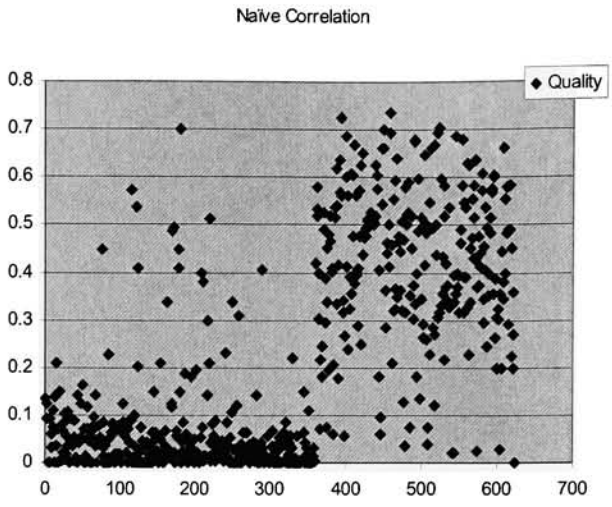


Figure 17: Naive correlation match quality, bin based differencing

Results shown for each entry image library

Figure 18 shows the top five matches for the provided sample image. The heuristic application results in a quality that is directly proportional to the particle weight assigned. This correlation proceeds by first identifying the location and orientation in the map that most closely matches the current pose estimate, then using the heuristic to generate a difference measure.

Figure 18: Sample naive correlation, best matches

Match quality of 0.838 (#452), 0.836 (#180), 0.832 (#524), 0.827 (#547) and 0.824 (#493), respectively

A sample image (Figure 19) is compared with the best fit entry based on position (Figure 20) from the image library. Figure 21 Shows the red, green and blue channel histograms superimposed. Finally the resulting difference measure is presented in Figure 22.

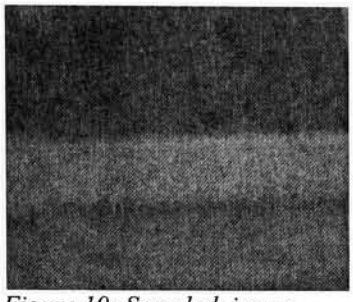


Figure 19: Sampled image

78.94x 15.98y – 89°

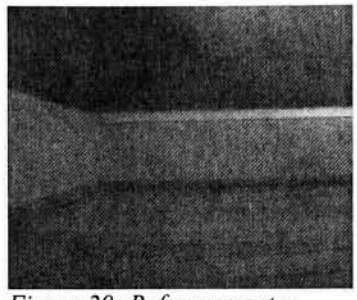


Figure 20: Reference entry

80.0"x 8.0"y – 90°; Entry #27

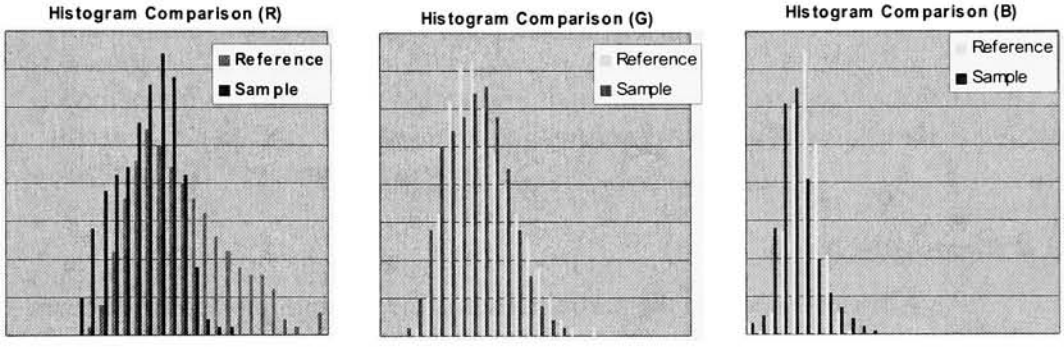


Figure 21: Comparison of sample and reference image

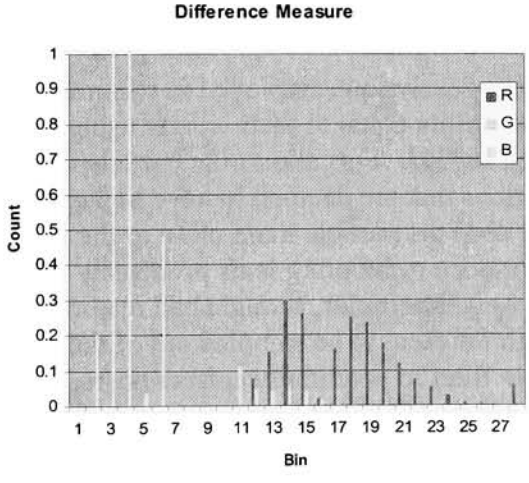


Figure 22: Sample difference measure

Note one undesirable effect of this heuristic is that a sizable amount of color information in the range that shows up strongly in the noise floor is discarded (Figure 23). Consequently the difference measure has undue bias towards images which differ distinctly in these bins.

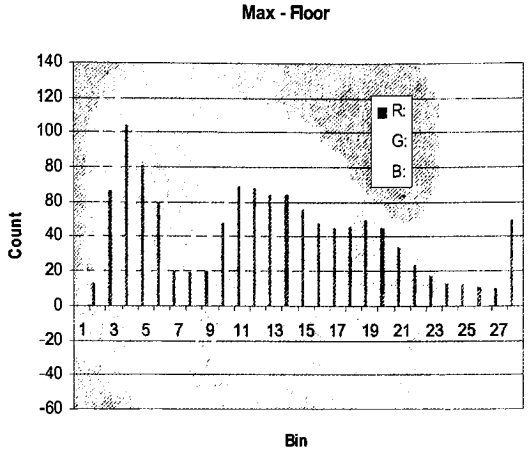


Figure 23: Adjusted reference histogram

Data shown is across entire image library

3.4 Resampling

A critical aspect of the particle filter is the resampling step that occurs between applications of the sensor model and motion model. This step dictates how the particle population evolves. In its most elementary form, resampling can be performed by simply thresholding the particles by their weight. Repopulation can then occur any number of ways, e.g. copying surviving particles or random samples from the search space.

The resampling technique employed in this implementation propagates particles with a probability equal to their weight using a select with replacement technique described in [31]. This allows the particle mass to more thoroughly explore those positions that are deemed to have higher likelihood by the sensor model, and should keep the particle mass close to the ideal PDF. One notable problem with importance resampling is its propensity for tunnel-vision. Because it focuses on high weight particles, a single bad reading could drive particle weights low enough for them to be sampled out of the population, thereby blinding the particle filter to potentially correct poses. This is particularly pronounced in the kidnapped robot problem, and is discussed further in section 4.3.

3.5 Localization Architecture

The hybrid control hierarchy is a promising mechanism for creating a responsive, yet deliberate system. This approach grants a maximal amount of flexibility to the design, without precluding robust operation. A control hierarchy loosely based on this paradigm has been implemented on top of the custom hardware (consult appendices A and B).

Since the platform cannot localize in real-time due to the processing constraints, it is difficult to implement localization as a behavior separate from the behavior(s) that are actually enacting motion. In the time that it takes the implementation to acquire an image, process and update the pose estimates, the robot itself could be displaced or rotated enough to create a discontinuity between the odometry readings and the image that gets examined. As such, the localization behavior is subordinated to the motion control, which prevents any movement while localization processing is underway. A small, but necessary concession. Note that this approach would not impact the responsiveness of the robot if the platform could localize in real-time. However localization updates would still be at the mercy of the motion API, which is probably not desirable. For example a single forward motion that traversed 5' would only drive one localization update. Continous, real-time localization is approximated by tuning a localization frequency parameter.

3.6 Trials

The localization implementation with naïve image correlation was put to the position tracking task over the course of various trials described in the following sections. The term 'trial' is used to denote a sequence of experiments, where each experiment or 'run' conducted varies one or more key localization parameters.

3.6.1 Success Indicators

The task of localization itself is somewhat subjective, particularly when probabilistic estimation methods are employed, as rarely if ever will a robot be able to posit a guess with 100 confidence that exactly mirrors its real location. For trials conducted here, the localization effort is deemed successful if both of the conditions below are met.

Greater than 90% of the particles are centered about a single point

The standard deviation from the center of mass of these particles is less than 10% of the arena dimensions.

For the test arena the second measure above equates to a circular region 1 foot in diameter. Essentially it is required that a perceived collapse of the PDF occurs about a single point with an arbitrary threshold. A localization run is deemed a failure if either of the following conditions are met.

The time taken to localize is in excess of 5 minutes

The localization behavior has met the success criteria for a pose estimate that is greater than 10% of the arena dimension from the actual robot location

3.6.2 Position Tracking

Being able to consistently present an estimate of robot position is a critical component of a localization strategy. The first trial is consequently concerned with determining the success of the localizer with respect to the position tracking task, which differs from global localization (presented in the following section) only by the initial position estimate that is supplied to the localizer. That is, the particle population at time zero has zero variance and is centered about a single coordinate.

The task put to the robot required the traversal of a distance of 1000" (~83') while remaining well localized. Results for a largely successful position tracking run are shown below. The path the robot actually followed for this run is illustrated in Figure 24.

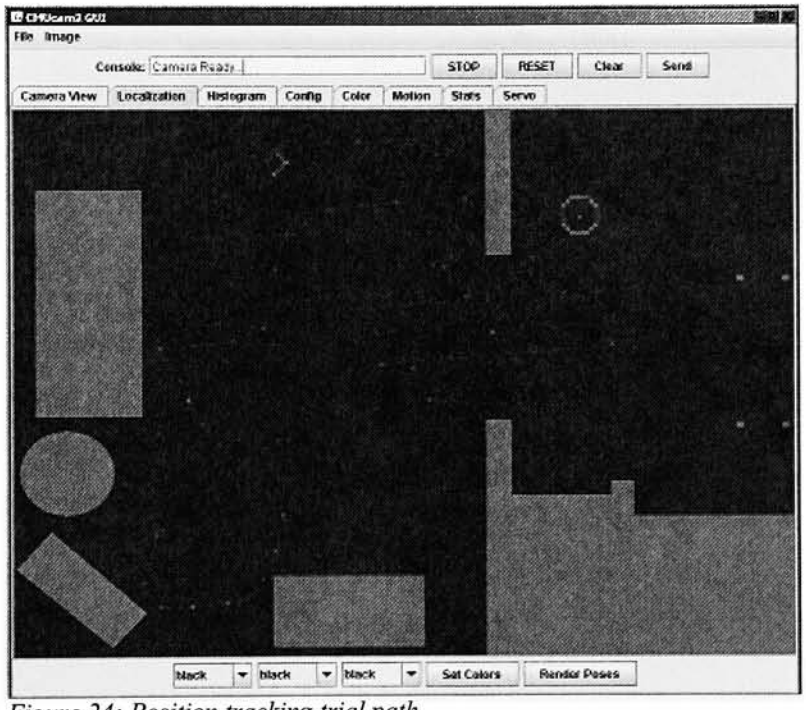


Figure 24: Position tracking trial path

Distance traveled – 61 feet; Arrow indicates starting position;

The final particle population is shown in Figure 25. Position estimates are clustered in a 16" radius around the actual robot location. Note that the actual density can be difficult to interpret based on this graphical representation, as high numbers of particles typically overlap near the center of mass.

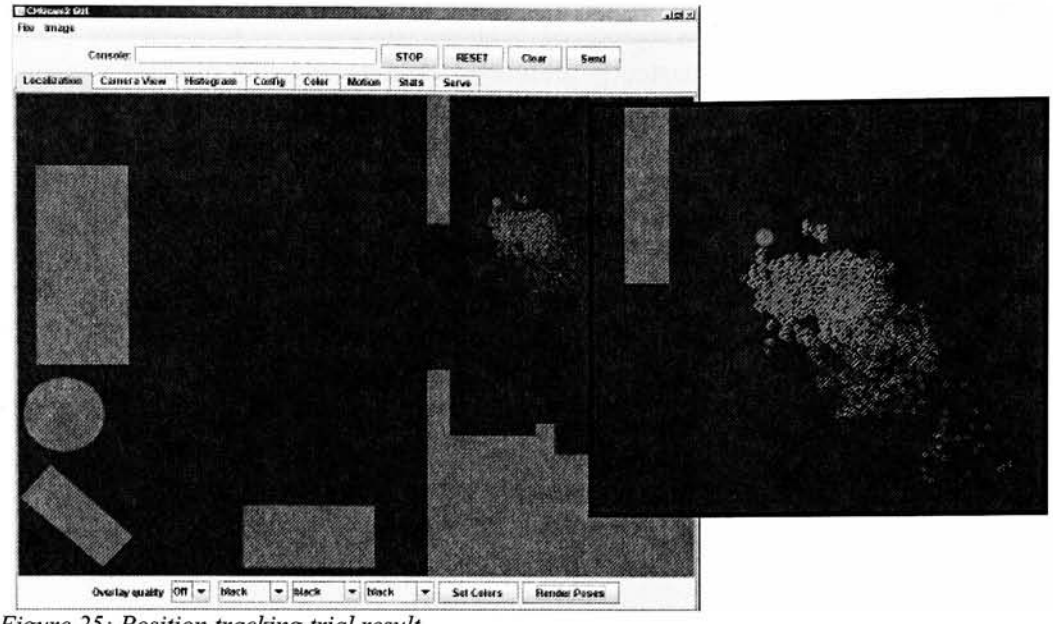


Figure 25: Position tracking trial result

Localizer's final position estimate shown as hollow circle; Actual position shown as solid circle; Final position confidence 0.97; Final error 10"; 5000 particles

A range of data collected from this trial is plotted in Figures 26 and 27. Data is shown for particle counts[1] vs. the distance traveled towards the 1000" goal. Runs that did not achieve the goal are given credit for the distance traversed prior to leaving the 'well-localized' state.

