PacSLAM

Arunkumar Byravan, Tanner Schmidt, Erzhuo Wang

Project Goals

The goal of this project was to tackle the simultaneous localization and mapping (SLAM) problem for mobile robots. Essentially, the SLAM problem is borrowed from the robotics domain, in which it is often applied to situations in which a mobile robot is operating in an unknown environment that it attempts to map by navigating and sensing obstacles using (generally) a laser range finder. The difficulty lies in that both the odometry and range finder readings are noisy.

We chose the Berkeley Pac-Man projects as the basic framework to test our solutions to SLAM. Additionally some tests were also carried out on real data collected from a four wheeled mobile robot equipped with the necessary sensors.

In order to solve SLAM in the Pac-Man domain, we had to first modify the framework such to fit the problem; i.e., by simulating a laser range finder and noisy odometry. We then implemented a variety of SLAM algorithms, modifying them slightly where appropriate in order to make them more suitable to the Pac-Man environment.

System Design

For the SLAM framework, the primary design goals were to integrate nicely with the existing Pac-Man framework, and also to provide easy extensibility and configurability. We ended up adding the following classes:

- *RangeFinder*:
  This class modelled a laser range finder. It is instantiated with three parameters: the standard deviation of the noise added to the distance reading of each ray, the field of view, and the number of rays. The rays are distributed evenly across the field of view. The class provides a method for getting the directions of every ray given the direction the range finder is facing, and a method that returns a noisy distance reading for each ray given the position and direction of the range finder and the location of all obstacles that block rays. This is stored in the Grid class used elsewhere in the Pac-Man framework, and in practice this was always just the wall locations, but it would be trivial to include ghosts or food in this obstacle matrix.
- *MotionModel*:
  This is essentially a data class which encodes the nature of the noise in Pac-Man’s motion model. It uses the Actions.Directions enum in order to integrate well, and for configurability it encodes the motion model as a dictionary that maps from intended actions to Counter objects which store a set of possible resulting actions and the probability that those actions will result from the intended action. The resulting actions

can even be lists of actions, allowing, for example, for Pac-Man to move forward two grid squares when attempting a forward motion of one square.

- *SlamAgent*:
  This follows the pattern of the other Pac-Man playing agents, implementing the necessary methods to allow you to play Pac-Man with a SLAM agent using "-p SlamAgent". On each time step, when the game queries the agent for its next action, it allows for specific implementations of the SLAM agent to update internal state based on current observations. For the actual implementation of various SLAM algorithms, we use a modular approach, and currently have three SLAM modules:
    - **dpslam**: DP-SLAM, as introduced in [1], refers to an efficient method of allowing each particle its own map by storing changes made by individual particles in a global map using balanced trees rooted at each grid square. The rest is standard SLAM techniques.
    - **fastslam**: FAST-SLAM, as introduced in [3], factors the full posterior over the pose of the robot and the surroundings into the product of a conditional distribution of the robot's pose and the locations of the landmarks (the environment is represented by a set of discrete landmarks). The conditional distributions are estimated with Extended Kalman Filter (EKF). The particle filter approach is employed to inference the posterior. Each particle stores the pose of the robot and the mapping of the landmarks. The complexity of FAST-SLAM is n(MN), where M is the number of partiles and N is the number of landmarks.
    - **gridslam**: GRID-SLAM, as introduced in [2], is a rao-blackwellized particle filter approach to solving SLAM using grid maps. Each particle has a map of its own and represents a possible trajectory for the robot. Two key improvements suggested are:
        - Accurate sampling of particles by combining information from the motion model and by matching observations
        - Adaptive re-sampling

      Due to the highly accurate sampling stage, GRID-SLAM generally uses an order of magnitude lesser number of particles compared to conventional SLAM algorithms. The velocity motion model from [4] was implemented as the odometry model for the robot.

GRID-SLAM was also implemented on real data collected from a mobile robot (data collected for ESE-650: Learning in Robotics course at the University of Pennsylvania). The robot was equipped with a Hokuyo Laser scanner, an IMU, Gyro and Wheel encoders. The test environments consisted of indoor hallways with ramps (upto 25 degree pitch). The Laser scanner was at a fixed height above the ground, pointing straight ahead with a FOV of 270 degrees. Some infrastructure from a previous project was re-used:

- *UKF for orientation tracking:*
  An Unscented Kalman filter was implemented for computing & tracking the 3D orientation of the robot given IMU and Gyro data. A few helper functions were also used
- *Scan Matching:*
  Code for matching a new observation (laser scan) to an existing map

Results

- DP-SLAM:
  We were able to successfully implement DP-SLAM in the Pac-Man framework. The accuracy of the maps generated depends on the number of particles, the noise in the range finder model, and the motion model. In testing, we used a motion model in which Pac-Man moved in his intended direction with 80% probability and did not move with 20% probability, and a typical range finder noise model had a standard deviation of 0.1 grid squares from the actual distance. Despite the relatively simple model, the algorithm required a somewhat surprising number of particles to work well; as you can see in figure 2, on very large maps, the map would deviate slightly from ground truth even with 25 particles. Of course, increasing the number of particles incurs a runtime penalty, which is characterized in figure 3.

