Particle Filter

Introduction:

Particle filters are useful tools when determining the general localization of a robot when there are elevated levels of uncertainty. In this part of the assignment, using a pre-created bag file, the program will generate a particle field for each velocity measurement and display that field using matplotlibcpp. The program was run on the same bag file three times and at each time the noise standard deviation was changed to determine the effect on the filter's spread.

Methodology:

The particle filter program is simple. On launch, the node will attempt to connect to the cmd_vel topic or will wait until this topic is present. Then, the node subscribes to cmd_vel and will continue to spin until it is closed by the user. The node consists of a velocity command callback method that processes the data from the topic and generates the particle field. The particle field is initialized to contain 100 points and the standard deviation of the noise is set to 0.1. For each velocity measurement, the linear and angular velocity is separated and for each particle (an object containing x and y values) in the cloud (a vector of the particle objects) the proportional x and y is calculated and set. Noise is then added to those x and y values in a random format (standard deviation * (random seed / random max + 1)). Still in the velocity command callback method and using matplotlibcpp, the particle points are displayed on a grid. This results in a series of grids that display continuously of the particles as the robot moves.

Effect of Noise Deviation:

To determine the effect of the standard deviation of the noise, the program was run three separate iterations at the follow values: 0.1, 0.5, and 0.75. It was clear that the greater the standard deviation, the farther the spread of the points. However, there was also an effect of where the particle cloud was displayed on the grid. Figures 1 and 2 show the particle cloud when the standard deviation was 0.1 at time 10 and 20. The points are all close together and move in a straight line. Figures 3 and 4 show the particle cloud when the standard deviation was 0.5 at time 10 and 20. The points are more spread out than Figure 1 and 2 and move more in more of a curve than Figure 1 and 2 as well. Finaly, Figures 5 and 6 show the particle filter when the standard deviation was 0.75 at time 10 and 20. Compared to Figure 3 and 4, there is a noticeable difference in spread and in position as the cloud at time 20 is at position 100 while in Figure 4 the cloud is closer to 125. The value of standard deviation is clearly important as it affects the certainty of the robot's position. Given additional information (values from sensors, loop closure, landmark detection) the value of the standard deviation should start higher and be adjusted based on the certainty that can be derived from that information.