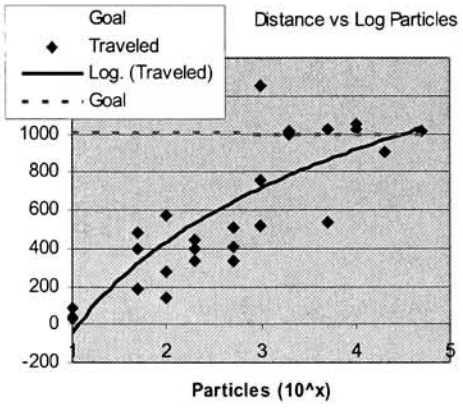


Figure 26: Distance traveled vs particle count, logarithmic scale

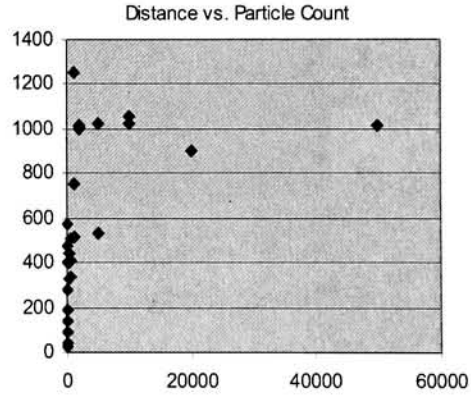


Figure 27: Distance traveled vs particle count

1 While the hardware can theoretically juggle up to 2,000,000 particles, anything above 50,000 is difficult to gather data on due to the time required to perform a single update step on the entire population.

Surprisingly, the robot was able to track its position relatively well at very low particle counts, despite never meeting the goal. In one instance it managed to travel 573.13" while operating the localizer on only 100 particles.

While periodically the localizer developed multiple hypothesis about the current robot position, the only condition that preceded a failed trial (in all cases) was a lack of pose estimates at or very close to the robot's actual position. This dearth of samples, or sample impoverishment, is solely a function of the number of particles available to the particle filter. In practice it is one of the few occasions where the mass of particles does an inferior job of representing the continuous PDF.

As the number of particles employed approaches infinity the particle mass becomes indistinguishable from the PDF. However, for realistic particle numbers it is useful to understand what the 'critical mass' is. That is, at what point does the implementation sufficiently estimate the PDF? Based on the experimentation conducted in this and other trials described below, that number is consistently at or just above 2,000. It is important to note that impoverishment was witnessed at and above this number, however it was always caused by uneven terrain in the arena that induced severe rotational errors. In this case the motion model was not able to generate samples consistent with actual robot motion. This is indicative of the sorts of problems real world robots encounter.

An appropriate response to the sample impoverishment scenario might be to relax the motion model, and even to attempt to account for extreme errors due to unforeseen interactions with the environment. However modeling unobservable errors would be a decidedly unscientific endeavor, further, the likelihood of generating an appropriate sample when say, the robot falls down a flight of stairs, is small. Another approach might be to permit resampling to some extent, from a uniform distribution over the search space whenever match quality dips below a dynamic threshold. High particle counts would be required to ensure an appropriate number of samples are generated outside of the central particle mass. Alternatively, the use of the dual of MCL [7] may be warranted, where replacement occurs with samples drawn from a pool of pose estimates that are consistent with sensor readings. Contrast this latter approach with MCL as previously discussed, where pose estimates are generated only through odometry and vetted by sensor inputs. To generalize, the sample impoverishment problem described here is really just a less extreme case of the kidnapped robot problem, which is discussed further in section 3.6.4 and 4.4.

In addition to continuously supplying a pose estimate, it is important to quantify the accuracy of these estimates in determining the overall effectiveness of the localization implementation. Average error recorded over the course of the trial is presented in Figure 28 for each run, with average error per unit distance traveled shown in Figure 29. Runs are ordered by particle counts, from lowest to highest.

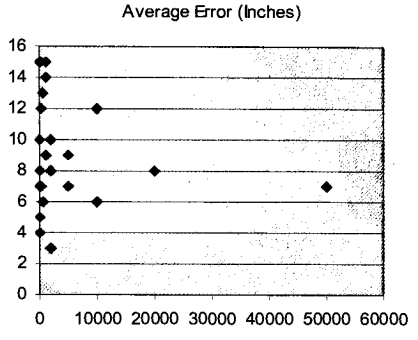


Figure 28: Average error vs particles

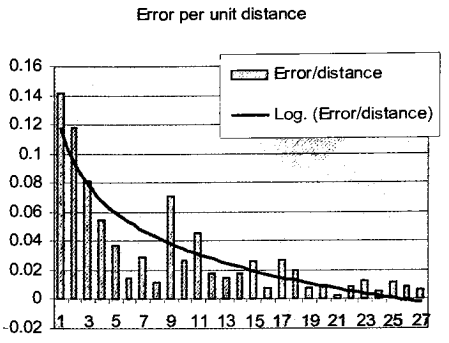


Figure 29: Error per unit distance;

At first glance it appears that positional accuracy increases with the number of particles. However the trend in Figure 29 falls off largely due to low particle count runs not meeting the goal, and consequently traverse a smaller distance. Figure 30 shows the proportional error when the effect of distance is eliminated.

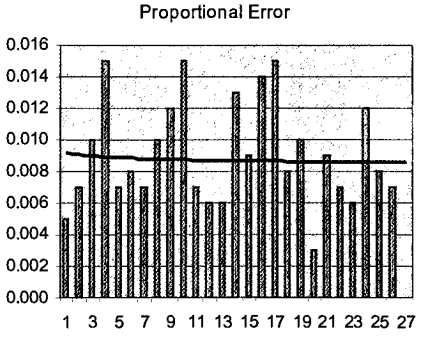


Figure 30: Proportional error

Shown with respect to the 1000" goal

For the position tracking task, the average error is consistent and appears to be independent of particle count at a ratio of about 0.009/1000" (9"). Errors were occasionally witnessed in excess of 30", but typically the robot's position estimate hovered between 0" and 16" of the actual.

To summarize, the particle count appears to only affect the likelihood that the particle filter won't exhaust its supply of position estimates. Beyond this, somewhat counterintuitively, positional accuracy appears independent of the particle population size. To improve upon these results it is important to understand what exactly is motivating the witnessed errors.

At first glance there are two types of localization errors to be concerned with beyond those previously discussed, positional *inaccuracy* and positional *uncertainty* (Figures 31 and 32, respectively). The former refers to distance between estimated and actual robot position, while the latter to standard deviation (which is essentially the inverse of the robot's *confidence* about its position estimate). Clearly both of these terms impact the usefulness of the localization results.

Figure 31: Position inaccuracy

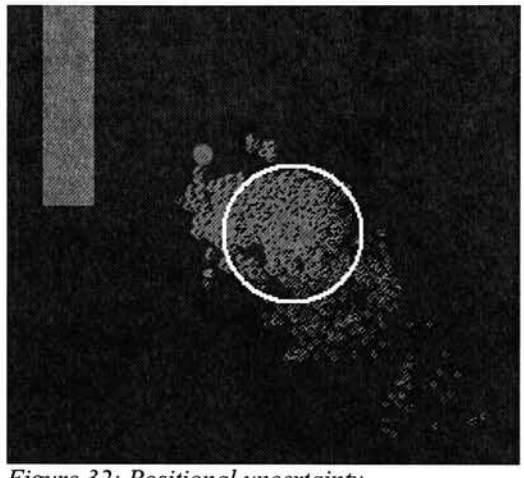


Figure 32: Positional uncertainty

It is hard to assign blame to any specific step undertaken during the localization process, as the position estimate is a product of their close interaction. It is possible that the motion model is failing to generate appropriate samples. However, early position tracking trials employing only the motion model (Figure 12) were verified repeatedly to produce correct particle densities according to robot motion. The successes witnessed in the first trial with respect to longer distances traveled would also seem to rule out such a conclusion. It is possible that the resampling algorithm employed is doing an unsatisfactory job of removing poor estimates from the population. Were this the case it should be possible to reduce both error terms by adding more sensor data without introducing new sample positions. In practice this can be accomplished by rotating the robot about its axis repeatedly. Figure 33 shows the progression of a reasonably well localized robot incorporating additional sensor readings.

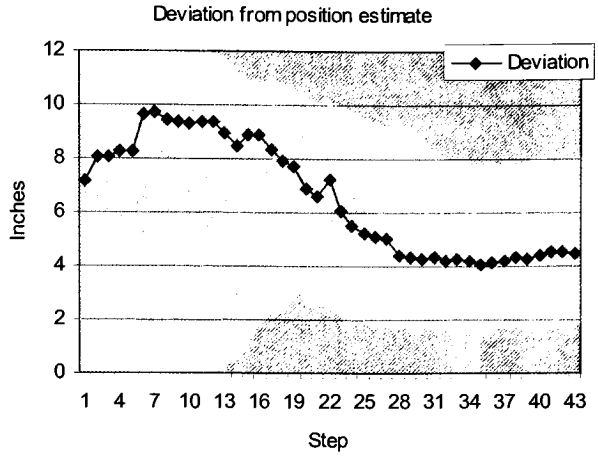


Figure 33: Effect of additional measurements on deviation

The behavior visible in the curve shows that additional sensor information motivates an improvement in overall confidence. However the repeated culling of the particle population without the addition of motion model samples is probably driving the estimate towards the best matches in the image library artificially. This approach undermines the probabilistic nature of the particle filter. Moreover it is unlikely to allow the localizer to arrive at the actual robot position estimate due particularly to differences between reference positions and samples, but also because of lighting variations, image noise and unexpected obstructions. Permitting resampling to occur as solely a function of the particle probability allows the localization implementation to stay true to its mathematical foundation.

Closer inspection of the sensor model's behavior over the course of a few trials reveals a central issue that was alluded to in section 3.3. Figure 34 depicts a particle population superimposed over a representation of sample match quality with respect to the reference map. A group of 8 circles is positioned about a single reference location, each corresponding to a reference orientation. Circle diameter is proportional to the reported quality of the match performed on the sample and corresponding reference image.

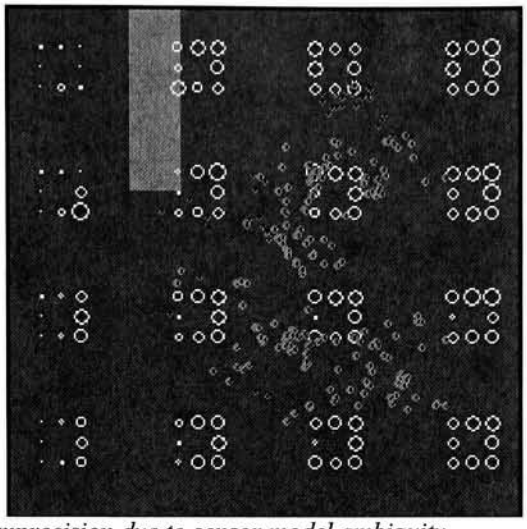


Figure 34: Imprecision due to sensor model ambiguity

200 particles, global localization trial

Match quality for the current sample shown for 8 orientations at each reference position in green. Circle diameter is proportional to match probability

At this position, the sensor model will likely be unable to further refine the position estimate due to the ambiguity in the image correlation. While the probabilities are not equal, they are close enough to delay the ideal collapse of the PDF almost indefinitely. The ambiguity evident here is at the core of the positional uncertainty problem.

3.6.3 Global Localization

To measure the effectiveness against the global localization problem, the robot was moved to a random location in the room and set about the localization task. A sample run from this trial is presented below, with the path the robot followed shown in Figure 35.