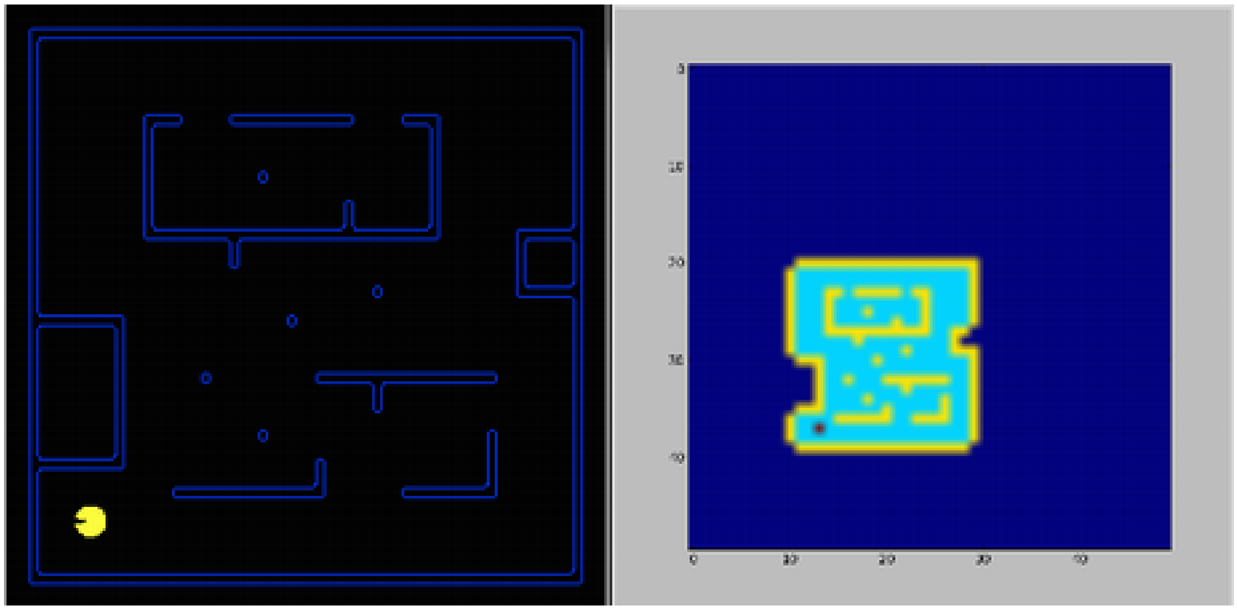

Figure 1: A typical display for PacSLAM. The ground-truth map and Pac-Man state are shown on left, and the agent's belief about the map and its position in the map are on the right. Unknown squares are shown in dark blue, free squares are light blue, occupied squares are yellow, and Pac-Man is the red dot.

