Methods for Intelligent Localization and Mapping during Haptic Exploration [*]

Monika A. Schaeffer
Department of Computer Science
The Johns Hopkins University
Baltimore, MD 21218 USA
monika@jhu.edu

Allison M. Okamura
Department of Mechanical Engineering
The Johns Hopkins University
Baltimore, MD 21218 USA
aokamura@jhu.edu

Abstract – *This paper presents a set of algorithms for use in simultaneous localization and mapping during haptic exploration. Several solutions are provided for the problem of a single spherical robot finger exploring a known smooth surface, starting with an unknown pose. Using pose estimates, pattern matching is performed between the robot's internal model of the surface and the known model. The robot finger is guided to explore regions of the surface that will maximize the probability of recognition. Simulation results demonstrate the effectiveness of one algorithm. In addition, it is shown that haptic exploration and dexterous manipulation can be achieved concurrently when multiple robot fingers are used.*

Keywords: mapping, probabilistic map, SLAM, CML, dexterous manipulation, haptic exploration

1 Introduction

In this paper, we develop methods for localization and mapping during exploration with robotic fingers. Haptic exploration and dexterous manipulation of unknown objects is important for several reasons. First, manipulation of objects whose shape is nominally known is difficult because of changing contact conditions and the resulting uncertainty of object pose within a multi-fingered grasp [13]. Information about surface geometry and other properties can be used in pattern matching to obtain improved pose estimates. Second, there is an interest in the development of a "haptic camera" [9] that can autonomously acquire a haptic model of an object. Robotic fingers, equipped with force and tactile sensors and the artificial intelligence to explore an entire object, can meet this goal. The captured haptic model can then be used in a haptic virtual environment or supervised telemanipulation. Third, while computer vision and laser range finding can acquire shape information, some object properties, such as friction and stiffness, require haptic exploration. These properties are important for manipulation and a complete haptic object model. The term "haptic" can include both tactile (cutaneous) and force (kinesthetic) information. Thus, haptic exploration or a haptic object model could include both types of information.

In this paper, we present algorithms for the case of a single spherical robot finger exploring a smooth surface. The shape of the surface being explored is known a priori, but its position and orientation relative to the robot finger is not. It is assumed that the robot finger can be controlled to move parallel to the surface using force or tactile sensing, as was experimentally demonstrated in [12]. The position of the center of the finger, rather than the contact point, is used for surface estimation, since it is inherently less noisy than tactile sensor data. However, tactile sensors could be used to acquire data to create a "haptic map" of the object surface.

To address this problem, we consider Simultaneous Localization and Mapping (SLAM), which is a collection of problems where an agent must simultaneously estimate the position of observed landmarks and itself while exploring and mapping an area. Previously, most of the research in this area has been for the application of safe, real-time obstacle avoidance and navigation of robots with vision and range sensors. There is a rich literature in the field of SLAM and pattern recognition, and relevant examples include [1, 11, 18, 19, 21]. The application of these methods to dexterous manipulation differs from the typical SLAM application in several ways. First, a common goal in SLAM is for mobile robot systems to avoid obstacles. In our case, we are not trying to avoid regions on the surface, but map an entire surface. Second, sensing occurs only locally with robotic fingers or "globally" using external sensors such as computer vision, whereas mobile robots typically use ultrasonic and other non-contact sensing methods that cover a small region of the workspace. Despite the differences between traditional SLAM applications and our research in haptic exploration, the goal remains the same – to efficiently place the robot and its observations on a comprehensive global map for immediate use.

[*] 0-7803-7952-7/03/$17.00 © 2003 IEEE.

2 Background

2.1 Simultaneous Localization and Mapping

SLAM in the context of mobile robot navigation has been a topic of great interest in robotics and artificial intelligence over the past decade. It is crucial in autonomous navigation for an agent to at least be able to track itself from a known starting position, and if possible, to be able to locate itself globally with no a priori knowledge of its position. SLAM has thus been studied extensively in the robotics field. A good overview is provided by the SLAM Summer School web site [17]. Many of the successful algorithms are in the form of Markov localization and other probabilistic algorithms, of which an overview is provided in [19]. Recently, algorithms have been developed to solve difficult problems such as tracking dynamic objects [6], and are used in applications such as vehicle navigation in urban areas [20].