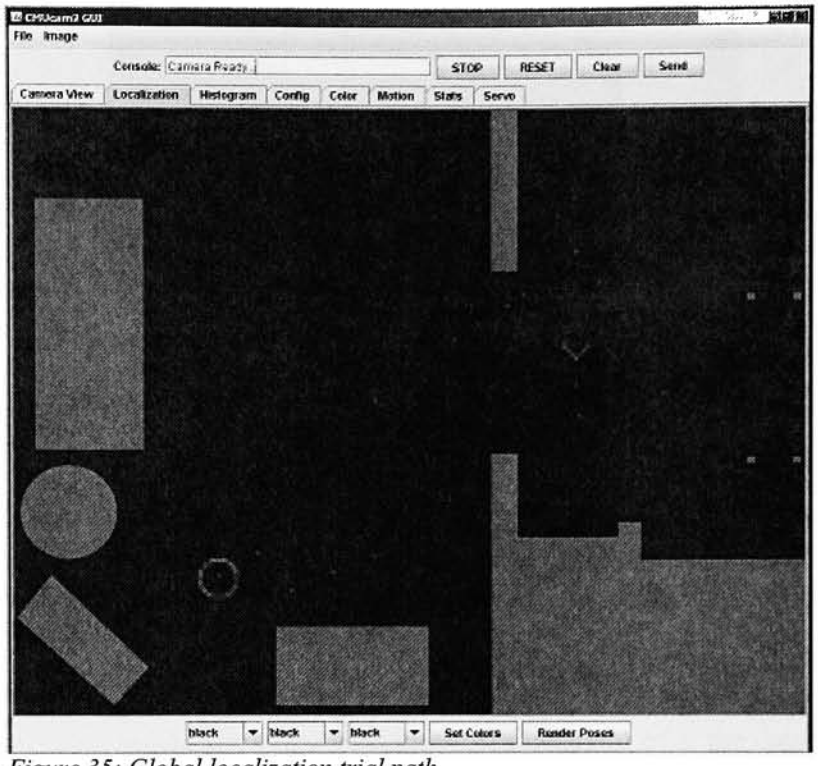


Figure 35: Global localization trial path

This particular run resulted in a localization failure, the progression is shown in Figure 36. Significant errors were witnessed when turns were executed on a transition from wood to carpeted floors. The errors measured were outside of three standard deviations of the expected error built into the motion model for rotational motions. This coupled with sample impoverishment put the localizer in an unrecoverable state. Leaving the robot doomed to wander the arena with an incorrect position estimate until surviving samples by chance overlap with the actual position.

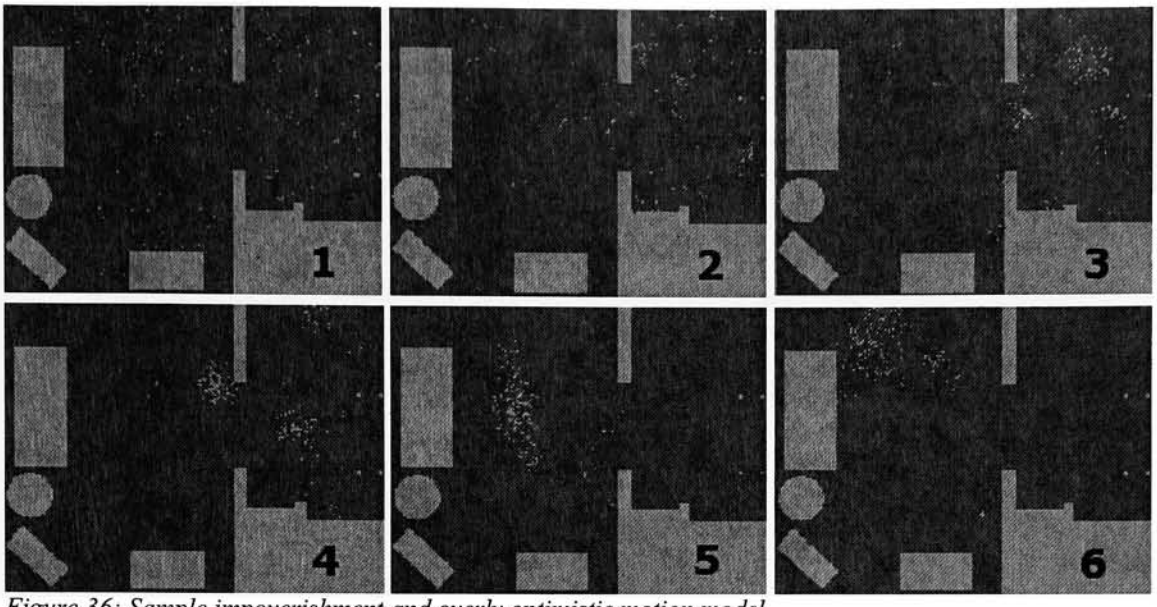


Figure 36: Sample impoverishment and overly optimistic motion model

Failed global localization trial, 500 particles; Confer with path in figure 35

The final error value results for the trial which this example is drawn from are shown in Figure 37. Clusters of runs where this situation arose are marked by square points in the plot. Triangular points mark runs where sample impoverishment occurred during localization. At low particle counts, the random particle generation frequently failed to produce a sample close enough to the robot or at an acceptable orientation.

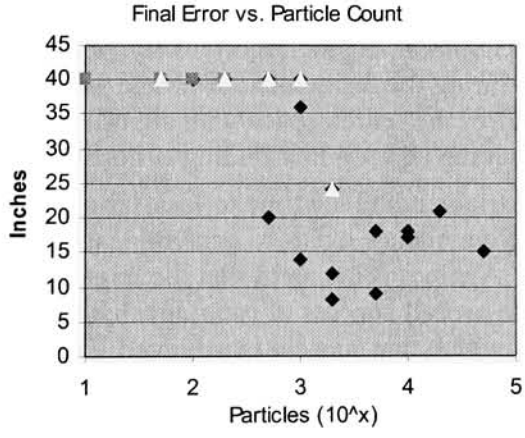


Figure 37: Final positional error vs particle count

The data indicates that global localization has a soft threshold at about 2000 particles, below which point not much chance of success exists. Beyond this

it is difficult to interpret a mathematical relationship between particle count and final error from the trial data. However once the robot is well localized, the behavior of the localizer should mirror that discussed in the preceding section. It is assumed then that further scrutiny of the final error produced by the global localization challenge would lead to the same conclusion. That is, that positional uncertainty and inaccuracy are driven in large part by a sensor model that routinely fails to disambiguate poses in the same region of the arena.

The robot was also required to traverse a 1000 inch distance with an unknown initial position, similar to the first position tracking trial. The results are shown in Figure 38.

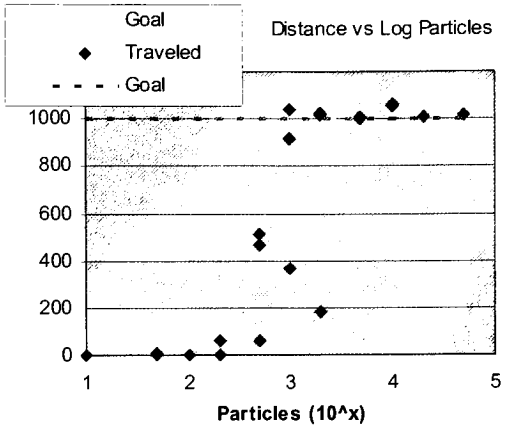


Figure 38: Distance traveled vs particle count

Unlike the distance trial performed in section 3.6.2, there is no steady climb to the goal by localization runs with particle counts lower than ~2000. Sample impoverishment is much more likely, both when initial samples are generated and during the refinement of poses. In the case of the latter, the algorithm explores the search space with significantly fewer particles than during the position tracking task, so this finding is not surprising.

A third trial studied the time to localize for various particle counts, shown in Figure 39. No evidence exists to generalize a dependence of time to localize on particle count. Again it is apparent that the larger particle populations contribute favorably to the overall success of robot localization, but neither to the accuracy nor speed with which that success is achieved. Note that the results shown in Figure 39 are the product of the success measure stipulated at the beginning of the experimentation to a large extent. It is not immediately apparent though that other *useful* success measure(s) would elicit any different a result. Figure 40 shows the collapse of the PDF during a typical trial run. The typical global localization run takes on average 20 to 30 seconds to arrive at an acceptable position estimate (as defined by the success criteria).

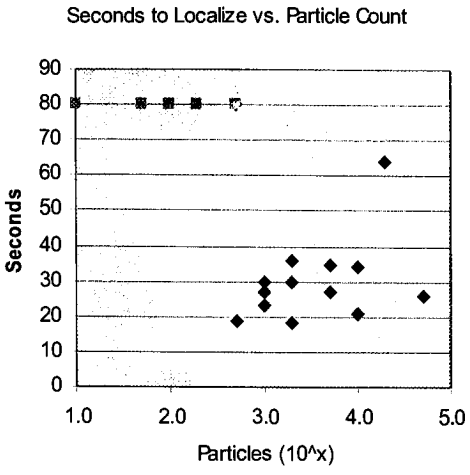


Figure 39: Time to localize

Square points indicate failed runs

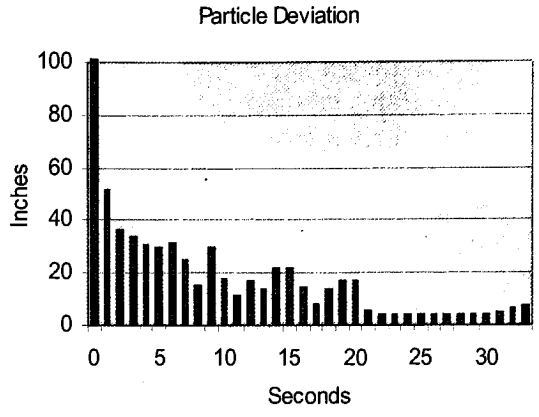


Figure 40: Evolution of robot position estimate

1000 particles

3.6.4 Kidnapped Robot

There is no basis for a solution to the kidnapped robot problem due to the resampling technique that was employed in the initial implementation. While the motion model permits recovery from some small errors in rotation and translation, pose samples are exclusively drawn from the previous population. As such, the lack of samples in the immediate vicinity of the kidnapped robot's new location would be disastrous to the localizer. The net effect would be exactly as described in section 3.5.3 above.

3.6.5 Further Observations

The use of a 'closest match' to source the comparison histogram based on positional information is suspect, as it is not likely that the robot resides in the same position or orientation that the reference image was taken from (hence the term naïve correlation). Clearly color histograms can vary dramatically from different positions, particularly when obstacles are relatively close to the camera. In [8] visibility regions are derived to supply a suitable subset of images that are focused on nearly the same position in the environment. Additional positional information must be made available to make this calculation, such as an occupancy grid or similar. In lieu of visibility regions or a similar technique for improving the ability of the sensor model to drive particle refinement, it may be sufficient to increase the frequency with which the localizer is invoked in addition to refinement of the histogram comparison.

Color histogram comparisons present significant challenges when attempting to produce unique matches to reference images from a sample. Absent the ability to distinguish between images that were taken in close proximity, the sensor model is prevented from refining pose estimates as cited previously. Color histograms themselves can be finicky when employed to match scenes. A sample and reference image pair is shown in Figure 41, followed by histograms of the red, green and blue channels in Figure 42. While a human would likely conclude that the images themselves are virtually identical, the histograms communicate another story to the localizer when the difference measure is employed.

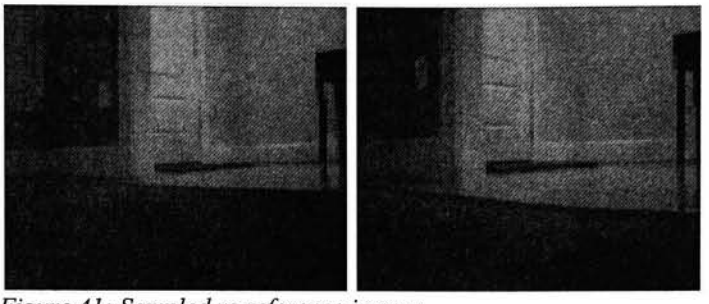


Figure 41: Sampled vs reference images

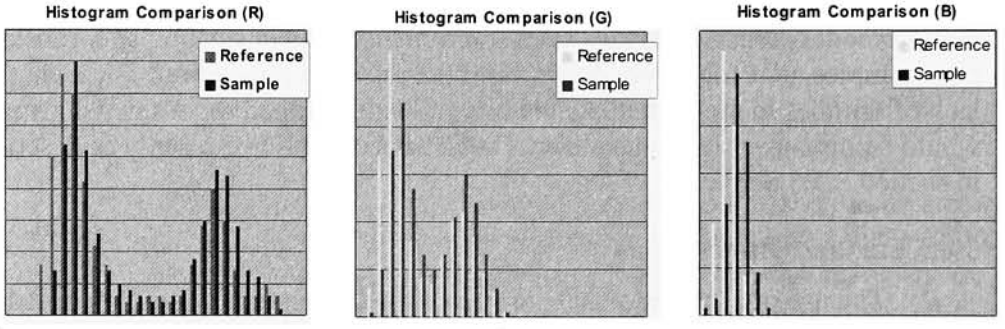


Figure 42: Sample vs reference, histogram shift

The monochannel histogram in Figure 43 highlights the subtle shift in color information between the two images. Lighting in the test arena was consistent during the acquisition of both images.

