Kaijen Hsiao
kjhsiao@mit.edu

Part A: Topics of Fascination

1) I am primarily interested in SLAM. I plan to do my project on an application of SLAM, and thus I am interested not only in the method we learned about in class, but also the method outlined in the DP-SLAM reading on the course webpage. I only want to implement one method, but I need to know how both methods work so that I can figure out which is easiest/best to implement. I will probably want to do my advanced lecture on a SLAM algorithm.

2) I am also interested in probabilistic roadmaps. We covered basic probabilistic roadmaps in class, which is something that I use in my own research, to control two simulated robotic arms. I am also interested in learning more about RRTs and MOPs, and would not mind doing an advanced lecture on a specific algorithm related to them.

3) Finally, as a topic I am very interested in learning about in class that hasn't been covered, I am particularly interested in MDPs and POMDPs. I have learned about HMMs, but not about POMDPs, and the information in Russell and Norvig is woefully inadequate.

Part D: Researching a Critical Reasoning Method
Paper 1: Thrun, Sebastian."A probabilistic online mapping algorithm for teams of mobile robots." International Journal of Robotics Research, 20(5):335-363, 2001.

The SLAM algorithm we learned about in class (Leonard's method of using multiple overlapping submaps with Kalman filters) is useful when errors in your robot's motion can be accurately represented using a Gaussian. However, for robots with nonlinear, inaccurate motion, using a Gaussian to represent the robot pose can be a poor approximation. Instead, one can use particle filters, which represent the robot pose essentially with a cloud of possible poses. Instead of a mean and

covariance matrix, the uncertainty in the robot pose can be captured by recording a large set of 'particles' that each contains one possible robot pose. Each particle is weighted by the probability that the robot is actually in that pose based on the latest observation. At each time step, the particles are sampled according to their probabilities, and the motion model is used to extend them according to the new motion input (with randomly sampled error included). Thus, the particle cloud expands as the robot moves, but a good observation will weight only those particles that are likely correct with a high weight, causing the cloud to shrink again. This is also a good method to represent branching hypotheses; for instance, if the robot could likely be in one of two separate places, the standard Kalman filter method cannot represent this fact with a Gaussian.

The idea of particle filters is the basis for a whole set of major SLAM algorithms, mostly developed by Sebastian Thrun. The reason I picked this paper is because, unlike later papers that all assume that you know all about particle filters and basic SLAM, it gives a coherent explanation that starts from the beginning. This includes what particle filters are, as well as how to do other low level details of the SLAM algorithm, such as how to compute the most likely robot pose given a map estimate (using gradient ascent), or how to represent the map as an occupancy grid whose squares reflect the probability that there is an obstacle there.

The actual system discussed in this paper is somewhat primitive compared to later particle-filter-based SLAM algorithms. In summary: the robot uses a laser rangefinder, and the map is represented using an occupancy grid rather than landmarks. However, only the maximum likelihood map is kept track of, rather than multiple map hypotheses, as we will see in later papers. The system is able to close loops, but it does so in one lump adjustment using a lot of gradient ascent once a conflict is found. The paper also discusses mapping with multiple robots that can communicate with each other constantly (one robot is the 'team leader', and new robots must be able to localize themselves in the team leader's map, at which point they start adding to the map just as the first robot does) and 3-D structural and texture mapping (their robot gets a panoramic camera; the 3-D info is not used in SLAM in any way, the 3-D images are just built using the SLAM results).

The algorithm used in this paper will probably not be used for my cognitive robot, but many of the low-level details from this paper will probably be used, and knowing this algorithm helps one to understand the algorithms in later papers. While it might be useful for multiple spy robots to be able to make the map together, it is not necessarily safe to assume that they can be in constant communication with each other.

Paper 2: M. Montemerlo, S. Thrun, D. Koller, and B. Wegbreit. "FastSLAM: A factored solution to the simultaneous localization and mapping problem." In Proceedings of the AAAI National Conference on Artificial Intelligence, Edmonton, Canada, 2002. AAAI.