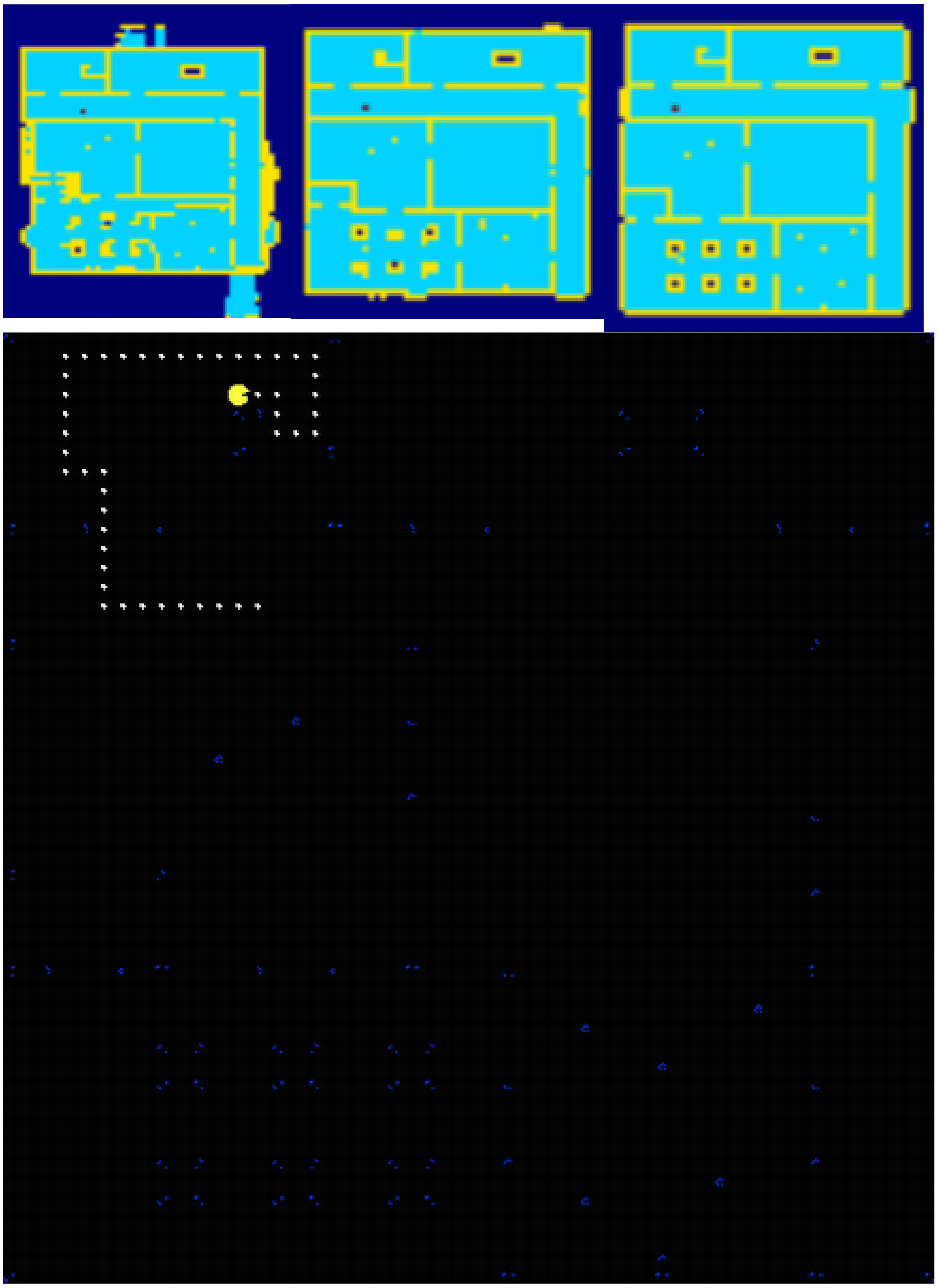
Figure 2: (top left) 5 particles, σ = 0.1 (top middle) 25 particles, σ = 0.1 (top right) 25 particles, σ = 0.025 (bottom) ground truth.

Figure 2 gives a good qualitative overview of the effect of the number of particles on the resulting maps. The top left shows what happens when there are an insufficient number of particles to deal with the noise in the readings. Comparing the top left and top middle maps shows how increasing the number of particles can drastically improve map quality, and comparing the top middle and top right maps shows how decreasing the level of noise in the range finder readings can further improve accuracy, yielding a nearly perfect map.

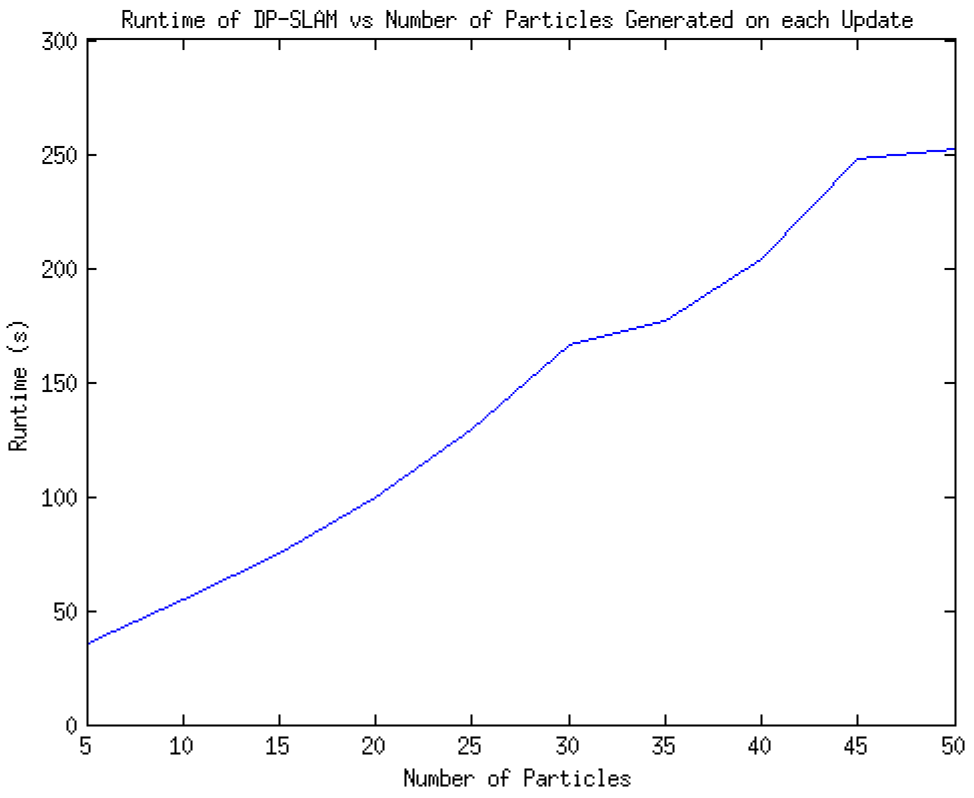

Figure 3: The effect of the number of particles used by DP-SLAM on runtime. Note that this plot lists the number of particles generated on each update, not the number of particles in use. The number of particles in use is variable, but when N particles are generated at each step, there are at most 2N-1 particles stored.