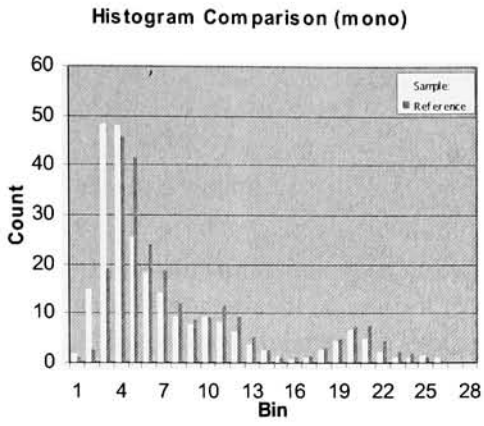


Figure 43: Summation Histogram

The slight change in orientation between sample and reference motivate a portion of the perceived differences, however these shouldn't manifest as a curve shift so much as a change in peak height. Closer inspection of the image and many images like it reveals that these differences stem in large part from a change in noise pixel intensity. Enlarged regions of the sample and reference images are shown in Figure 44. This differences likely come about due to changes in the CCD camera's temperature. Long trials heat up the voltage regulator and on board microprocessor which transfers to the camera.

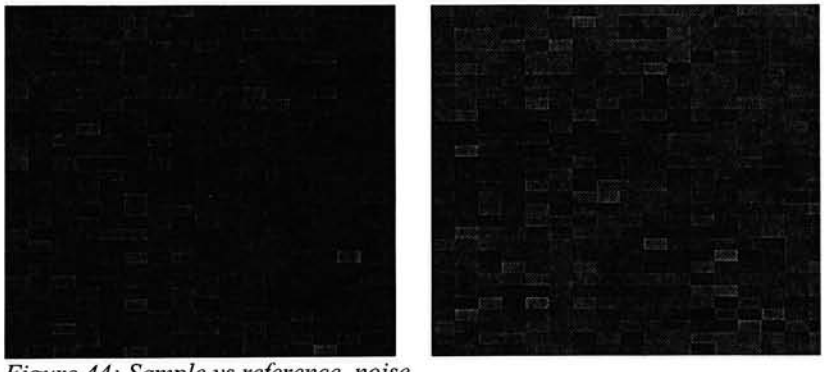


Figure 44: Sample vs reference, noise Enhanced to show pixel differences

These discontinuities negatively affect the sensitive image correlation that is central to the particle filter update. Sample and reference images should employ techniques to reduce the effects of noise.

4 Refined Vision Based MCL

The absence of a solution to the sample impoverishment and kidnapped robot scenarios severely limit the real-world applicability of the implementation described in section 3. To a lesser extent the same can be said of the errors witnessed in position estimation. This section discusses a revised implementation that addresses each of these shortcomings.

Previous results indicated particles counts above 2000 did not affect the global localization result with respect to time to localize, positional accuracy or positional certainty. Consequently the focus of the trials conducted to validate the new implementation were in the 10-2000 particle range.

4.1 Motion Model

The motion model is the source of the randomness that permits the particle filter to explore the searchspace of possible poses. As witnessed previously, the motion model is actually constraining this exploration when extreme odometric errors occur. In all cases these errors were observed when one or more of the rover's wheels were unweighted when traversing different types of terrain. Since the rover only translates forward, executes turns about its center of gravity, is skid-steered (both wheels in a bank turn at the same rate) and has independent odometry, it is feasible to compensate for this type of error with a relatively simple change to the motion model. Specifically, the implementation takes the form of a heuristic that curtails wheel travel estimates when significant departures from expected encoder readings are detected on the same motor bank.

4.2 Sensor Model

The analysis performed in section 3 indicates the sensor model is largely responsible for both positional uncertainty and positional inaccuracy. Revisions to the implementation are described in the following sections.

4.2.1 Image Smoothing

The strong noise component evident in the reference image library is drastically reduced by a simple smoothing operation. This operation is accomplished by convolving the following 3x3 pixel averaging mask about the given image.

$$\begin{vmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{vmatrix}$$

A sample image and its smoothed counterpart are shown in Figures 45 and 46, respectively.

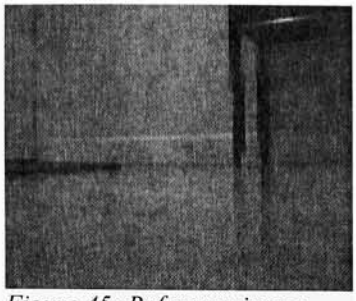


Figure 45: Reference image

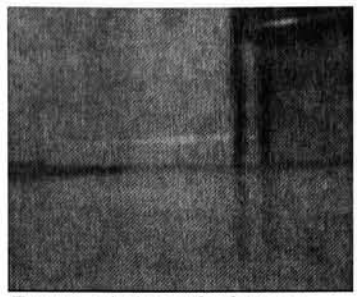


Figure 46: Smoothed image

Differencing the histograms of the two images is shown in Figure 47. Since the smoothing operation is performed on a per channel basis, the quantities of each color are preserved, but theoretically now represent more accurately the color distribution in the initial scene.

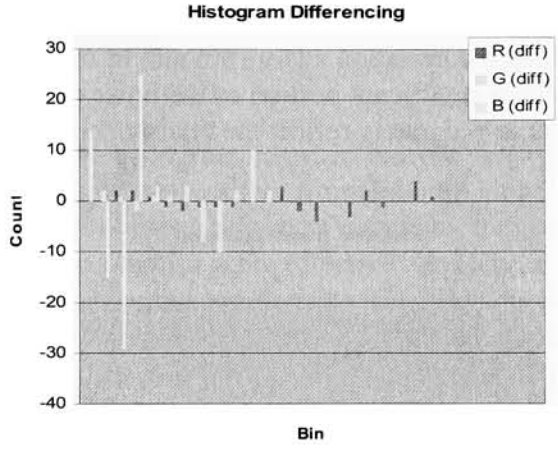


Figure 47: Smoothed histogram differencing

The effect this has on image correlation is shown in Figures 48 and 49, where the same sample image is compared against the image library before and after smoothing. In the first, a significant amount of ambiguity is obvious below comparisons producing probabilities of 0.5 or less.

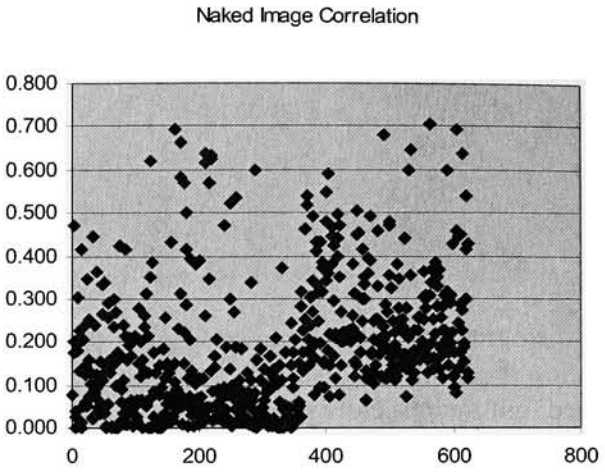


Figure 48: Sample image correlation to image library

In the smoothed correlation a large amount of the sensor ambiguity has been removed. With a significant portion of the noise removed the match quality produced should more accurately reflect the relationship between the sample and reference images.

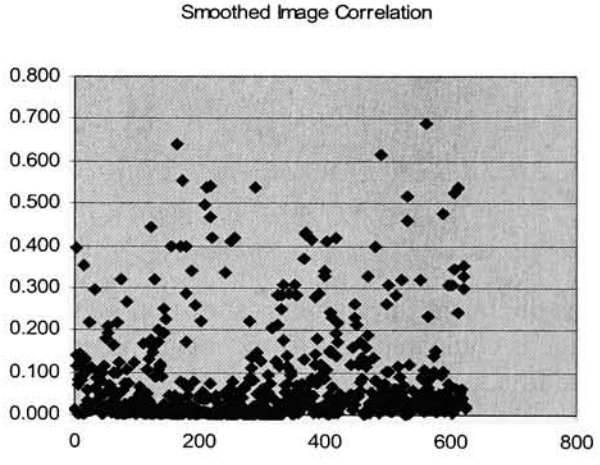


Figure 49: Sample image correlation to image library, smoothing incorporated

A sample run is shown in Figure 50. The robot was directed along the same path used to generate the results shown in Figure 40. The position estimate is less prone to deviate once the particle population collapses about the robot's actual pose, which occurs at about the eighth second. The use of smoothed images in the comparison improves the time to localize by between 5 and 10 seconds.

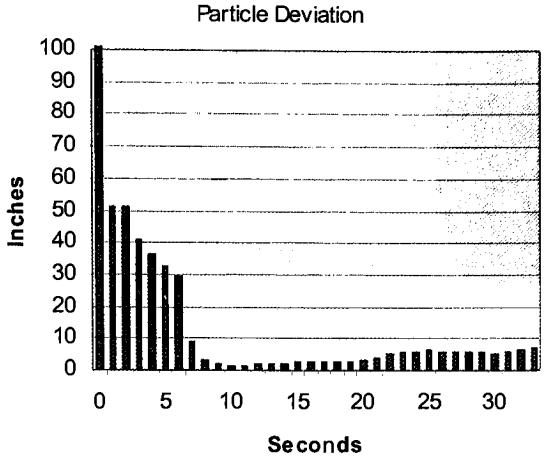


Figure 50: Evolution of particle deviation

1000 Particles

4.2.2 Edge Detection

Employing the histogram based colorspace analysis has proved to result in ambiguity that prevents the localizer from obtaining high positional accuracy. The color vision sensor provides a wealth of information that can be exploited in a variety of ways to accomplish more effective correlation. The extraction of edge information is one such way, and permits the exploration of the arena in a new dimension. Most methods that produce edge information are sensitive to noise in the images they operate on. As such sample and reference images are averaged prior to extraction of the edge data. A Sobel edge detector was applied globally, with 3x3 horizontal and vertical edge detection matrices respectively, which are defined below. A sample image after application of the edge detector is presented in Figure 51.

$$\begin{vmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{vmatrix} \quad \begin{vmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{vmatrix}$$

Figure 51: Edge data derived from sample

Individual channels

Perceiving only edge data in the environment presents a slight problem in that a computationally inexpensive means for correlating detected edges, that is both translation and scale invariant, is not known by the author to exist. The invariant qualities are critical elements of the correlation step, as map images are only available at discrete intervals. It is all but guaranteed that the sample image used in the comparison will be from a different perspective than the reference. The solution is to perform correlation by way of an approximation of edge content. Color histograms are an effective means of doing just that. Though a substantial amount of information produced by the edge operator will be discarded, more than enough is retained to make the comparison meaningful. The histogram's bin distribution will consistently be proportional to the number, intensity and color of edges present in the original edge extraction. Note edge orientations and relative positions are among the more significant data lost in the conversion.

Color histograms produced for images processed in this manner (Figure 52) look significantly different than their non-edge counterparts. The pixels are tightly clustered around low intensities, corresponding to regions of the image that contain little to no edge information. This background pixel information is ignored by way of thresholding when comparing edge histograms to amplify the effect of edges in the source (Figure 53).

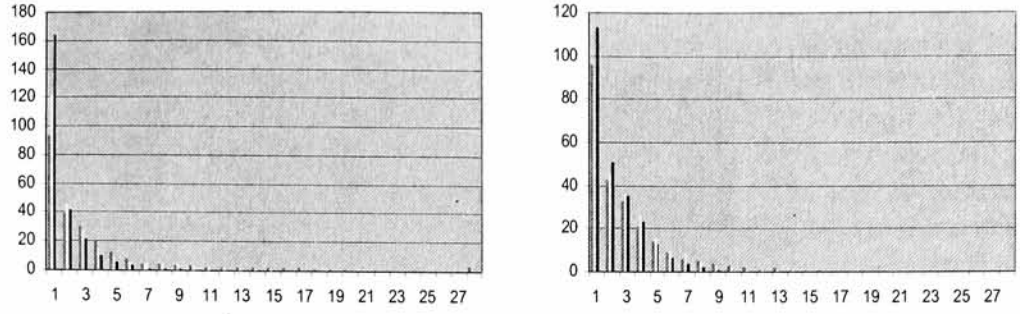


Figure 52: Color histograms of edge images

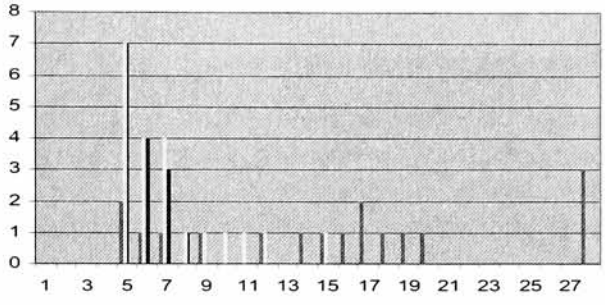


Figure 53: Edge histogram differencing

The best matches for the sample image (Figure 51) against the edge data in the image library reference are shown in Figure 54. These upper tier match results are tightly clustered around the sample image location and so are presumed to be capable of driving the localizer to arrive at position estimates.

Figure 54: Best matches across edge library

0.896 (#258), 0.859 (#420), 0.857 (#282), 0.840 (#242), 0.816 (#450), respectively

Match results across the entire library for the sample image are presented below in Figure 55.