In this paper, we apply some SLAM methods to the problem of haptic exploration. The basic algorithms we present do not differ greatly from algorithms already in use, but they are modified as appropriate for our application. We show that these methods can be adapted to the set of tasks and assumptions required for robotic exploration and manipulation of a known object starting in an unknown pose.

2.2 Haptic Exploration and Dexterous Manipulation

Dexterous manipulation is an area of robotics in which multiple manipulators, or fingers, cooperate to grasp and manipulate objects. Such systems become especially difficult to analyze and control when contacts are made and broken during the manipulation process (finger gaiting) and when rolling or sliding contacts occur. The field of dexterous manipulation is quite mature, and an overview is provided in [13]. For quasi-static manipulation, it is also necessary to maintain a stable grasp. Many grasp quality measures have been considered, e.g. [3, 7, 8, 22].

Over the last decade, there has been increasing interest in autonomous robotic systems that can explore unknown objects to determine their geometric and dynamic properties, e.g. [4, 9, 14, 15, 16]. However, none of these systems are able to automatically direct the motion of robotic fingers toward unexplored regions of an object surface, and none consider the problem of simultaneous exploration and manipulation.

We also note the role of tactile sensing for both manipulation and exploration. Tactile sensing provides the location of the contact between the robot finger and the object. Depending on the sensor configuration, it can also reveal properties about the object's surface properties, such as texture and friction. In this work, we assume that only the location of the center point of a spherical robotic finger is known, and that the finger must use this position only in the exploratory procedure. One reason for this assumption is that our future research will involve continuously rolling robotic fingertips, upon which it will be difficult to mount tactile sensors for contact localization.

3 Algorithms for Exploration

In this paper, we address the problem of a single robot finger, the agent, trying to locate itself on the surface of an object based completely on relative positions. The agent begins in a pose (position and orientation) that is unknown in the surface coordinates. This is often the case in dexterous manipulation, even if the object shape is known. The surface direction z (up, as in a Gauss frame) is assigned to match the global z. The agent is given a map of a subset of the object surface and asked to identify its location on the map. This corresponds to a dexterous manipulation problem where the initial pose of the object is uncertain, but a region of contact between the finger and the object is known or can be estimated. (If the robot finger is unable to locate itself on the local map given, other regions of the object surface can be used as the local map for haptic exploration.) We present four different algorithms that can be applied to this problem.

3.1 Algorithm I: A simple probabilistic map

The first algorithm we implemented is a simplified version of the position probability grid algorithm [2]. The agent maintains a probabilistic map of positions that correspond to locations on the map. When it takes a move, each point on the map spreads itself out to a circular distribution of points at about the distance the agent moved. Then points are weeded out if they are not likely candidates for the agent's position. Thus, the probability $p(l|t,r_t)$ of the agent being at location l at time t given the sensor reading r_t (this reading consists of the agent's internal x, y, and z coordinates, but the x and y are used primarily to verify that the agent moved as expected) is calculated by the update function

$$p(l|t,r_t) = p(l|r_t)\eta\sum_{l'} p(l'|t-1,r_{t-1}) \tag{1}$$

where l' is the set of all locations such that

$$||l'-l|-d|<d\varepsilon \tag{2}$$

ε is the factor of error the agent assumes it might make in his movement and d is the distance it believes it moved. Above, η is just a normalizing function so that the sum of all probabilities at time t is 1.

If we assume that the agent does not know its global z value, then the term $p(l|r)$ is more difficult to calculate. The function as a whole becomes

$$p(l|t,r_t) = \eta\Sigma_{l'} p(l'|t-1,r_{t-1})p(l|l',r_t-r_{t-1}) \quad (3)$$

and $p(l|l',r_t-r_{t-1})$ must be calculated for each l'.

3.2 Algorithm II: Pattern matching, combined with a probabilistic map

The above implementation has its weaknesses, most notably that it pays no attention to previously collected data. This Markov assumption is not appropriate here because the algorithm does not remember the agent's heading, and position alone is not sufficient. The agent could take two steps in a straight line and consider the place it started because there is no information to tell it about the history of its orientation. Because of this, the agent would often find the correct solution and then later jump to incorrect solutions.

One way to fix these problems is to integrate a pattern-matching algorithm into the previous algorithm. To do this, the agent maintains an internal map of all the data it has collected. Using the probabilistic map it generated, starting at the most probable point, it attempts to match its data to the copy of the map. Since the probabilistic map does not contain any directional data, it rotates the data set until it best matches the surface. If it cannot find a good solution, it removes that point from the probabilistic map. The algorithm stops when one location clearly dominates the rest and it finds a good solution at that location. This algorithm can be further optimized by allowing the agent to "predict" the next measurement, so it only needs to update itself when the prediction is wrong.