- FAST-SLAM

FAST-SLAM was implemented under the Pacman framework who is able to localize itself and map the surrounding, with noisy observation and a probabilistic motion model mentioned in the DP-SLAM part. The fastSlam Agent was tested on the openSlam. lay as in Figure 4.1, and bigTestSlam.lay as in Figure 5. For the smallTest in Figure 4, 15-20 particles are needed to obtain an acceptable map. The accuracy is evaluated using the Dice coefficient; the run time was recorded with a 3.1GHz CPU.

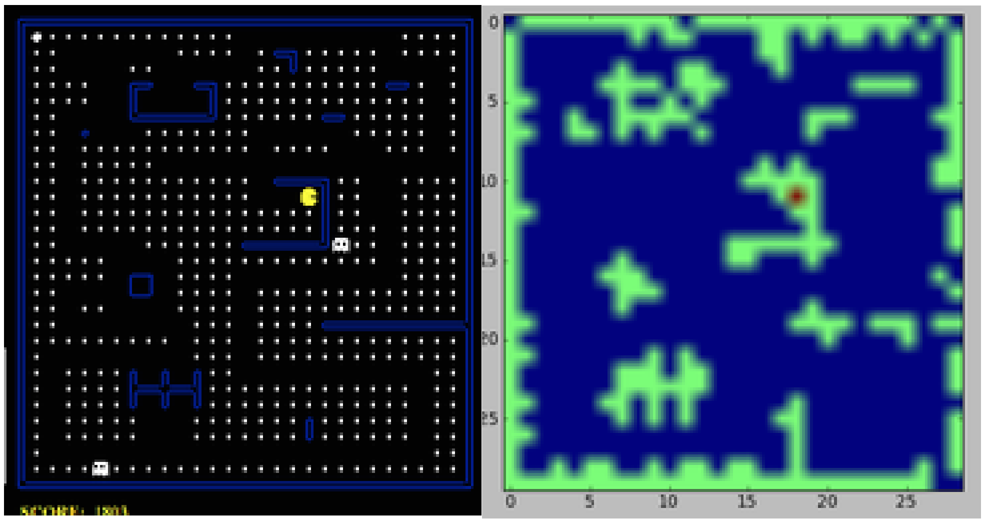

Figure 4.1Test result using openSlam.layout. 15 particles were used and accuracy 76%

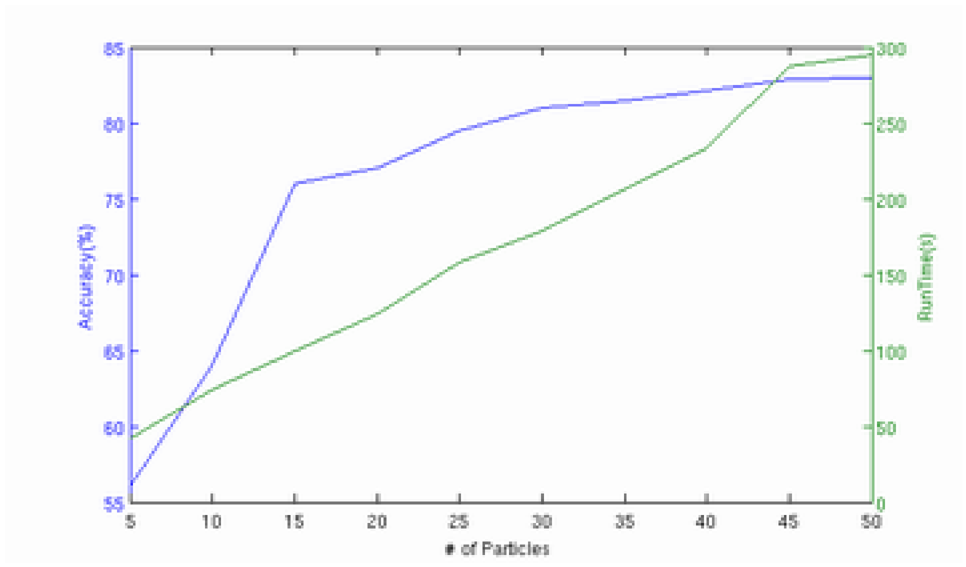

Figure 4.2: With the smallTestSlam layout shown in Figure 4.1, the impact of the number of particles on accuracy and and runtime