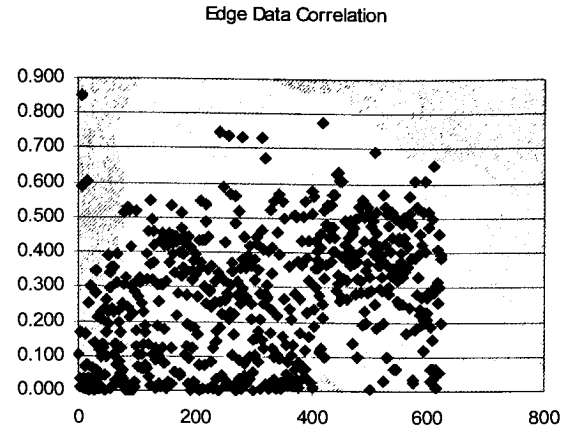


Figure 55: Edge data correlation

The correlation results are more ambiguous (most noticeably below qualities of 0.6) than those obtained from the raw image comparison (Figure 48). As the sensor model upgrade was undertaken to avoid just such a condition, comparison of edge data histograms alone are of questionable value in this endeavor. A sample run along the same path used when previously testing particle collapse (cf Figures 40 and 50) is shown in 56. As expected, it is evident that edge data behaves poorly with respect to the two important measures of localization performance, mean time to localize and positional certainty. In this example the robot took in excess of 25 seconds to arrive at a satisfactory estimate, and proceeded to regress into uncertainty in the 28th second of the run. This is consistent with the behavior witnessed across the trial.

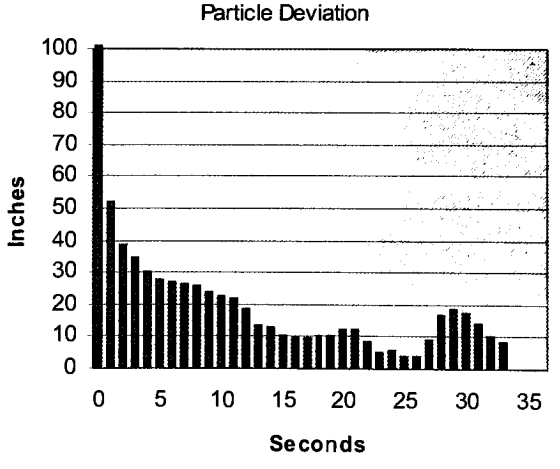


Figure 56: Evolution of particle deviation

4.2.3 Blended Features

The use of multiple features dramatically improves the uniqueness of the correlation conducted by the sensor model as is implied by the distribution shown in Figure 57. Note particularly the large number of samples with virtually zero probability which will rapidly be resampled out of the particle distribution. Smoothed color histogram and edge histogram data was used for this comparison. The match qualities are much lower (even for the best matches) than previous correlation results due to the combination of the two probabilities. Note this has no bearing on the particle filter update, as all particles weights are normalized after each iteration.

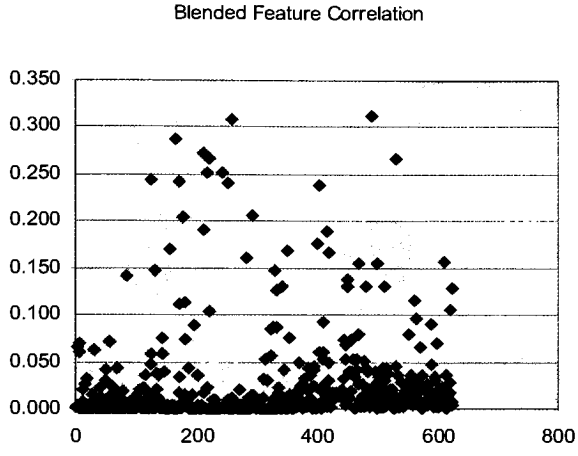


Figure 57: Blended feature correlation

Particle deviation from the pose estimate for a sample run is shown in Figure 58. In general, the implementation has proven to be adept at remaining confidently localized once the particle distribution collapses about a position estimate. This is attributed to the reduction in ambiguity witnessed in the sensor model.

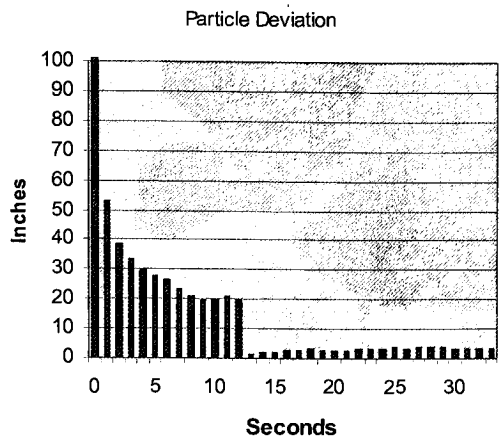


Figure 58: Evolution of particle deviation

Figures 59 and 60 plot final estimate error versus particle count for blended features and color histograms, respectively. The data shown is across an entire global localization trial. In all graphs, square points mark runs which failed to generate sufficient samples to bootstrap the localizer, while triangular points mark runs that succumbed to sample impoverishment.

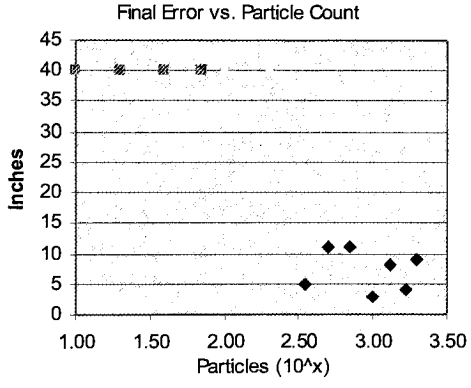


Figure 59: Error vs particle count, blended features

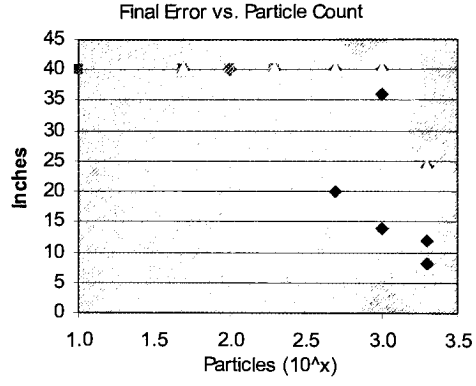


Figure 60: Error vs particle count, color histogram

The data reveals that the accuracy of the position estimate appears to have increased. The mean final error for successful localization attempts employing only the color histogram is 17.7", while the same measure for the blended features is 7.3". For this trial, the traversal of a minimum distance was not required. This is a key difference between this global localization trial and that of section 3. In theory, however, the distance traveled should be independent of the error in position estimate.

It is interesting to note that the blended features have lowered the critical particle mass at which sample impoverishment is encountered. It would seem the less ambiguous sensor model forces the particle population to explore the most likely poses early on. This equates to more particles per pose, yielding a more accurate representation of the PDF at each pose. That is, a smaller likelihood that any of the given poses will succumb to sample impoverishment at the local scale. The net effect is a decrease in the number of particles required to successfully localize.

Shown below are the differences in time to localize for blended features (Figure 61) and color histograms (Figure 62). The mean time to localize for blended features is 19.1 seconds, for color histograms it is 29.2 seconds. However it is difficult to conclude whether or not any real improvement has been made due to the small number of successful trials in this particle range.

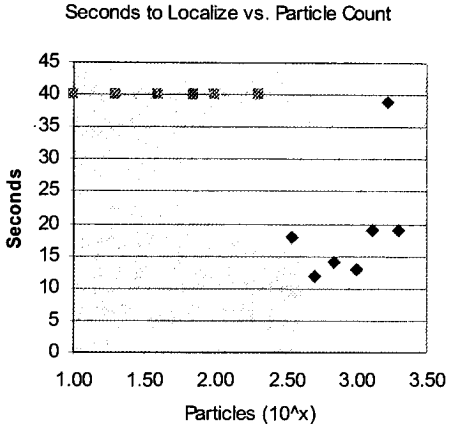


Figure 61: Time to localize, blended features

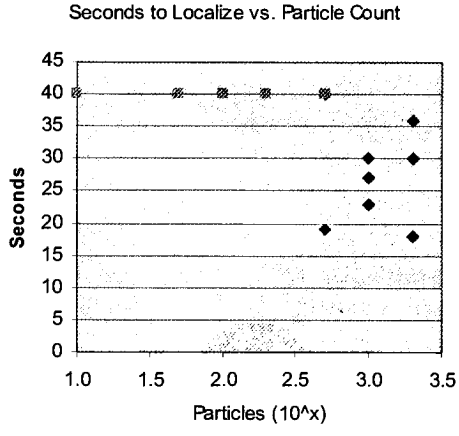


Figure 62: Time to localize, color histograms

4.2.4 Reference Image Selection

As implemented, the sensor model selects a single best-match image from the image library to generate a difference against a sample, a weakness noted earlier. The simplicity of this approach comes at the cost of particle weightings that can be at odds with reality, particularly when the robot is in close proximity to an obstacle. This typically manifests as weights that are driven significantly higher or lower than its actual position with respect to the reference image should allow. This is especially true for samples taken farthest from a reference image. A better means for deriving the quality measure is to make use of multiple images in the vicinity of the pose. Such a method was implemented, where each reference image then contributes to the final measure with a proportion equal to its distance from the pose. While this is clearly still susceptible to obstructions and can induce incorrect weights, it is believed to be a marked improvement.

4.3 Reseeding

A significant problem with MCL is the algorithm's tendency to exhaust its ready supply of particles when sensor or motion model errors outside expected bounds occur. It has been suggested [7] that the use of a dual mode sampler can aid in preventing this condition from arising. The key idea being that the sensor apparatus can additionally serve to generate new position estimates by comparing samples to the reference map. Sample output of such a process is shown in Figures 18 and 54. For the purposes of this paper, this process is referred to as reseeding.

The logical place to implement the reseeding seems to be inside the sensor model. However the sensor model's role in the particle filter is to reform the existing PDF, simulating the effects of the $p(o_t|x_t)$ term in the update equation. Seeding new position estimates in the sensor model has no real mathematical basis. Further, incorporation of the reseeding at a stage of the filter prior to the normalization step seems probabilistically unjustifiable.

The motion model is responsible for generating the posterior of pose estimates based on robot motion, tempered by motion sensor feedback. Recall the term $p(x_t|x_{t-1}, a_{t-1})$ in the update equation that motivates the development of the motion model. In the case of this robot, the 'motion sensors' take the form of motor shaft encoders. The notion of reseeding is really just another means of pose sample generation, where the 'motion sensor' happens to be the robot's camera. To put it another way, the question the robot's particle filter is asking for each particle (with respect to the posterior generation step) is: what is a reasonable estimate of my new location given the motion that has just occurred? From a scientific standpoint the shaft encoders are no more qualified to answer that question than the on-board camera. This insight prompted the incorporation of the reseeding into the motion model, where the pose prediction is forked to two different modeling functions based on the normalized particle weight. The first, implementing the model put forth in section 3.2, and the second a function that draws matches from the database (with replacement) that are cohesive with current vision sensor information. The second modeling function is invoked with probability equal to the inverse of the particle weight. That is, the particles with current pose estimates that the sensor model has deemed least likely to be correct are the first to be reseeded. To permit the samples generated by this new split function to properly probe the pose space it is necessary to model the error inherent in the vision sensor. The process for doing so is generally the same as was described in section 3.2.1 and 3.2.2. The net result is the addition of Gaussian noise to the image sample prior to executing the correlation, example shown in Figure 63.

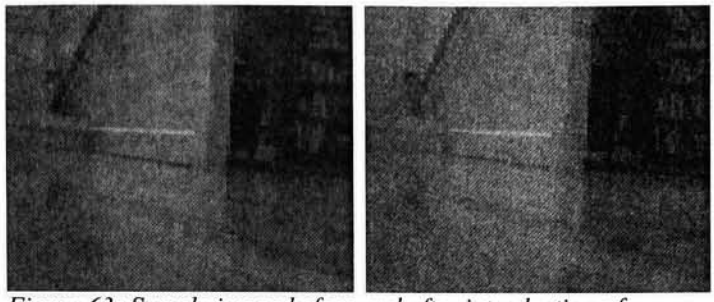


Figure 63: Sample image before and after introduction of Gaussian noise

Noise intensity exaggerated

It is evident after only a few runs at low particle counts that the sensor driven motion model and new reference image selection largely eliminate the sample impoverishment scenario that plagued the initial implementation. Successful global localization runs are witnessed with particle counts as low as 20. Figure 64 illustrates the positional certainty over time of a such a run.

