590DF Project 2

1. **Representing Robot Pose Uncertainty** Implement the sample-based method for representing the pose uncertainty of the robot. At a minimum, this means defining a class with the following functions and data: a) A set of poses that represent the $\langle x, y, \theta \rangle$ pose of the robot in the global workspace. Usually we'll work with 500 to 20K poses; it's nice to be able to initialize the pose set to an arbitrary number of poses. b) A function to initialize the poses to a uniform distribution within a given area, and another function to initialize them to a Gaussian distribution with independent mean and variance for each of the three pose dimensions. c) A function to move the poses according to the robot motion. This function should take as arguments the robot error characteristics, along with a distance, angle and rotation. It should update every pose in the pose set. d) A drawing function for showing the poses as a set of red dots on the Saphira window. To draw the pose cloud, the sample class should inherit from SfArtifact, and use the draw(SfWin *) function. The member functions for performing (b) and (c) should be made available in Colbert, so that you can experiment with them. To test your functions, try the following two scenarios. (1) Initialize the pose set to a uniform one within a small position window, but an arbitrary angle. Move the robot one meter forward, turn around 180 degrees, and move it one meter back. (2) Initialize the pose set to a single point, then move the robot forward 4 steps of 1 meter. How does the pose cloud evolve in each case? Do you understand why?

2. **Robot Pose Updating** Connect your sample functions to robot movement. Write a program that will update the point cloud according to the robot motion. Some things to note: a) Don't update every cycle, since it can be expensive. Wait until the robot has moved or turned enough, e.g., 200 mm and/or 10 degrees. b) You can figure out how far the robot has moved and turned, as well as the direction it has moved, by comparing the current robot pose to its pose at the end of the last update. c) Use an asynchronous task to do the heavy computation of the update. There should also be a microtask coordinating the update, that is, performing the checks mentioned in (a) and (b), and drawing the results.

3. **Robot Localization** This is the key part of the project. Here the update part of sample-based robot localization must be programmed. At each update of the particle filter, each sample has to be weighted with the likelihood of the sensor scan. Using the beam-based sensor model, this likelihood is given by the multiplication of the likelihoods $p(z|d)$ of all sonar measurements. To get $p(z|d)$, you must construct a sensor model for the sonars, where d is the distance to the next obstacle in the map given the sensor and the position of the robot (we provide functions to extract d for arbitrary robot positions and sonar sensors). The sensor model should distinguish two different cases. One in which the sensor beam hits the obstacle in an almost perpendicular angle, and one in which the beam is very likely to miss the obstacle. It is not necessary to obtain the sensor model experimentally: You can generate a mixture of a Gaussian and a uniform or exponential density with reasonable values, and you can adjust the parameters to get things to work correctly. Please plot $p(z|d)$ for values of z and d between 0 and 3 meters, as a 3D plot. Your plot should resemble the plots shown in class. Finally, integrate the

update part of the particle filter by using the motion model you programmed in part 1. Note that you can use sfROBOT→MoveTo to set the position of the robot. One additional problem is how to determine the current position of the robot based on the sample set.

To test this whole scheme, start with the robot well-localized in some part of the map (use a Gaussian initial distribution), and see if it stays localized as you move. The ultimate test for your localization will be global localization. Start with a uniform sample set over the space of the map, or some reasonable portion of the map. Move the robot around, periodically sampling the sonars and performing localization, until the robot becomes localized. You should move the robot using your collision avoidance method. Here, the initial sample set has to be quite large to cover the required space.