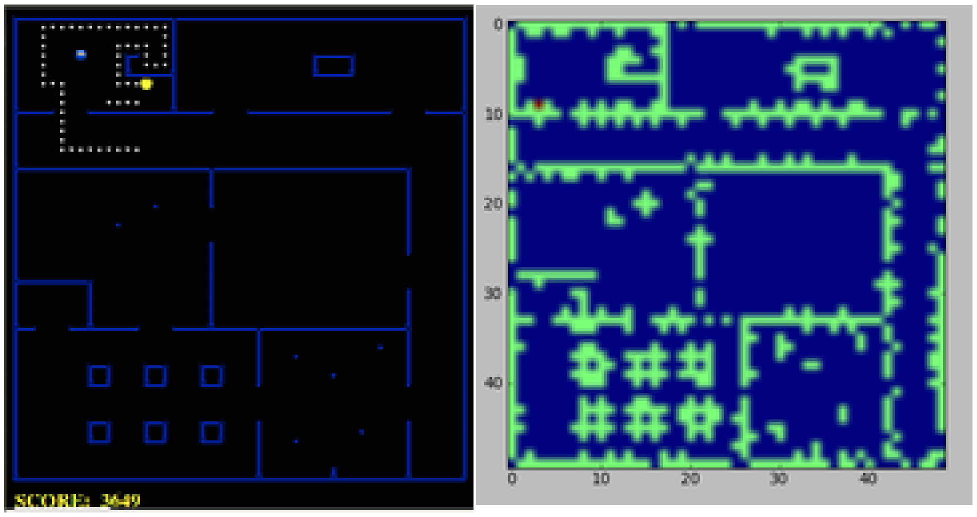

Figure 5. Test with bigTestSlam.lay. 15 particles, accuracy: 74%

Remark:

One of the innate drawback of the FAST-SLAM is the difficulty in resolving data association since it uses discrete set of landmarks to represent the surrounding. During the test, this disadvantage led to a result that the accuracy does not necessarily increase with higher resolution of observation. This is because if the resolution of observation is very high, the density of the measurement of the landmark will be highly overlapping, which increase the difficulty resolving the data association. This is shown in Figure 6

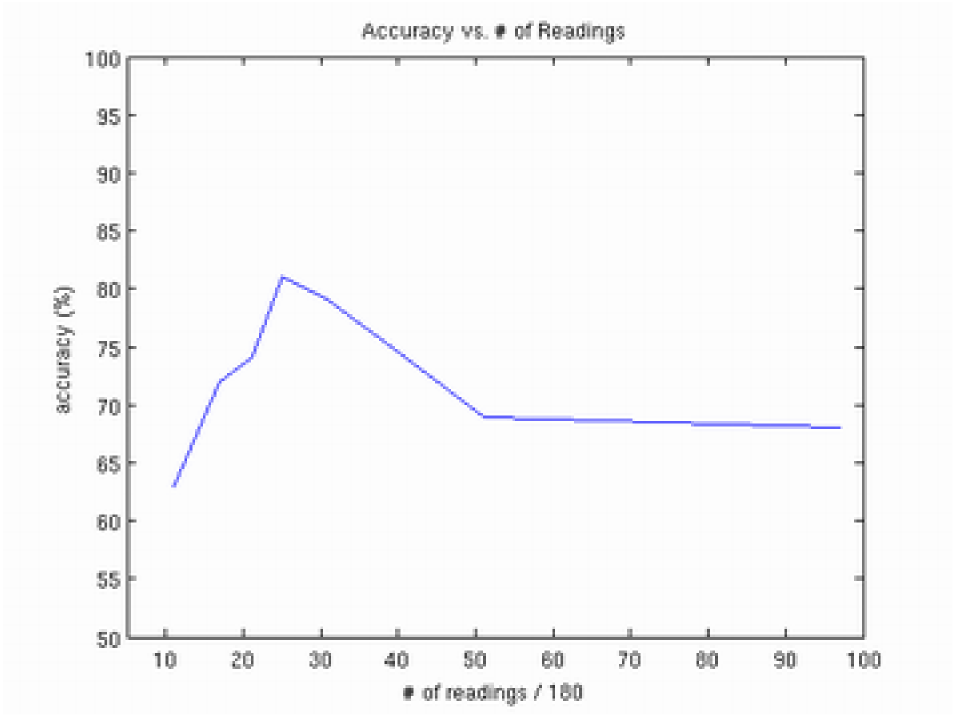

Figure 6. Using the openSlam.layout, 15 particles, the number of readings per 180 degree ranges from 11 to 97, which means a resolution ranging from 1.88 to 18. The accuracy reaches the peak at 25 readings per 180, then decrease, due to the data association problem

- GRID-SLAM:

  GRID-SLAM was successfully implemented in the Pac-Man framework and the real data as well. We tested the algorithm under similar conditions as DP-SLAM (on a similar map, with the same motion model & rangefinder noise). Due to the highly discrete nature of the Pac-Man world and known motion models, sampling particles by integrating information from both results in highly accurate tracking. Thus, GRID-SLAM can work well with just a single particle. Results on the environment from Figure 2 are shown below in Figure 6.

  Of course, if one did not assume knowledge of the motion model and assumed a uniform prior among the neighbouring states, many more particles would be needed for accurate results. When tested under such a scenario, GRID-SLAM fails due to the highly discrete nature of the environment, which causes multiple locations to look similar during scan-matching. A better way to solve the matching may improve results here.