FastSLAM appears to be a well-known, much-used algorithm, which is why I chose this paper. As before, a laser rangefinder is used to collect data, and the algorithm is based on particle filters. Unlike the last paper, landmarks are used rather than occupancy maps. Also unlike the last paper, each particle contains not only the robot pose, but also a Gaussian for each landmark in the map. Thus, each particle records one hypothesis for the map, and these can diverge widely if need be, rather than relying only on the one maximally likely map as in the last paper. Despite estimating the full posterior distribution over both the robot pose and all the landmark locations, the computation time scales logarithmically with the number of landmarks in the map (O(M logK), where M is the number of particles and K is the number of landmarks). This is not quite as good as Leonard's submap algorithm's time complexity of O(1), but it appears to be quite sufficient for quite large maps (with 1 million landmarks) even with only 100 particles, and was just as accurate as Leonard's method on the locations tried.

This paper also discusses the data association problem, which is the problem of deciding whether a landmark is the same as one you've seen before (and which one it is) or if it's a new one, which is another low-level detail not often covered in these papers.

For my cognitive robot, I would use a particle-filter algorithm, but may or may not want to use landmarks (vs. an occupancy map). If I were to use landmarks, this is probably the algorithm to use.

Paper 3: "DP-SLAM: Fast, Robust Simultaneous Localization and Mapping Without Predetermined Landmarks." Austin Eliazar and Ronald Parr, 18th International Joint Conference on Artificial Intelligence (Acapulco, August 2003).

This paper is on the reading list for the class, and thus it is the one I started with before realizing that I had no idea what particle filters were. However, after reading the first two papers (the algorithm in this paper is an extension of the work in the first two papers), it becomes much easier to understand.

The algorithm in this paper combines features of the first two papers. Like FastSLAM, it represents the entire map with each particle, and can thus keep track of hundreds of candidate maps. However, unlike FastSLAM, and like the first paper, it uses an occupancy grid as its map representation. As before, a laser rangefinder is used to collect data.

The main idea is that it finds a way to make copying of particles feasible when each particle has to keep track of an entire occupancy grid. If each new particle just copied over the old map and changed it, it would take O(MP) time, where M is the size of the grid and P is the number of particles. Instead, the algorithm presented keeps a particle ancestry tree, so that each particle does not have to contain the entire map. Each square of the occupancy grid keeps a separate tree that keeps track of the IDs of every particle that has made changes to the occupancy of that square, along with their observations. Each particle keeps track of its ancestors, and thus when the particle wants to find the value of a grid square, it checks each of its ancestors to find the most recent change to that square. If nobody has changed that square, the occupancy of that square is unknown.

The complexity of this algorithm can become somewhat bad (O(ADPlgP), where A is the area that the laser sweeps out, D is the depth of the ancestry tree, P is the number of particles, and M is the size of the occupancy grid), but in their experiments, the complexity was closer to A*lg(squared)P, which is < M.

If I wanted to use an occupancy grid rather than landmarks for my cognitive robot, this is probably the algorithm I would use.

Part E: A Simple Project For Your Cognitive Robot

My cognitive robot was a tiny spy robot (think cockroach-bot), capable of slipping in through a crack in a building and autonomously mapping the place while detecting motion and trying to avoid being seen. Even though I would do my project in simulation, the full capabilities of such a robot would be too much to implement.

Thus, here is a manageable project that embodies the salient features: a tiny simulated robot, driven by a human (to avoid having to develop a solid exploration function) to map the inside of a simulated building using either FastSLAM or DP-SLAM. Even ignoring the motion-detection-and-people-avoidance problem, this is already a significant project. This is because the tiny size of the robot requires the robot to do things a bit differently than most SLAM robots. The robot would have to be able to get over low thresholds and the like, and so a real-life tiny spy robot would probably have to be legged, likely giving it terrible odometry. (The simulated robot need not be legged; it just needs to approximate the movements of a legged robot.)

While no tiny laser rangefinders are currently available, it is quite conceivable that a tiny laser rangefinder could be developed. This is because the laser itself is just a tiny diode; most of the mechanism of a laser rangefinder is for precisely aiming the laser. If a MEMS device

with tiny mirrors could be built to do that job, a laser rangefinder could be made that was tiny enough to fit on a tiny cockroach-bot. However, the robot would have to creep along walls to avoid being seen, rather than charging down the middle of large corridors, possibly complicating the typical problem of exploring corridors and rooms.