3.3 Algorithm III: Process of elimination

The two previous algorithms still ignore a main source of error: the agent does not associate direction with its probability-location data.

One simple solution is to create a table listing every pose (position and direction) and removing poses that are not possible. The calculations are simple – at each step, the agent "moves" all the points on the list and then compares its height to the height it would be at each pose. It deletes those poses that cannot fit, and works its way down to one single entry. An entry in the list might be π_{nt} = [global x, global y, angle of internal x-axis to global x-axis, difference between global z and internal z values], where π_{nt} represents the n^{th} pose in the global frame in the list at time t.

The agent then takes a step and updates to π_{nt+1}. Global x and y change based on the internal changes and the angle of the internal x-axis. (The difference in z values remains the same, unless the implementation allows for the agent to shift around his internal map, in which case, the value for all π change by the same amount.) The agent then calculates for each π_{nt+1} the expected z value reading, and removes π from the list if the expected reading is different from the actual reading.

Note that it does not make sense to work probabilities directly into this algorithm, since the calculated probabilities would tend to be near $1/n$ or 0, where n is the number of points whose probabilities are not 0. Thus, the calculation is not particularly helpful in this case.

This algorithm works surprisingly well, despite some major flaws. It of course assumes that there are a discrete number of locations and directions that can make a discrete number of starting poses for it to work with. (For example, in our standard setup explained later, there were 324 starting locations and 11 possible headings, for a total of 3564 poses). As maps get larger, these numbers will increase. Even with steps taken in the program to restrict the amount of calculation times at each step, the agent is not practical in large maps and some real world situations.

3.4 Algorithm IV: Combining evolution, probability, and process of elimination

This algorithm is an adaptation and extension of the MCL algorithm explained in [5]. It eliminates the problems of assuming a discrete number of starting poses and dealing with too much data at the beginning.

The agent starts by randomly selecting a starting sample of poses to serve as its candidate pool. It should pick no more than can be reasonably handled due to computational and storage limitations, but enough so that there is a good sample of all possible poses. These poses are also given an associated probability, which starts out as $1/n$, where n is the number of chosen points. Alternatively, each point could be given a measure of goodness, a heuristic acts as an unnormalized probability.

In this algorithm, each pose represents a range of poses. When the agent moves, it updates the given poses, but when it calculates how likely they are to be correct, it considers the entire range. In this case, the agent will actually make probability calculations based on how likely it is to get its readings if it is in that range:

$$p(\pi_{nt}|t) = p(\pi_{nt} \pm \delta t) = p(\pi_{nt} \pm \delta\pi_{nt-1})\, p(\pi_{nt-1}|t-1) \quad (4)$$

$p(\pi_{nt} \pm \varepsilon|\pi_{nt-1})$ is based on the differences in height and the height is stored in π_{nt}, so

$$p(\pi_{nt}|t) = p(\pi_{nt} \pm \varepsilon|r)p(\pi_{nt-1}|t-1) \qquad (5)$$

If we add a field to π to store $p(\pi_{nt-1}|t-1)$, we can now make the Markov assumption and ignore previous steps.

$$p(\pi_n|t) = p(\pi_n|r)p(\pi_n|t-1) \qquad (6)$$

At each turn, the agent discards a number of points that cannot work, based on some probability threshold. At this time, it selects a number of points that are promising and uses them to generate a new generation of points. Each member of the new generation could have a single parent, and its x and y positions and heading could be selected randomly from a normally distributed area around its parent's values. Children are added to the same candidate pool as their parents, and all candidates are expected to compete on equal footing regardless of age.

As time increases, the range each point represents slowly contracts, the distribution of change in each generation is lowered, and the allowed size of the candidate pool shrinks until the agent is left considering small variations on a single point. When the agent has found a perfect fit, the algorithm stops. This algorithm is efficient, since the calculation time at each step can be tightly controlled. There is never any lengthy list upon which calculations need to be performed, but it still considers more points than the previous algorithm.

4 Implementation

For this study, we implemented Algorithm III for a simulated robot finger in Matlab. In the current implementation, the agent stores a complete list of all the possible poses and works through a process of elimination. Steps are taken to cap the running time. By restricting the information that the robot functions can access, we separated the "real world," which modeled the world the agent was in, but could not directly access, from the agent's internal model.