  In case of real data, GRID-SLAM works reasonably to generate a map of the environment. More particles are needed here due to the continuous nature of the problem. Results on two datasets with 5 particles are shown in Figure 7. Increasing the number of particles should theoretically improve the accuracy of the estimation, though the maps look quite similar. Execution time is linear in the number of particles used.

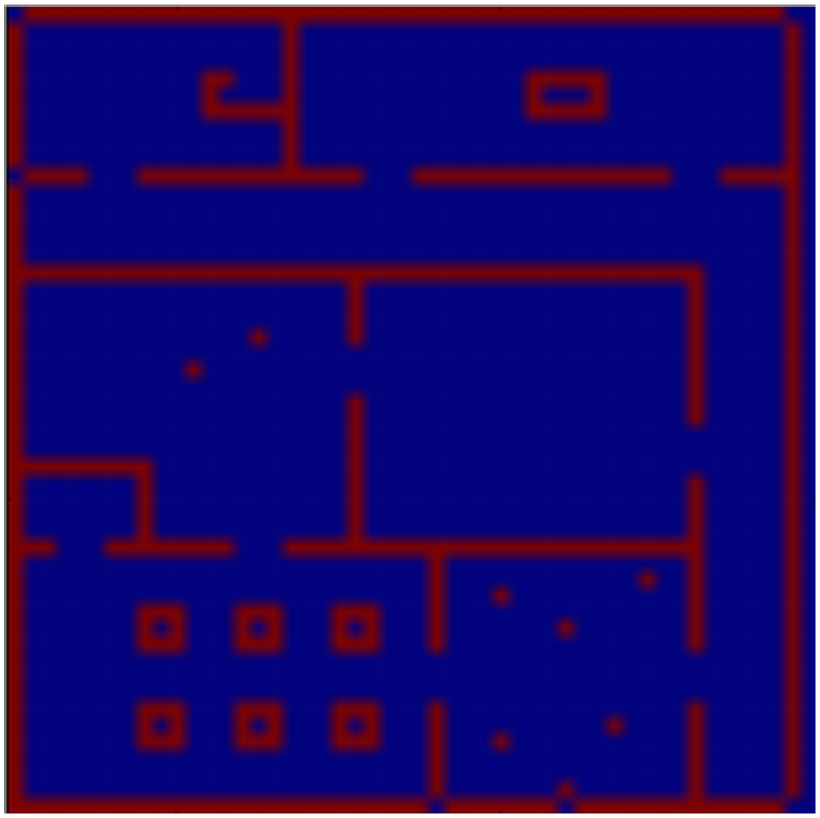

Figure 6: Grid-SLAM performance on the map from figure 2. Note the 100% accuracy.

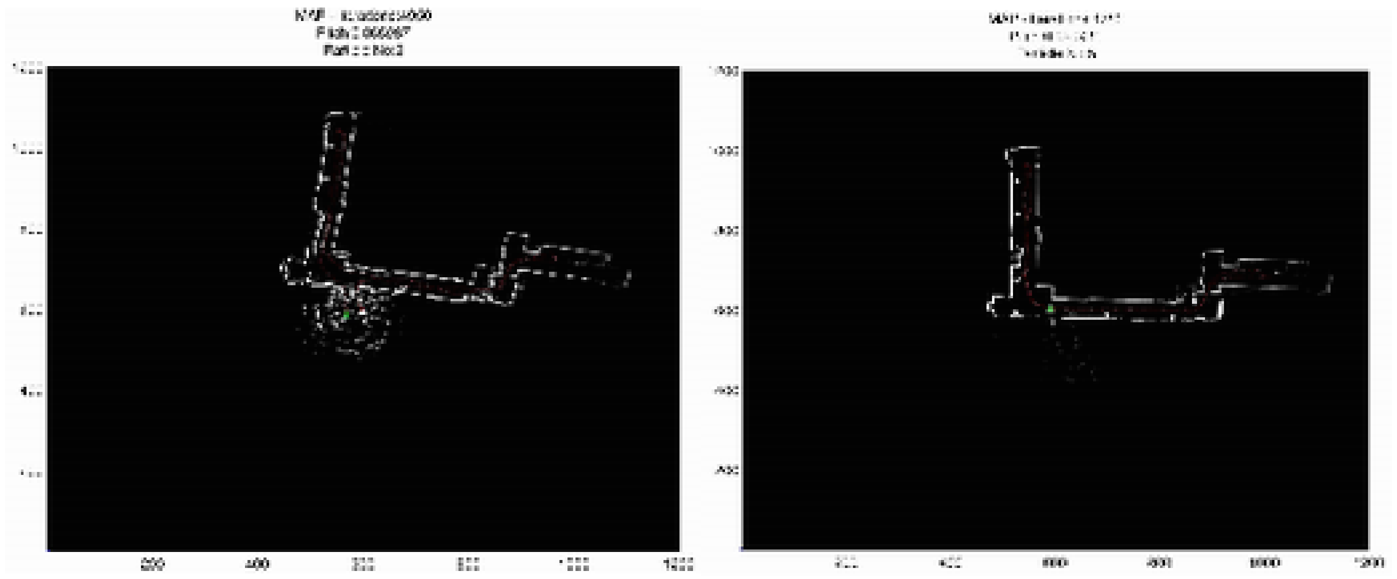

Figure 7: GRID-SLAM performance on two real world datasets. Results using 5 particles. Map created by particle with highest weight shown Environment on the right has ramps.