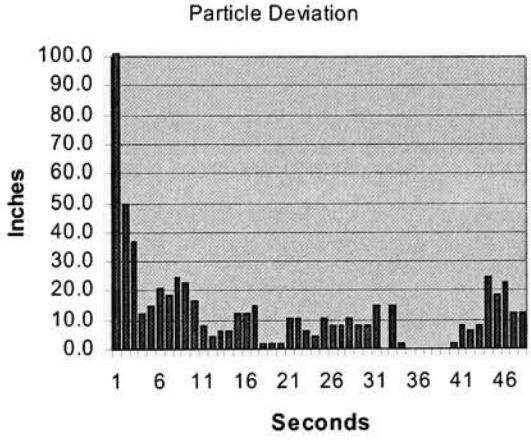


Figure 64: Global localization run, 20 particles

The positional certainty graph here is somewhat misleading. While these small particle count instantiations of the localizer can rapidly localize, it is frequently the case that samples in the immediate vicinity of the robot disappear as quickly as they arrived. With respect to the run depicted in Figure 64, initial localization occurred at 11 seconds and then tracked surprisingly well for 4 seconds, at which point the previous particle agreement disintegrated. This same scenario played out repeatedly for the duration of the run. At low particle numbers, the distribution is highly sensitive to the fluctuations in match quality and the inaccuracies still inherent in the updated reference image selection technique.

An unexpected finding with vision based sampling is a net decrease in positional certainty. Particles generated by the vision sensor have a tendency to accumulate in regions that correlate even marginally well with sampled images. This same scenario is depicted in Figure 65. Particles to the far right and left of the central mass are largely sourced from the vision based sampler.

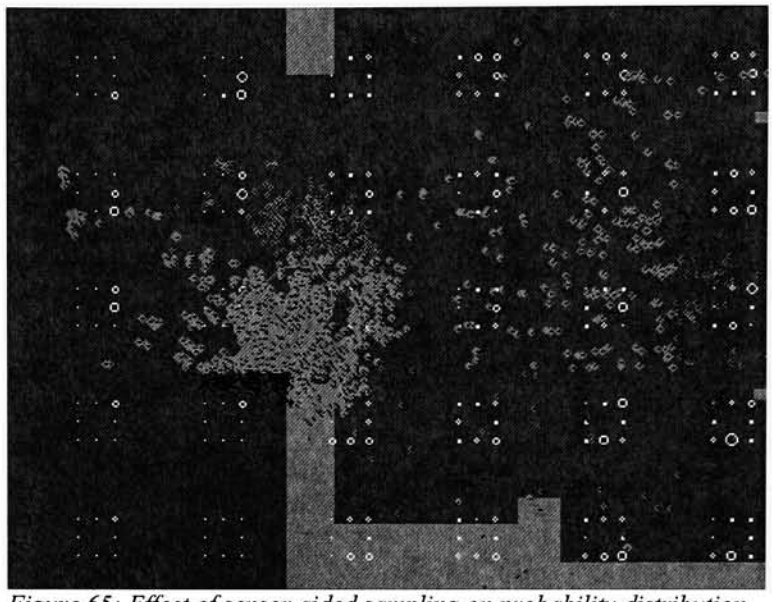


Figure 65: Effect of sensor aided sampling on probability distribution

End of 3000 particle global localization trial, blended features. Actual robot position shown as solid red dot.

It seems the solution to the kidnapped robot problem with MCL is had at the expense of some additional level of ambiguity. Though it may be that the new sampler is just doing a better job of revealing the true probability density function. A plot of the evolution of the positional uncertainty is shown in 66.

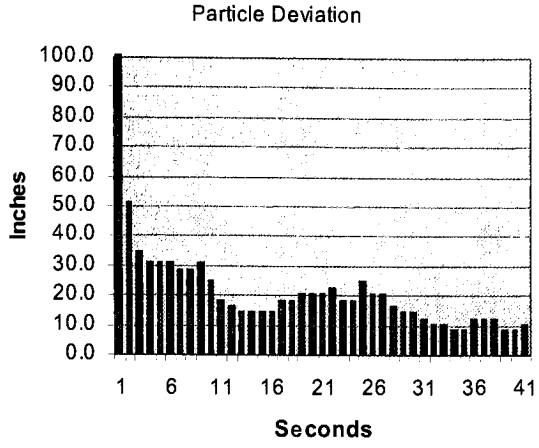


Figure 66: Positional uncertainty

Single run of 3000 Particles (blended features)

Further experiments would seem to confirm this manifestation is independent of particle count. Figure 67 shows uncertainty across all runs, with particle counts ranging from 10 to 3000. Compare this with Figure 68 which shows the same information against the vanilla resampling strategy.

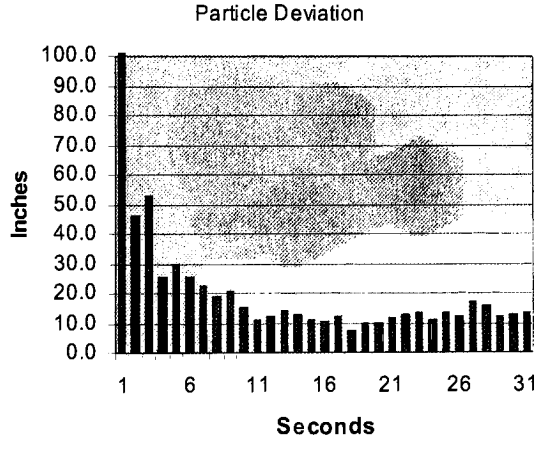


Figure 67: Effect of sensor aided sampling on positional certainty

Average across all trials using sensor aided sampler

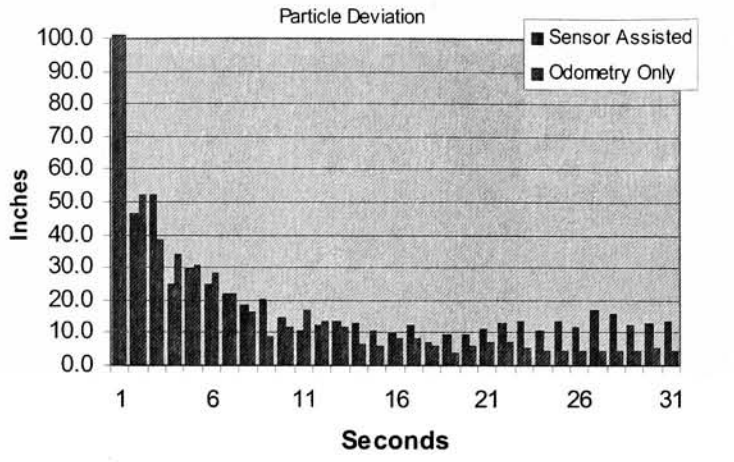


Figure 68: Positional certainty, by sampling strategy

A critical measure of the new sampler is to test how the robot responds to being spontaneously and unwittingly relocated to new position. A sample run from the kidnapped robot trial is shown in Figure 69.

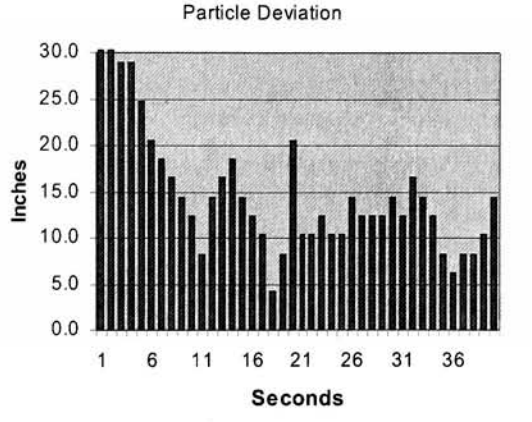


Figure 69: Kidnapped robot trial

1000 particle run, y-axis scaled

In this run the localizer arrived at the correct position estimate after 17 seconds, at which point it was halted and relocated. A brief period of relatively large uncertainty follows this event while the robot explores various possible locations in the environment. A correct position estimate isn't arrived at until about 15 seconds later (t = 35 seconds). In general the kidnapped robot scenario no longer presents a significant problem to the localizer. Arriving at an accurate position estimate after entering the 'kidnapped' state typically takes between one to two times as long as the initial collapse.

Finally, it is worthwhile to discuss the resource usage of the various incarnations of the vision based localizer. Each of the features discussed previously in section 4 require very little CPU time to compute, which is consistent with the goals set out initially. The overhead required to extract each feature is shown in Figure 70 in seconds of measured CPU time. The relative times to execute feature comparison for the four methods are shown in Figure 71.

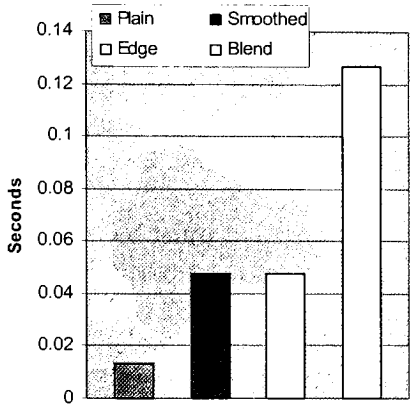


Figure 70: Feature extraction overhead

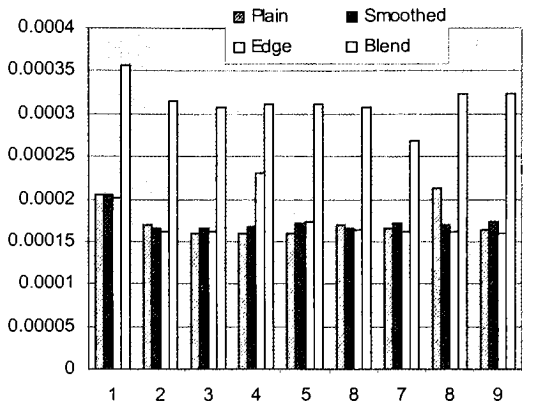


Figure 71: Relative feature comparison times

However, a large increase in the overall CPU time required by the localizer to complete an iteration is now noticeable, particularly when employing a large number of particles. This is a direct result of the sensor assisted sampling. The effects are shown below in Figure 72. The reference column indicates mean time for the same localizer invocation with sensor assisted sampling disabled.

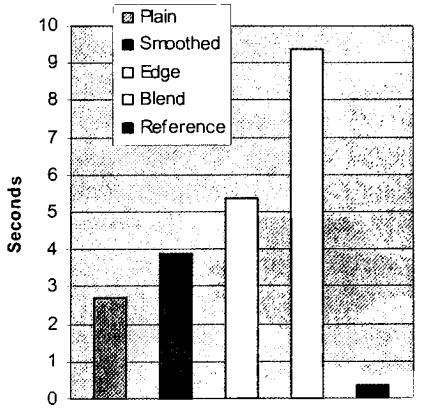


Figure 72: Single localizer invocation CPU time

1000 particles

5 Conclusion

Mobile robot localization is a perpetual challenge to robotics researchers. Not only must the robot arrive at a correct estimation of its position, it must typically do so with a minimum of computational overhead to permit responsiveness to the real-time environment in which it finds itself. Robots must be tolerant of the various unexpected forces acting upon them during their travels in the real world, should their designers expect them to achieve much success. Monte Carlo Localization has proven to be effective in this regard. The construction and tailoring of the custom robot platform has provided a means to study the effectiveness of a vision based MCL implementation. The use of simple visual features obtained from a monocular vision sensor has motivated an understanding of the key requirements of particle filters as they pertain to mobile robot localization.

The cumulative effect of errors in macro level robot motion must be measurable. Furthermore, these errors or their constituent parts must be conducive to modeling. The motion model generates position estimates that form the scientific basis of the discrete PDF representation and permit the exploration of the search space. Sample impoverishment, a situation that arises when the localizer exhausts its supply of particles in the vicinity of the robot, is inevitable in the real world. This condition can be remedied by the judicious application of an inverse sampling technique, where the density representation is supplemented by position estimates generated from sensor input.

The calculated randomness introduced into the particle filter by the motion model is indispensable, but in truth randomness is easy to come by. The sensor model, being responsible for reining in the samples generated by the motion model, bears a disproportionate amount of the burden in arriving at a correct position estimate. It has been shown that simple features such as color histograms and histograms of edge images can be part of a successful localization strategy. While the comparison performed with these features is accurate enough to cause the particle filter to cull many incorrect pose estimates, it is given to reporting similar match qualities in many different poses in the same region of its environment. So it is often the case that the localizer is not able to reduce the PDF to a sufficiently narrow estimate. It is clear that the sensor model must be unambiguous to as great an extent possible in a given locality to allow accurate position estimates to be achieved. Fusing the measures that represent information from different features in the vision data has proved to be an effective means for disambiguation, thereby increasing positional accuracy as well as certainty.

While trivial to extract from the vision sensor, color histograms present challenges when integrated into the sensor model. Scenes in the robot's environment can appear very different in colorspace as scene lighting changes. Figure 73 shows sample and reference histograms of the same scene at different times of day. The histogram bin counts have shifted to the right to reflect the change in overall brightness.