We begin by creating a map of the object surface. The size and shape of the map are controlled by user-defined variables and its resolution is 10 discrete points per unit length. For our experiments, we primarily used a 20x20 map based on the "peaks" function built into Matlab. Occasionally, the map is generated with repeating parts or extended plains. The agent is placed on the map, but not told its pose. (We use 11 discrete directions to simplify the probabilistic map. This is an assumption of Algorithm III and would not be necessary for Algorithm IV.) The agent is given a copy of the map, and the goal of determining its pose with respect to the map.

The robot finger can sense with high accuracy the position of its center relative to where it started and what direction it was facing. This corresponds to the case of an actual robot finger, where joint sensors would be used to determine the finger position. We assume that the finger can roll along the surface. We do not slide the finger since it is very difficult to control sliding and manipulation simultaneously. However, the finger does not know the contact location on its own surface, and thus, exactly where the surface contact point is. This is a primary source of error.

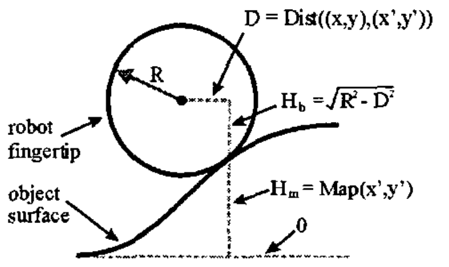


Figure 1. Estimation of the position of the center point of a spherical robot fingertip due to interference with the object surface.

4.1 Details of Calculations

In our implementation, there are some lengthy pre-calculations to be completed before the robot finger begins exploring. These calculations only need to be done once, and thus would not affect controller performance. The agent creates a list of all the possible initial poses. This is a large number, of the order of 11*Mapsize^2. (Implementation of Algorithm IV would fix this and the need for a discrete number of starting poses.) The agent then calculates the height it would be at each position, and saves it with the list as a reference.

As previously mentioned, the robot finger can only sense the location of its center. In our simulation, we must determine the actual contact between the surface and the finger. This is an expensive calculation, but when the exploratory procedure is implemented on an actual robot, this step is not necessary. In that case, the z position of the finger center is forced by physical interference between the finger and the object surface. For the purpose of simulation, we calculate the agent's minimum z position as shown in Figure 1. Since the agent cannot be below this point without cutting into the surface, the maximum is taken over all (x', y') points in the circle that cut the agent through the center parallel to the x-y plane.

$$z = Max_{x'y'}(H_m + H_b) \qquad (7)$$

where H_m and H_b are the heights of the map and the ball.

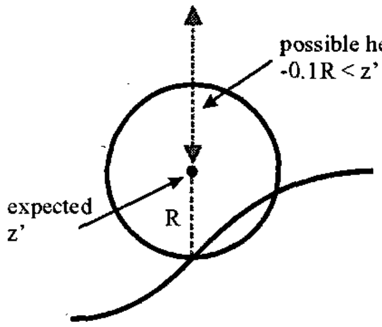


Figure 2. Estimation of the position of the center point of a spherical robot fingertip due to interference with the object surface.

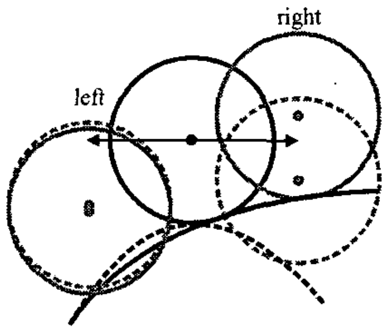


Figure 3. If the robot finger needs to test two possible poses (dashed and solid), it will move right in order to maximize the difference in the resulting pose.

4.2 Exploration Behavior

The agent is now free to attempt to determine its pose. It creates a new internal map with data points at every integer location with respect to the starting pose. The agent marks each discrete position visited with its height, and positions not yet explored with non-positive values. The agent expands the map to accommodate the spaces already explored. The agent also maintains a list of possible poses. After each move, the list is updated to represent where the agent would be if the previous pose was correct and that movement was made. The impossible poses are then filtered out.