Surprises

Probably the biggest surprise when developing PacSLAM was the effect of the coarse discretization of the Pac-Man world. Whereas we initially expected this to make solving the SLAM problem easier, it actually made some things much more challenging. For example, the fact that the Pac-Man world is already a perfect grid would seem to be perfect for grid-based mapping approaches, but in fact, a noisy sensor reading of a wall will have only about a 50% probability of landing in the grid square that actually is the wall and about a 50% probability of landing in the grid square right in front of that (we accounted for this by computing the most likely generator of a noisy sensor reading based on both the position and direction of the reading). Also, it made multiple positions look similar from the scan matching perspective.

Another surprise was the fact that it is extremely important that you always have at least one particle that has the correct estimated pose and direction and a mostly correct estimated map. This is because, due to the high degree of uncertainty in the problem, the algorithms tended not to degrade gracefully but fail in rather extreme ways whenever a failure occurred. Thus, given a reasonable noise model, the number of particles required for DP-SLAM to maintain accurate tracking was often high enough to cause surprisingly slow performance.

In the case of the real world data, the biggest surprise was the effect of pitch on the map estimation (environment in figure 7). As the robot went up/came down the ramp, the rangefinder began to see ceilings and walls. As this looked similar to the wall at the ends of the hallway, scan-matching began to result in spurious position estimates which caused the algorithm to fail. To solve this, we decided to trust only the motion model estimate in case the pitch was beyond a certain threshold (0.2 radians). In practice, this resulted in much nicer looking maps and better tracking. Figure 8 shows a map with no pitch thresholding (compare with the map shown in Figure 7 - right). Another issue due to the presence of ramps was the fact that the motion was no longer planar. The velocity motion model in [4] assumes that the motion is planar. Incorporating the pitch into this was a challenge. While this works presently, it is to be seen how better we can improve the motion model in the future.

Conclusions and Future Work

Though we originally planned to do some comparison of the three algorithms implemented, this turned out to be more challenging than anticipated, for a number of reasons. First of all, evaluating map accuracy is quite tricky. One could imagine simply comparing the number of grid squares that are correct, but a common failure mode is for regions of the computed map to be shifted by one or two grid squares relative to another region, which would score poorly using such a metric despite being quite close to ground truth. Furthermore, the evaluation would depend on whether the goal was mapping, i.e. generating an accurate map, or localization, i.e. generating an accurate trajectory, or some weighted combination of the two. Also, the performance on each algorithm depends on a large number of factors, including the properties of the map, the FOV, number of rays, and noise level of the range finder, the nature of the noise in the motion model, and algorithm-specific parameters such as the number of particles used, etc. Therefore one algorithm may perform better than the others on some settings of the aforementioned parameters, but not on others. Finally, some of the algorithms turned out to be more amenable to the Pac-Man world than others, and nearly all were modified from their original forms to cope with the nature of the problem, so the comparison may not in fact be that useful. While we were thus unable to perform a quantitative analysis of the relative performance of the algorithms, we were able to characterize the individual algorithms as discussed in the results section.

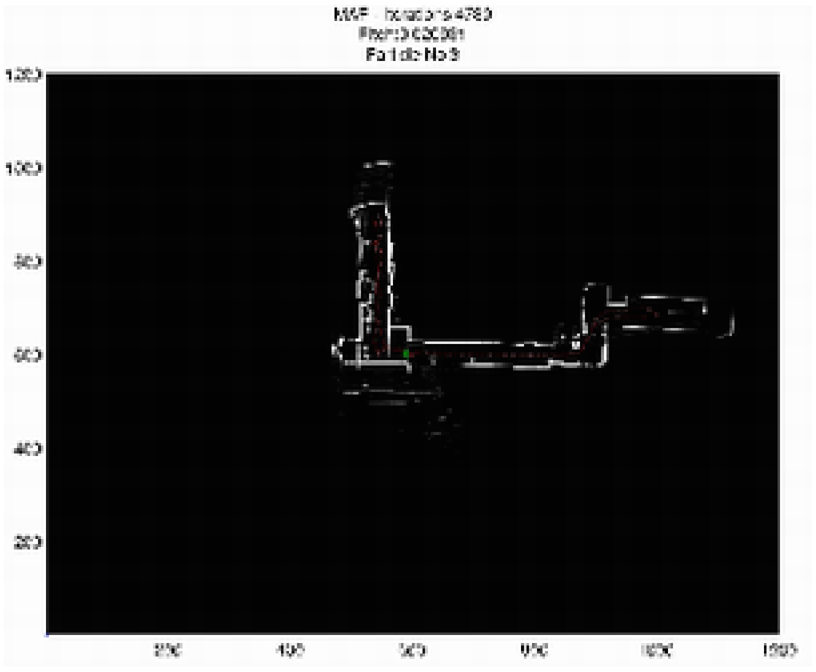

Figure 8: GRID-SLAM performance on the environment in Figure 7 - Right with no pitch thresholding. LIDAR scan matching is done all the time irrespective of robot's pitch. Robot underestimates distance to far wall on the center-left side of map due to a ramp's presence which causes false wall readings