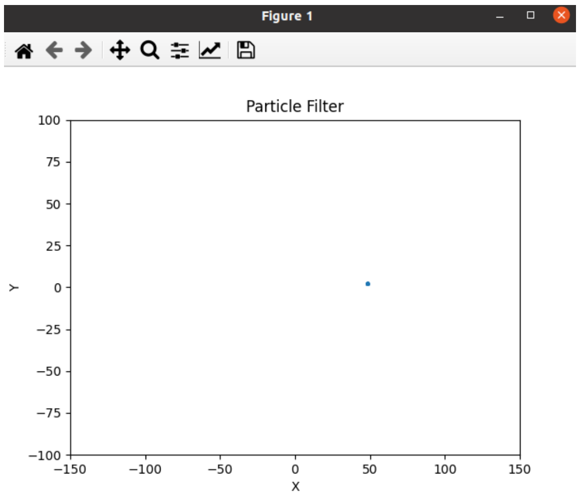

Figure 1: particle field with std=0.1 at 10s

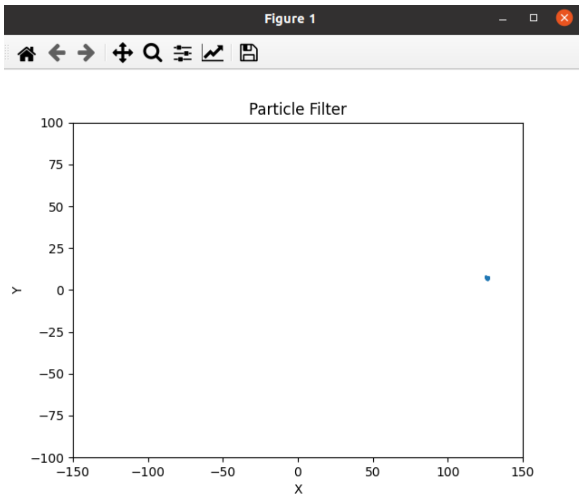

Figure 2: particle field with std=0.1 at 20s

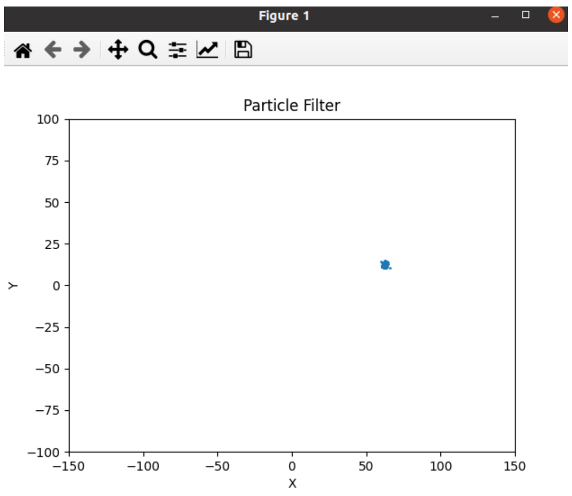

Figure 3: particle field with std=0.5 at 10s

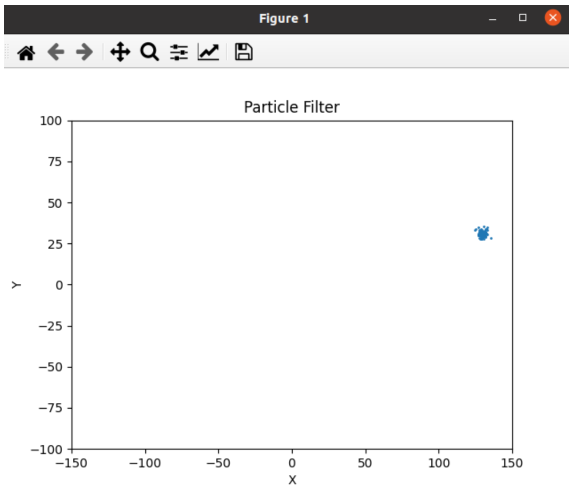

Figure 4: particle field with std=0.5 at 20s

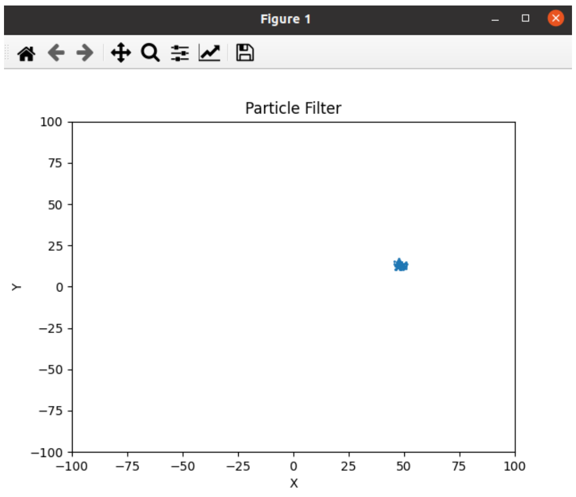

Figure 5: particle field with std=0.75 at 10s

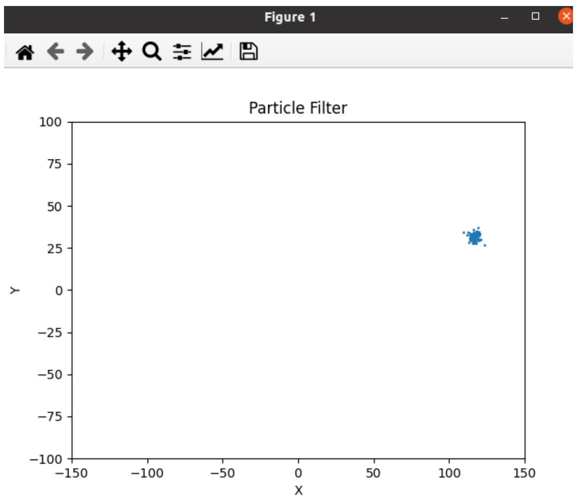

Figure 6: particle field with std=0.75 at 20s

Challenges:

The largest challenge I faced on this project was implementing matplotlibcpp. I chose to do the assignment in cpp as that is what I have worked with the entire semester, but that meant I had to implement matplotlib in a specific manner. I needed to add the h file to the implement directory and change the cmakelist file to include python and the include directory. After implementing these changes, I did not run into any other key issues, but this took me a bit of time to figure out.