Movement is controlled by a navigation function that considers the four compass directions, where north is a single step in its internal x direction. After each move, the agent needs to update the list of possible poses. This is done in two sweeps, a first, inexpensive test, and a second, more accurate test. The first test simply considers the height of the map of the map at the center of the agent's position. Heights that fall out of region of –0.1*R to 3*R are deleted from the list as shown in Figure 2. This is problematic with very steep slopes, but we are considering objects that are easily manipulated and therefore do not have steeply sloped surfaces. (If steep slopes are a possibility, the agent could calculate the needed range in preprocessing.) The second round of eliminations is done only on the five poses most likely to fail the second test. This assures that the test runs in constant time at most, and the expensive test does not go to waste. In this test, the agent's height at the pose is exactly calculated using Equation (4). If the agent's height falls out of a very tight range of that predicted by the pose, the pose is eliminated from the candidate pool.

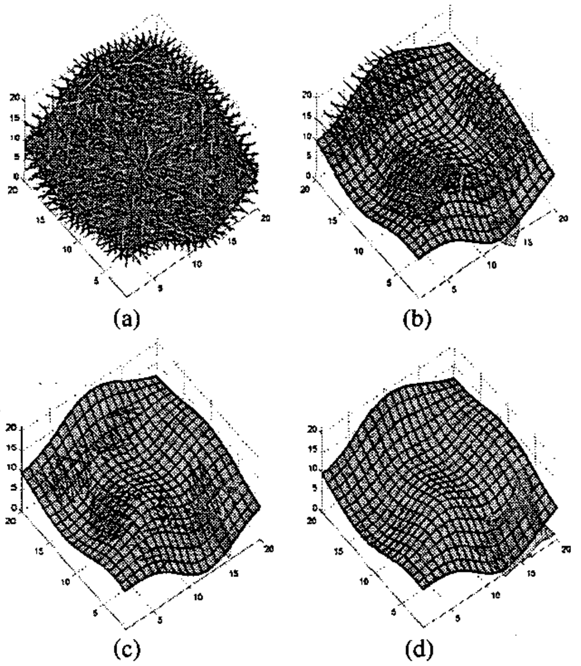


Figure 4. Results for exploration with a single finger. (a) t = 1, (b) t = 5, (c) t = 10, and (d) t = 23.

Since the agent's goal is to eliminate as many poses as possible, we wish to make moves that will allow it to learn the most about the environment. It would be expensive for the agent to calculate the exact amount of information gained from each move, so instead we perform a simpler calculation. 10 poses are picked at random and we obtain the z value of the map for a move from each pose in each compass direction. This creates 4 arrays of 10 values each. We compute the standard deviation of each of the arrays, and the agent moves in the direction that has the highest standard deviation (Figure 3).

4.3 A graphical example

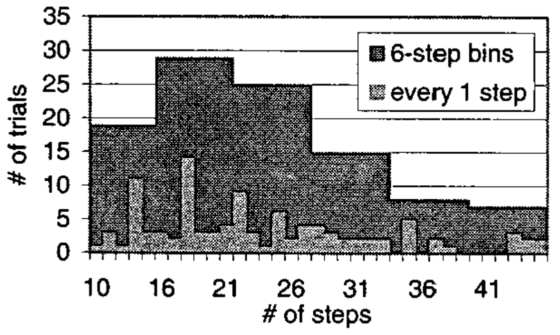
The four images shown in Figure 4 represent the agent at four different steps in the same run. Each line corresponds to an entry in the candidate pool, with one end marking where the agent thinks he is, and the other pointing in the direction of the x-axis. Map (a) is the starting condition, where the agent has to consider every point and every direction. The agent, shown as a ball, is hidden in the right near the wall. In maps (b) and (c), the agent has moved and narrowed down the choices. Remaining ambiguity is from similarities in different areas of the map. The agent has also experimented with placing previously collected data on the map, as shown by the mesh placed on the surface. By map (d), the agent has found its pose. This was done in 23 steps, and could have been faster if the agent had not started near the edge of the surface (a wall). For manipulation of 3D objects, there will be no such wall at the edge of a surface area under consideration.

5 Results

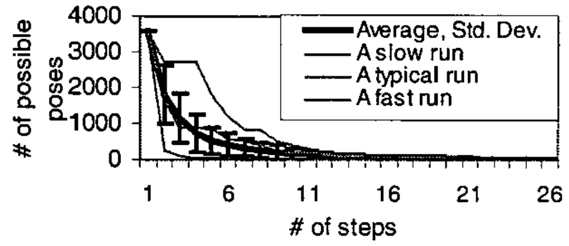
The current implementation of Algorithm III works well and could be implemented in real time for a robot working with small surfaces. The main problems are the discrete nature of the data and large set of candidate poses. Our proposed solution is Algorithm IV, which has not yet been implemented.