As for future work, there are a number of interesting extensions that could readily be applied to the PacSLAM framework we developed, including:

- **Ghost Tracking**:
  Rather than just building empty maps, we could attempt to track ghosts using noisy range finder readings as well. This could be done somewhat trivially by having the range finder return the class of object that stopped the ray (where class $\in$ [ghost, wall]), but could be done more interestingly by not differentiating the two, and having particles include ghost positions along with Pac-Man positions and maps as. The difficulty would lie in distinguishing moving obstacles from walls that seem to move due to noisy readings, but by tracking through multiple time steps, it should be possible.
- **Automated Map Exploration**:
  Currently, Pac-Man can only build a map using human direction through the SlamKeyboardAgent. However, it would be interesting to formulate PacSLAM as a search problem in which Pac-Man attempts to map his entire environment subject to some evaluation function, perhaps on the amount of steps he takes, and have the agent plan accordingly.
- **Maximum Distance for Range Finder**:
  In the current implementation, the laser range finder model will return a noisy reading of any distance, no matter the magnitude. This is unrealistic, as in real-world scenarios, there is a maximum distance. It would be interesting to see what the effects of imposing a maximum distance reading would be in the Pac-Man world, effectively increasing ambiguity for situations such as wide open areas or long hallways.

For the real world data as well, there are a few interesting areas for improvement:

- **Loop Closure:**
  Currently, the implemented algorithm does not have any special cases to check for loop closure. Detecting when a loop closure happens and using the additional constraints to improve the map estimation should help improve the accuracy of the algorithm
- **Improved Motion model:**
  As mentioned before, improving the motion model to produce better estimates non-planar estimates of motion based on the pitch & roll will also improve performance
- **Ceiling/Floor detection:**
  A convenient way to detect the ceiling and floors when the robot is on a ramp would help us use the LIDAR data at all times, improving scan matching
- **Scan Matching:**
  The current scan matching algorithm matches the map to the observation given a list of probable positions and returns the one with the highest probability. Gradient descent along the observation likelihood coupled with raytracing the map to find free cells should

improve accuracy

References

[1] Austin Eliazar and Ronald Parr. 2003. DP-SLAM: fast, robust simultaneous localization and mapping without predetermined landmarks. In Proceedings of the 18th international joint conference on Artificial intelligence (IJCAI'03). Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1135-1142.

[2] Giorgio Grisetti, Cyrill Stachniss, Wolfram Burgard. 2007. Improved Techniques for Grid Mapping with Rao-Blackwellized Particle Filters. IEEE Transactions on Robotics.

[3] Michael Montemerlo, Sebastian Thrun, Daphne Koller and Ben Wegbreit. 2002. FastSLAM: a factored solution to the Simultaneous Localization and Mapping problem. In Proceedings of the AAAI National Conference on Artificial Intelligence.

[4] Sebastian Thrun, Wolfram Burgard, Dieter Fox. 2005. Probabilistic Robotics. MIT Press.

Appendices

Work split-up:

- GRID-SLAM on Pac-Man and real world data - Arunkumar Byravan
- DP-SLAM & PacSLAM framework - Tanner Schmidt
- FAST-SLAM on Pac-Man - Erzhuo Wang

Externally-written code:

The PacSLAM framework is based on the Pac-Man projects from UC Berkeley. Code written for ESE650 - Learning in Robotics course at the University of Pennsylvania (https://www.grasp.upenn.edu/courses/learning_robotics) has been re-used for implementing GRID-SLAM on the real world data. Details are explained in the report. The three SLAM algorithms are implemented based on ideas proposed in their respective papers as cited in this report. The red-black tree was implemented by borrowing from code at http://code.activestate.com/recipes/576817-red-black-tree/.

Running PacSLAM:

There are two implemented options for running PacSLAM. The first is by adding the argument "-p SlamKeyboardAgent" and the second is using "-p SlamTestAgent". The former allows you to control Pac-Man as normal as he maps, and the second will control Pac-Man such that he follows a trail of food particles. This is for use with maps “smallTestSlam.lay” and “bigTestSlam.lay”.

The slam module is set using “-a slamModule=[dpslam | fastslam | gridslam]”, and the motion model is set using “-a motionModelType=[DeterminisiticModel | NoisyStoppingModel | NoisyTurningModel]”.

Running GRID-SLAM on real data:

MATLAB code along with the datasets are attached in the submission. A README file explaining the options is also attached.