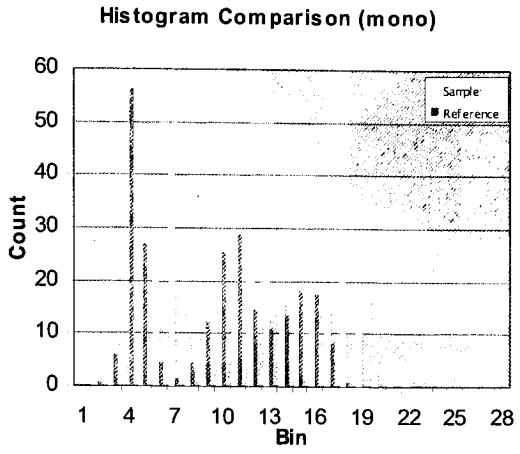


Figure 73: Histogram shift due to brightness changes

A legitimate approach might be to only deal with normalized histograms to minimize the effects of both uniform and isolated scene lighting variations. For example, in [28] it was concluded that normalization improved performance, albeit only for outdoor scenes. Various techniques for determining the 'distance' between color images have been proposed. Examples include color coherence vectors [32] and the earth movers distance [26] among others. To the extent that these techniques can more accurately assess the differences between two images they will be able to improve upon the results achieved with the naïve histogram correlation.

The use of features as described here and in [8] have been proven to be effective at driving particle filters for the purpose of localization. However these features, rotation and scale invariant as they may be, present no real mathematically grounded strategy for accommodating vision sensor perspective changes. Consequently their suitability for many real world environments such as confined spaces is questionable. Potential improvements in a vision based localization approach can likely be arrived at by fusing absolute distance information such as that yielded by laser range finders, sonar or radar. Augmenting the process in this manner would allow texture information to be assigned to surfaces in the environment. An appropriate sensor model could then perform correlation on these surfaces based on robot pose, obviating the need for loose approximations.

Lastly, it would be an interesting undertaking to explore more appropriate and efficient means of gathering, storing and retrieving reference images. Moreover, the impact of such factors as quantity, uniformity, resolution and interval of reference images has not been studied. It is very likely that these factors are able to motivate more effective localization strategies.

6 References

1. R. A. Brooks, A Robust Layered Control System for a Mobile Robot, *IEEE Journal of Robotics and Automation*, 2(1), 14-23, 1986
2. J. Rosenblatt, DAMN: A Distributed Architecture for Mobile Navigation, In AAAI Spring Symposium on Software Architectures for Physical Agents, Stanford CA, March 1995. http://citeseer.ist.psu.edu/article/rosenblatt97damn.html
3. A. Rahimi, T. Darrell, Bayesian Network for Online Global Pose Estimation, Proceedings of International Conference on Intelligent Robots and Systems (IROS), 2002. http://www.ai.mit.edu/projects/vip/papers/iros2002.pdf
4. Gerhard Weiß, Christopher Wetzler, Ewald von Puttkamer, Keeping Track of Position and Orientation of Moving Indoor Systems by Correlation of Range-Finder Scans, In *Proceedings of the International Conference on Intelligent Robots and Systems*, 595-601, 1994. http://citeseer.ist.psu.edu/wei94keeping.html
5. D. Fox, W. Burgard, and S. Thrun, Markov Localization for Mobile Robots in Dynamic Environments. Journal of Artificial Intelligence Research, 11:391-427, 1999. http://citeseer.ist.psu.edu/fox99markov.html
6. D. Fox, W. Burgard, and S. Thrun, Markov Localization for Reliable Robot Navigation and People Detection,Proc. of the Dagstuhl Seminar on Modelling and Planning for Sensor-Based Intelligent Robot Systems, 1999. http://citeseer.ist.psu.edu/207759.html
7. S. Thrun, D. Fox, W. Burgard, and F. Dellaert. Robust Monte Carlo Localization for Mobile Robots. Artificial Intelligence, 128(1-2), 2001. http://robots.stanford.edu/papers/thrun.robust-mcl.pdf
8. Jürgen Wolf, Wolfram Burgard, Hans Burkhardt, Robust Vision-based Localization for Mobile Robots Using an Image Retrieval System Based on Invariant Features, IEEE International Conference on Robotics and Automation, 359-365 vol.1, 2002. http://www.informatik.uni-freiburg.de/~burgard/postscripts/wolf_icra02.pdf
9. P. Jensfelt. Approaches to Mobile Robot Localization in Indoor Environments. PhD thesis, Signal, Sensors and Systems (S3), Royal Institute of Technology, SE-100 44 Stockholm, Sweden, 2001. http://citeseer.ist.psu.edu/jensfelt01approaches.html
10. S. Thrun. Probabilistic Algorithms in Robotics. AI Magazine, 21(4):93-109, 2000. http://citeseer.ist.psu.edu/thrun00probabilistic.html
11. J. Gutmann, W. Burgard, D. Fox, K. Konolige, An Experimental Comparison of Localization Methods, in Proc. of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS'98), 1998. ftp://ftp.informatik.uni-freiburg.de/documents/papers/ki/gutmann-etal-iros98.ps.gz
12. K-Team BIOS Reference Manual, http://www.k-team.com/download/khepera/documentation/KheperaBIOSRefManual.pdf
13. K-Team K6300 Manual, http://www.k-team.com/download/khepera/documentation/K6300Manual.pdf
14. Arkin, R. C., Behavior-Based Robotics, Intelligent Robotics and Autonomous Agents, The MIT Press. Cambridge, Massachusettes, 1998.
15. Luger, G. F., Artificial Intelligence: Structures and Strategies for Complex Problem-Solving, Addison Wesley, 2002.
16. Robin R. Murphy, An Introduction to AI Robotics, The MIT Press. Cambridge, Massachusettes, 2000.
17. Russell, S., Norvig, P., Artificial Intelligence: A Modern Approach, Prentice Hall Series in Artificial Intelligence. Englewood Cliffs, New Jersey, 1995.
18. Y. L. Murphy, J. Chen, J. Crossman, J. Zhang, P. Richardson, L. Sieh, DepthFinder, A Real-time Depth Detection System for Aided Driving, Intelligent Vehicles Symposium, 2000. Proceedings of the IEEE, 122 – 127, 2000. http://www.engin.umd.umich.edu/~richarpc/00_IVS_monocular.pdf

19. S. Siggelkow and H. Burkhardt. Local invariant feature histograms for image retrieval. Technical report, Institut fur Informatik, Albert-LudwigsUniversit at Freiburg, January 1998. http://citeseer.ist.psu.edu/siggelkow98local.html
20. Stephen Se, David Lowe, Jim Little, Local and Global Localization for Mobile Robots using Visual Landmarks, Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), pages 414 420, Maui, Hawaii, October 2001. http://citeseer.ist.psu.edu/se01local.html
21. Luca Regini, Guido Tascini, Primo Zingaretti, Appearance-based robot navigation, Unknown publication. http://www-dii.ing.unisi.it/aiia2002/paper/ROBOTICA/regini-aiia02.pdf
22. S. Thrun, M. Beetz, M. Bennewitz, W. Burgard, A.B. Creemers, F. Dellaert, D. Fox, D. Hahnel, C. Rosenberg, N. Roy, J. Schulte, and D. Schulz, International Journal of Robotics Research, Vol. 19, No. 11, November, 2000, pp. 972-999 http://www.ri.cmu.edu/pub_files/pub2/thrun_sebastian_2000_2/thrun_sebastian_2000_2.pdf
23. F. Dellaert, W. Burgard, D. Fox, and S. Thrun, Using the Condensation Algorithm for Robust, Vision-based Mobile Robot Localization, IEEE Computer Society Conference on Computer Vision and Pattern Recognition, June, 1999. http://www.ri.cmu.edu/pub_files/pub1/dellaert_frank_1999_3/dellaert_frank_1999_3.pdf
24. H. Schulz-Mirbach. Invariant gray scale features. Internal Report 8/96, Technische Informatik I, Technische Universit at Hamburg-Harburg, 1996. ftp://ftp.informatik.uni-freiburg.de/papers/lmb/hsm_ib96c.ps.gz
25. Hanns Schulz-Mirbach. Invariant features for gray scale images. In G. Sagerer, S. Posch, and F. Kummert, editors, 17. DAGM - Symposium \Mustererkennung ", pages 1-14, Bielefeld, 1995. Reihe Informatik aktuell, Springer. http://citeseer.ist.psu.edu/schulz-mirbach95invariant.html
26. Y. Rubner, L.J. Guibas, and C. Tomasi, "The Earth Mover's Distance, Multi-Dimensional Scaling, and Color-Based Image Retrieval," Proc. DARPA Image Understanding Workshop, pp. 661--668, May 1997. http://robotics.stanford.edu/~rubner/papers/rubnerIuw97.pdf
27. W. Burgard, A.B. Cremers, D. Fox, D. Hahnel, G. Lakemeyer, D. Schulz, W. Steiner, and S. Thrun. Experiences with an interactive museum tour-guide robot. Technical Report CMU-CS-98-139, Carnegie Mellon University, Computer Science Department, Pittsburgh, PA, 1998. http://citeseer.ist.psu.edu/article/burgard98experience.html
28. P. Blaer and P.K. Allen. Topological mobile robot localization using fast vision techniques. In IEEE International Conference on Robotics and Automation, pages 1031-1036, Washington, USA, May 2002. http://www.cs.columbia.edu/~pblaer/papers/icra2002.pdf
29. M. J. Swain and D.H. Ballard. Color indexing. International Journal of Computer Vision, 7(1):11-32, 1991.
30. M. A. Stricker and M. Orengo, Similarity of Color Images, SPIE Proceedings Vol. 2420, 1995.
31. Cooperative Localization and Multi-Robot Exploration, I. M. Rekleitis. Ph.D. thesis, School of Computer Science, McGill University, Montreal, Quebec, Canada, 2003. http://www.cim.mcgill.ca/~yiannis/Publications/thesis.pdf
32. G. Pass, R. Zabih, and J. Miller. Comparing images using color coherence vectors. In Proceedings of ACM Multimedia 96, pages 65--73, Boston MA USA, 1996. http://citeseer.ist.psu.edu/article/pass96comparing.html
33. J. Domke, Y. Aloimonos, "Deformation and viewpoint invariant color histograms", Proc. BMVC (British Machine Vision Conference, September 2006, Edinburgh, UK. http://www.cs.umd.edu/~domke/papers/2006bmvc.pdf

Appendix A – Hardware Specification

Key limitations in RAM, Flash and CPU were identified early on with the Khepera II robot (Figure 74) and its vision turret (Figure 75), the initial platform upon which research was to be conducted. Particularly problematic was the small amount of Flash when considering the feature database required to implement the proposed vision based MCL. The specifications for the Khepera are shown in Figure 76.

Figure 74: Khepera II robot

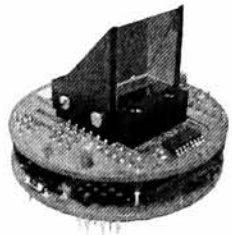


Figure 75: K6300 matrix vision turret

Processor	Motorola 68331 @25MHz
RAM	512 KB
Flash	512 KB
Motion	2 DC motors w/ incremental encoders
Speed	Max: 1 m/s, Min: 0.02 m/s
Sensors	8 Infra-red proximity sensors
I/O	3 Analog Inputs (0-4.3V, 8bit)
Power	Rechargeable NiMH Batteries
Communication	Standard Serial Port, up to 115kbps
Size	Diameter: 70 mm
	Height: 30 mm
Weight	Approx 80 g
Payload	Approx 250 g

Figure 76: Khepera II hardware specification

A custom built platform allows each of those constraints to be addressed. A brief description of this robot platform follows.

Motion

The rover base is comprised of a Lexan cut chassis (Figure 77) and four wheels, each with its own 7.2V DC motor. None of the wheels articulate, consequently skid steering is the only means of turning. Although skid steer is error prone, this actually lends itself well to the particle filter approach as a healthy amount of randomness in the robot motion is required.

Figure 77: Chassis with motors and wheels installed

Motors are driven through a controller board (Figure 78) that receives pulse-width modulated inputs from the Robostix I/O board (described below). Odometry measurements are accomplished through the use of quadrature encoders (Figure 79) on each of the four motors.

Figure 78: Motor controller board

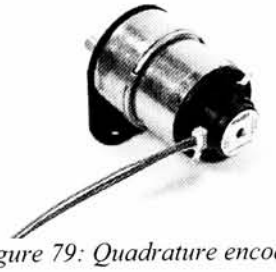


Figure 79: Quadrature encoder

Sensing

Basic collision avoidance is supported via infrared proximity detectors (Figure 80) mounted on the front of the chassis. This will be the lone sensory input to the collision avoidance behavior. Consequently the robot will always favor forward motion.

Figure 80: IR proximity sensor board

Vision will be accomplished by way of the CMUCam2+ (Figure 81) on a fixed vertical mount, facing forward. The CMUCam2+ consists of a color CMOS camera and on-board microcontroller, which provide motion-detection, image grabbing and sources various image statistics. It also supports region-growing and segmenting capabilities typically employed for 'blob-tracking'.

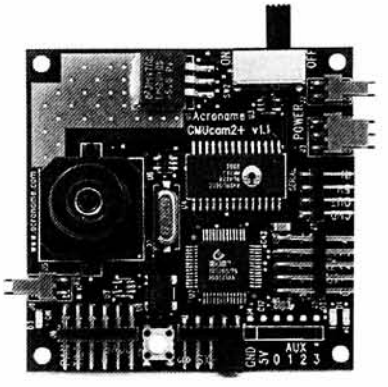


Figure 81: CMUCam2+

Computation

A Gumstix (Figure 82) single board computer (SBC) running a variant of BusyBox Linux from Gumstix, Inc. will be employed to fuse the various sensory inputs and outputs with the localization and general behavior logic.