To test the algorithm, we ran the simulation on a 20x20 map from the built-in Matlab function "peaks," and with a robot fingertip radius of 0.8 units. Data was collected from 103 runs. We also ran simulations on larger versions of the same map for verification, and report some of that data separately. We observed the effect of other maps, but no data was collected. The maps were assumed to not have steep slopes or large plains. In all test runs, the agent never eliminated the correct pose from the candidate pool, and if the map had no large repetitious areas, the agent would always find the correct pose in a reasonable number of steps. On average, the agent needed 23.6 steps to determine its pose, with a standard deviation of 8.7 in a positive skewed distribution (Figure 5a). Most long runs were the result of the agent hitting walls and needing to backtrack. An informal test where the agent was placed close to the center showed that more than 17 steps were rarely needed. A more long-term decision-making algorithm could possibly improve this problem.

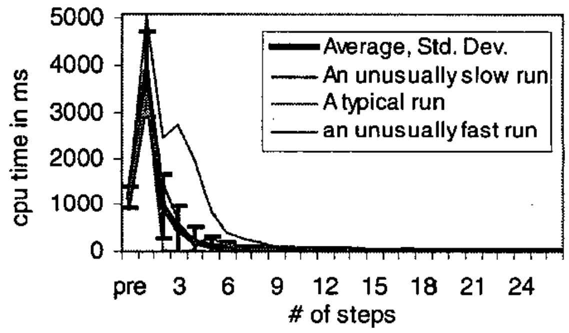
What was most important in determining how fast the algorithm performed was the dropoff rate, the rate at which the agent could eliminate choices (Figure 5b). In some cases, the first move eliminated as much as 94% of the original candidate pool. On average, about 50% of the choices were eliminated. In some starting poses, the agent would from the start run into the edge of the surface and be unable to eliminate much. Because we did not use the fact that the edges of the surface were the only obstacles that could stop the agent, reaching the edge of the surface did not provide any new information. We withheld this information in order to leave open the option of using an environment with unknown obstacles.



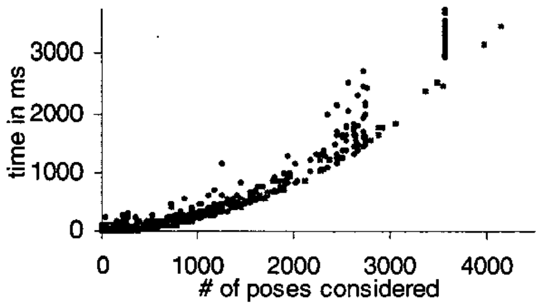
(a)



(b)



(c)



(d)

Figure 5. (a) Number of steps required for SLAM on a 20x20 map, over 103 runs. (b) Drop off rates. The agent was able to quickly reduce the large candidate pool. Plateaus in a run are caused by backtracking, which the agent avoided when possible. (c) Running time. (d) Correlation between run time and entries to process.

It is important that this algorithm be able to run in real time if it is to be implemented on an actual robot. We use CPU time as measured by Matlab as an indicator of runtime. (Figure 5c). This measures the actual time spent in the processor by the program, and is unaffected by other processes running on the computer. The code could certainly be optimized for speed and would run faster in C than in Matlab. Almost all of the running time is spent in processing after the first step. It is even slower than pre-processing. Averaged over all of the steps, the agent spent an average of 0.7 seconds per step. However, considering steps starting at step 5, the agent spent less than 0.04 seconds on the average step.

Figure 5d shows the correlation between the time it takes to complete a step and the number of entries in the candidate pool. The quadratic lower bound corresponds to the time it takes to perform the "quick and simple" sweep of the candidate list. Measurements above this are caused by the second sweep and the decision-making algorithm, which are both randomized and bounded by a constant. The lighter points are data collected from runs on larger maps, to show that this trend holds. It is impossible to do away with the first sweep, so the solution, as mentioned before, is to restrict the number of entries in the candidate pool. Again, Algorithm IV would not have this problem.

5.1 Summary of results

These experiments showed that Algorithm III could be implemented for a real-time robot under the assumptions of a small map with no steep slopes or drops, a discrete number of potential starting poses, and negligible error in sensor readings. The data shows that even with constant time per step caps on the more expensive tests, the cheaper tests force $\Omega(n^2)$ where n is the size of the candidate list, because of the cheaper tests that must be performed on every entry in the candidate list. However, because the algorithm is so quickly able to reduce the size of the candidate list, this was only a problem with the first few steps into the run.

The problems with Algorithm III basically imply a framework for implementing Algorithm IV. The agent should be able to work on arbitrarily large maps and recover nicely from sensor and control errors while at all times being able to strictly control the size of the candidate list.

6 Grasping and Manipulation

The current scheme includes only the goal of a single finger attempting to map a surface. Now we briefly consider that there are multiple fingers working together, contacting different locations on a three-dimensional rigid object, all trying simultaneously to develop a complete and accurate object model. We assume that the entire robot hand is controlled from one processor and the system is aware of each finger's pose and the current internal object map. The main problems are (1) maintaining a stable grasp during exploration (so as not to drop the object) and (2) manipulating the object to reach previously unexplored regions.

We can determine appropriate grasps using a simulator such as Miller's GraspIt! System [10], which can evaluate a grasp using a number of grasp quality measures. During manipulation,

$$\mathbf{f}_{obj} = G\,\mathbf{f}_{fing} \tag{8}$$

where G is the Gasp Jacobian (or grasp map), $\mathbf{f}_{fing}$ is the vector of forces applied by the fingers, and $\mathbf{f}_{obj}$ is the resulting force/torque vector applied to the object. A grasp quality measure can be applied to the grasp map. While each finger is performing exploration, an additional goal of the finger would be to prevent exploratory moves that lower the quality of the grasp. Or, fingers could simply be prevented from moving to regions of the workspace that create grasp quality measures below a certain threshold. We note that the entire geometry or current pose of the object does not need to be known in order to measure grasp quality. However, it is required that the contact points between the fingers and object be known, and that the forces applied by the fingers are sensed.

Manipulation may be necessary to change the pose of the object so the robot hand can explore regions of the surface that were unreachable in the initial grasp configuration. After each finger explores and attempts to recognize the portion of the object accessible in the current grasp configuration, it is possible that the entire object cannot be recognized because of repeating features or similarities between different objects. We assume that continuously rolling fingertips are employed, so that grasp gaiting is not required. We can consider two methods for manipulation. The first is simultaneous exploration and manipulation, where some fingers are no longer required to explore, and will simply move as required to maintain grasp quality. This will allow the fingers still exploring to have more freedom, and the object's pose within the grasp will change more rapidly as a result. Second, all fingers can stop exploration and the object's pose will be changed, with manipulation control based on a model determined from the initial exploration. There are many methods for manipulation control, which are outlined in [13].

7 Conclusions

This paper presented four approaches to adapting SLAM techniques to work for robot fingers with haptic sensors. One method, Algorithm III, was implemented and shown to be accurate and fast, despite being computationally expensive near the beginning of a run. Algorithm IV, not yet implemented, solves these problems, and is predicted to work even better when implemented on physical robot fingers running in real time. Areas for immediate future work are implementation of Algorithm IV, exploration of unknown shapes, a multi-fingered simulation, and implementation on a set of planar robot fingers, which are specially designed for dexterous haptic exploration (using continuously rolling fingertips).

Acknowledgments

The authors thank John Sheppard, Nat Duca, and other colleagues for their feedback on this paper. This work was supported by The Johns Hopkins University.

References

[1] K. Balakrishnan, B. Bousquet, and V. Honavar, "Spatial Learning and Localization in Animals: A Computational Model and its Implications for Mobile Robots," *Adaptive Behavior* 7(2), pp. 173-216, 2000.

[2] W. Burgard, D. Fox, D. Hennig, and T. Schmidt, "Estimating the absolute position of a mobile robot using position probability grids," *Proc. of the Thirteenth National Conference on Artificial Intelligence*, 1996.

[3] M. R. Cutkosky and R. D. Howe, "Human Grasp Choice and Robotic Grasp Analysis," in Dextrous Robot Hands, S. T. Venkataraman and T. Iberall, Ed., Springer-Verlag, pp. 5-31, 1990.