Figure 82: Gumstix SBC

A Robostix board (also from Gumstix Inc., Figure 83) will provide the I/O breakout and additional PWM channels required to physically connect the SBC to the various sensors and effectors. The SBC and the Robostix communicate primarily over the shared I2C bus.

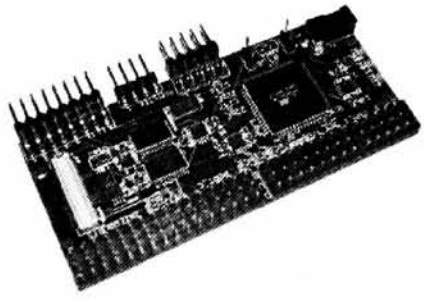


Figure 83: Robostix I/O board

Additional flexibility is provided by the NetCF expansion board shown in Figure 84. This board makes a Compact Flash (CF) card slot available which can host extra storage or a CF WiFi card to permit wireless operation. The NetCF also supplies an Ethernet connection for higher speed data transfer and ready integration into an existing IP network. The latter is the primary means of interacting with the SBC and Robostix I/O during development.

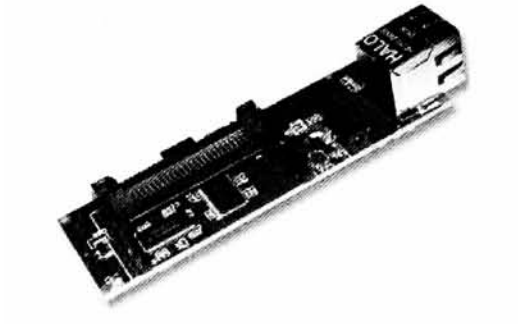


Figure 84: NetCF expansion board

Appendix B – Software Specification

Software Overview

A block diagram of the Gumstix system shown in Figure 85 depicts the relationships between the software and hardware components.

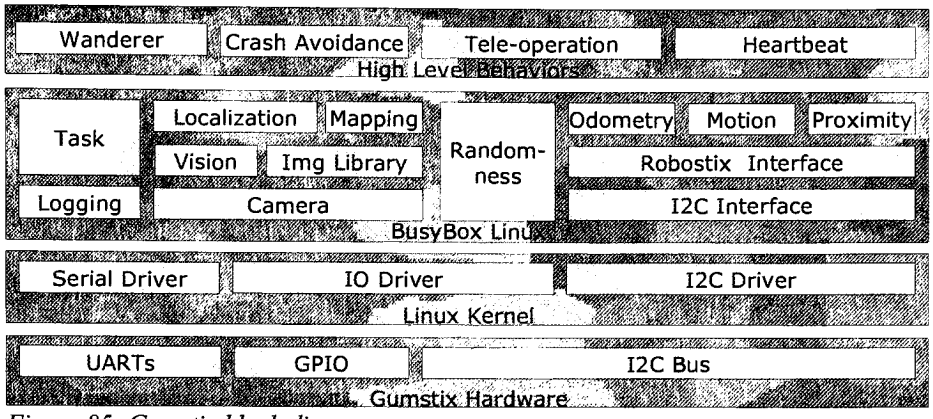


Figure 85: Gumstix block diagram

A similar diagram of the Robostix I/O hardware and related software components is shown in Figures 86.

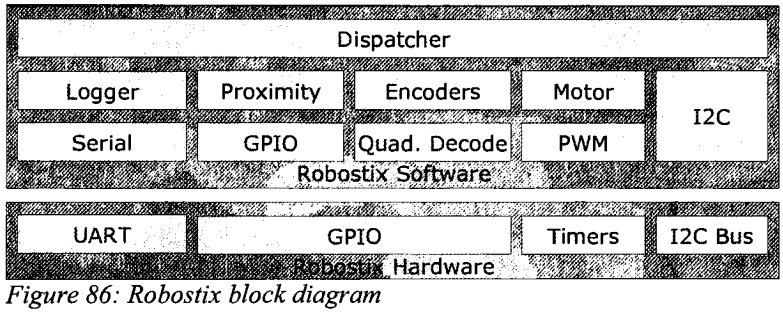


Figure 86: Robostix block diagram

Architecture

The control architecture is the foundation of most any mobile robot system. It provides a framework for building functionality and a set of rules to which all parts of the system subscribe. The mobile robot system developed is loosely based on the hybrid control architecture. The hybrid model allows responsiveness and high level planning (corresponding to the reactive and deliberative paradigms, respectively) to peacefully coexist. This affords a variety of well understood strategies for behavioral integration based on command arbitration. The proposed system employs a variety of behaviors to create a foundation from which higher level functionality may be implemented and studied. A simplified overview of the architecture is provided in Figure 87.

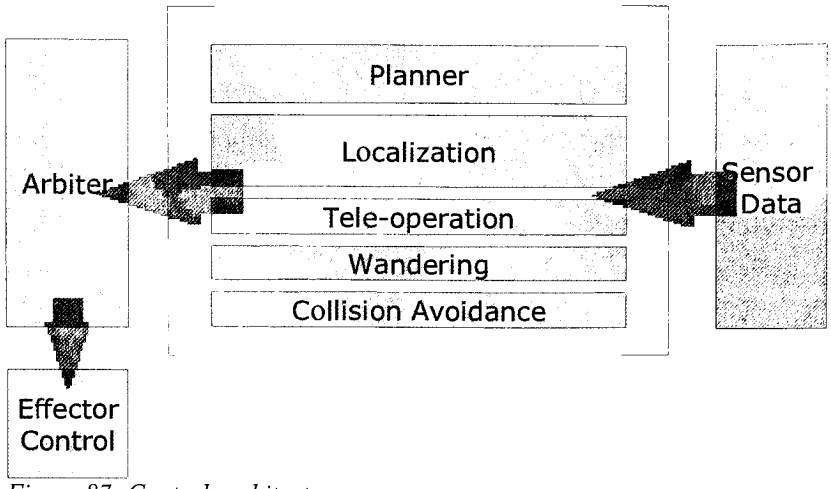


Figure 87: Control architecture

Each behavior is implemented as a separate task in the system. Sensor data from the infrared sensors as well as the vision module are available to all tasks in the system through a published sensor interface. Note the Arbiter itself also runs as a separate process. This maintains the abstraction that is sought between the behaviors, and allows those behaviors to interact with the arbiter in an asynchronous manner.

The Gumstix SBC is responsible for running the behaviors, all system level tasks described previously and most computationally intensive tasks. The CMUCam vision board is responsible for processing video images and extracting histogram, edge and other feature data as needed. The Robostix board will be programmed to manage only I/O related tasks.

Behaviors

Decomposing the system at the behavioral level grants a variety of benefits. Abstraction is implicit, as behaviors are running in parallel and are not concerned with the goings on of their peers. This allows each behavior to be implemented, debugged and tested independently.

Collision Avoidance

A behavior to protect the robot from damaging itself by colliding with various things in its environment is desirable. This will go a long way toward ensuring the robot survives through the various trials it will be subject to. This behavior is purely reactive, operating at a high frequency to ensure collisions are prevented. It is generally inactive, only providing direction when infrared sensor readings cross a programmed threshold.

Wandering

A wandering behavior is included to allow the robot to explore its environment. A variety of configurations may be used. Initially this was restricted to wall following. Higher level behaviors may or may not depend on this functionality.

Teleoperation

A remote control or teleoperation behavior was built in that operates on a strict priority based scheme with respect to the arbiter. When remote control is initiated, this behavior overrides all others in the system and grants control to the operator. Motor control, sensor data (IR, vision) and debug output are available through the remote interface.

Localization

The majority of the work in this research was carried out under the auspices of localization. As such the particulars are discussed in detail elsewhere. With odometry input from the motion API (below) and feature data from the vision turret, this behavior will make a position estimate available to all other tasks in the system.

Planner

The planner is the highest level behavior in the system. In theory it could be responsible for such macro level tasks such as chasing a ball, maintaining a formation or following a predetermined route. Planning was expected to be encoded in a simple hierarchy of finite state machines that dictate actions, or action plans. At any given time, one action plan would be selected, and has control of the robot. Localization experimentation would have been facilitated by an appropriate action plan or set of plans.

Framework

The more interesting of the framework components are described below.

Arbiter

The Arbiter is responsible for intelligently multiplexing requests for effector (motor, in this case) control. The initial implementation was restricted to a basic priority driven scheme, where motor control is acquired and released as a session.

Motion

The motion component abstracts motion specific work from the rest of the behaviors, accepting requests to move and notifying on completion of the appropriate tasks. This module is tied closely to the Arbiter to enforce serialization and integration of motion requests.

Vision

Algorithms are be required to distill features over and above those available in the CMUCam2+. These are used both to initially generate database images and to process new images at runtime after the initial localization implementation was completed.

Feature Database

The feature database is responsible for matching features in real time, given an algorithm to operate on. Note the underlying storage of the database must match the algorithm that is being used for matching. So entirely separate feature libraries must be maintained for each algorithm.

Appendix C – Graphical User Interface

The CMUCam2GUI is a Java front end for use with the CMUCam hardware. It affords visualization capabilities for most of the functionality that the CMUCam implements. The tool was extended to additionally:

- Accept and parse activity logs over TCP
- Extract particle information at three stages of the localization process and render in the context of a scale arena map in real time
- Provide histogram comparison for various algorithms
- Parse image libraries created during map generation
- Run sample images through histogram matching algorithms for every entry in the image library and generate a report
- Import raw images and processed image library entries
- Permit standalone/offline operation

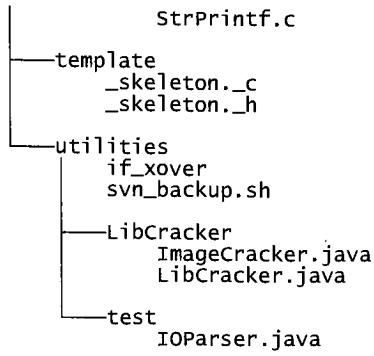
A screenshot depicting 500 particles before and after motion model application during a position tracking trial is shown below in Figure 88.

Figure 88: Real time particle rendering, extended CMUcam2GUI

Appendix D – Software Listing

```
Doxyfile
push

cmucam
    kermit.conf

    gui
        AboutWindow.java
        CameraImage.java
        CameraSerial.java
        CMUcam2GUI.java
        ColorTrack.java
        CommWindow.java
        guiPortSettings.stty
        Histogram.java
        Localization.java
        MainWindow.java
        Motion.java
        SerialComm.java
        serialPort.java
        sserial.dll
        Stats.java

gumstix
    camera.c
    camera.h
    common.h
    Config.h
    crash_avoidance.c
    crash_avoidance.h
    heartbeat.c
    heartbeat.h
    i2c-help.c
    i2c-help.h
    localize.c
    localize.h
    log.c
    log.h
    main.c
    Makefile
    map.c
    map.h
    motion.c
    motion.h
    rand.c
    rand.h
    redhat.mk
    rngs.c
    rngs.h
    robostix.c
    robostix.h
    shell.c
    shell.h
    task.c
    task.h
    vision.c
    vision.h
    wanderer.c
    wanderer.h

    common
        AvrInfo.c
        AvrInfo.h
        BootLoader-api.c
        BootLoader-api.h
        FILE_Data.c
        FILE_Data.h
        FILE_Parser.c
        FILE_Parser.h
```

```
        i2c-api.c
        i2c-api.h
        i2c-dev.h
        i2c-io-api.c
        i2c-io-api.h
        SerialLog.c
        SerialLog.h

├──robostix
    avr-mem.sh
    Config.h
    encoder.c
    encoder.h
    Hardware.c
    Hardware.h
    main.c
    Makefile
    motor.c
    motor.h
    rawio.h
    rs_commands.h
    Rules.mk
    serial.c
    serial.h
    svn-version.mk

    ├──common
        a2d.h
        a2d_8.c
        Args.c
        Args.h
        CBUF.h
        Delay.c
        i2c-master.c
        i2c-master.h
        i2c-slave-boot.c
        i2c-slave-boot.h
        i2c-slave.c
        i2c-slave.h
        lcd-hal-avr.c
        lcd-stdio.c
        memcpy_EP.S
        QD.c
        QD.h
        RCInput.c
        RCInput.h
        Robostix.h
        Servo.c
        Servo.h
        Timer.c
        Timer.h
        UART.c
        UART.h

    └──shared
        AvrInfo.c
        AvrInfo.h
        BootLoader.h
        Crc8.c
        Crc8.h
        Delay.h
        DumpMem.c
        DumpMem.h
        i2c-io.h
        i2c.h
        lcd-api.c
        lcd-hal.h
        lcd-printf.c
        lcd.h
        Log.c
        Log.h
        Str.h
```


StrPrintf.c
template
_skeleton._c
_skeleton._h
utilities
if_xover
svn_backup.sh
LibCracker
ImageCracker.java
LibCracker.java
test
IOParser.java