[4] P. Dario, P. Ferrante, G. Giacalone, L. Livaldi, B. Allotta, G. Buttazzo, A. M. Sabatini, "Planning And Executing Tactile Exploratory Procedures," *Proc. of the IEEE/RSJ Conference on Intelligent Robots and Systems*, Vol. 3, pp. 1896-1903, 1992.

[5] D. Fox, W. Burgard, F. Dellaert, and S. Thrun, "Monte carlo localization: Efficient position estimation for mobile robots," *Proc. of the National Conference on Artificial Intelligence (AAAI)*, Orlando, FL, 1999.

[6] D. Hähnel, R. Triebel, W. Burgard, and S. Thrun. "Map building with mobile robots in dynamic environments," *Proc. of the IEEE International Conference on Robotics and Automation*, Taipei, Taiwan, 2003.

[7] J. Kerr and B. Roth, "Analysis of multifngered hands," *International Journal of Robotics Research*, 4(4), pp. 3-17, 1986.

[8] Z. Li and S. S. Sastry, "Issues in dextrous robot hands," in Dextrous Robot Hands, S. T. Venkataraman and T. Iberall, Ed., Springer-Verlag, pp. 154-186, 1990.

[9] K. MacLean, "The 'Haptic Camera': A Technique for Characterizing and Playing Back Haptic Properties of Real Environments," *Proc. of the 5th Annual Symposium on Haptic Interfaces for Virtual Environments and Teleoperator Systems, ASME/IMECE*, DSC-Vol. 58, Atlanta, GA, November 1996.

[10] A. T. Miller and P. K. Allen. "GraspIt!: A Versatile Simulator for Grasp Analysis." *Proc. of the 9th Annual Symposium on Haptic Interfaces for Virtual Environments and Teleoperator Systems, ASME/IMECE*, DSC-Vol. Vol. 69-2, pp. 1251-1258, 2000.

[11] M. Montemerlo, S. Thrun, D. Koller and B. Weghreit, "FastSLAM: A Factored Solution to the Simultaneous Localization and Mapping Problem," *Proc. of the AAAI National Conference on Artificial Intelligence*, 2002.

[12] A. M. Okamura and M. R. Cutkosky, "Haptic Exploration of Fine Surface Features," *Proc. of the IEEE International Conference on Robotics and Automation*, pp. 2930-2936, 1999.

[13] A. M. Okamura, N. Smaby, and M. R. Cutkosky, "An Overview of Dexterous Manipulation," *Proc. of the IEEE International Conference on Robotics and Automation*, Vol. 1, pp. 255-262, 2000.

[14] A. M. Okamura and M. R. Cutkosky, "Feature-Guided Exploration with a Robotic Finger," *Proc. of the IEEE International Conference on Robotics and Automation*, Vol. 1, pp. 589-596, 2001.

[15] D. K. Pai, J. Lang, J. E. Lloyd, and R. J. Woodham. "ACME, A Telerobotic Active Measurement Facility," in *Experimental Robots VI, Lecture Notes in Control and Information Sciences*, Vol. 250, Springer-Verlag, pp. 391-400, 2000.

[16] M. Rucci and P. Dario, "Autonomous Learning of Tactile-Motor Coordination in Robotics," *Proc. of the IEEE Conference on Robotics and Automation*, Vol. 4, pp. 3230 -3236 , 1994.

[17] *Summer School on SLAM 2002*, http://www.cas.kth.se/SLAM/

[18] H. Tagare, D. McDermott, H. Xiao, "Visual Place Recognition for Autonomous Robots," *Proc. of the IEEE International Conference on Robotics and Automation*, pp. 2530-2535, 1998.

[19] S. Thrun, "Probablistic Algorithms in Robotics," *AI Magazine* 21(4), pp. 93-109, 2000.

[20] C.-C. Wang, C. Thorpe and S. Thrun, "Online Simultaneous Localization and Mapping with Detection and Tracking of Moving Objects: Theory and Results from a Ground Vehicle in Crowded Urban Areas," *Proc. of the IEEE International Conference on Robotics and Automation*, Taipei, Taiwan, 2003.

[21]S. B. Williams, "Efficient Solutions to Autonomous Mapping and Navigation Problems," Ph.D. Thesis, *Department of Mechanical and Mechatronic Engineering, The University of Sydney*, 2001.

[22] C. H. Xiong, Y. F. Li, H. Ding, and Y. L. Xiong, "On the Dynamic Stability of Grasping. *International Journal of Robotics Research*, Vo l. 18, No. 99, pp. 951-958, 1999.