Conclusion:

When exact localization is not possible based on prior knowledge or sensor accuracy, particle filters are an effective way to get a general idea of the location of the robot. The particle filter implemnted in this assignment uses just the cmd_vel topic derived from a bag file to populate a particle cloud and display it on a grid. Depending on the value set to the noise standard deviation, the general location of the cloud and its spread differs. In the future, I want to use a consistent display that maps all the clouds generated within a specific period.

Potential Field Follow the Leader

Introduction:

In the second assignment we were asked to implement a random walk, an algorithm that would have the robot randomly walk around a given environment while still able to avoid obstacles. The third assignment had us implement a potential field and help guide the robot to a specific goal while having attractive and repulsive forces helping the robot to avoid obstacles and eventually reach its desired goal. The final assignment has asked us to implement both concepts and have a system that generates a swarm of robots that follow a leader where the leader implements a random walk, and the swarm utilizes potential field.

Methodology:

I divided the potential field follow the leader into two parts: leader random walk and potential field swarm. The main idea behind the project is that the program takes in three parameters (the starting index of the robots, the number of robots, and the index at which the leader is). No matter which indexes the leader is set, that assigned robot will undergo a random walk and the other will undergo a potential field where the goal is the position of the leader. In theory, the swarm robots will head towards the leader utilizing a potential field to avoid the surrounding obstacles, the fellow swarm robots, and the leader.

To create a node that could simultaneously control the swarm robots and the leader robots, the pose and laser methods were created as structs and when called implemnted different actions accordingly. When the laser method was called, if the leader robot was the one calling the method, only the closest angle and range were set, and the laser method would act as if it would in a random walk. However, if a non-leader would call the laser method, it would set forcesx and y, sum_forces x and y, and several other variables exclusive to the potential field. However, the pose method worked the same regardless of the robot's identification. The publisher method is handled in a similar fashion and a vector of publishers (one for each robot) is created on launch.

While the node is running, it will continuously be updating the move command by looping through the indexes of the robots. If the robot is determined to be the leader, it will move forward until an object is detected within a certain distance and then rotate for 2 seconds. The robot will repeat this process until it is shut down by the user. If the robot is not the leader, the movement will be determined by the forces generated by the laser function. In theory, the closer to the leader the lesser the attraction force and farther from the leader the stronger the attraction force. The node can control the swarm and leader separately and the leader can reform a random walk while the swarm performs a potential field.

Challenges:

While in theory, since the goal of the swarm is the position where the leader is, the swarm should have a problem with colliding with the leader, it seems like the swarm struggles to abide by the attractive force. The swarm can avoid all obstacles (but the leader) with success; it seems as if the attractive force does not work. I have tried several things to get the attractive force working such as increasing gamma and limiting the sensor range (to decrease the repulsive forces), but neither have worked. What seemed to work the best is increasing the gamma to ten from five and inverting the force calculations from summing all of the forces to taking the difference of all the forces. This made it so the robots can follow the leader general, despite getting stuck occasionally within themselves or the wall. Figure 1 shows the robots on launch staying generally within the desired radius while Figure 2 shows the robots heading towards the leader after the leader collided with the wall while turning.

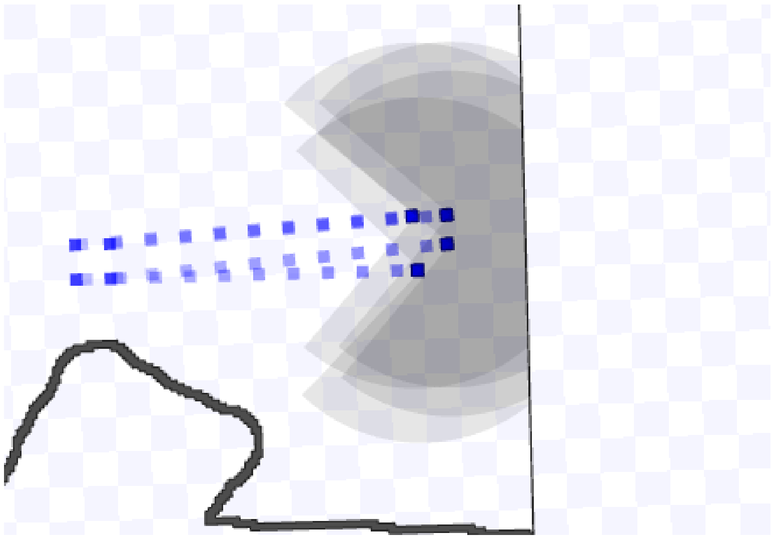

Figure 1: swarm following leader toward wall

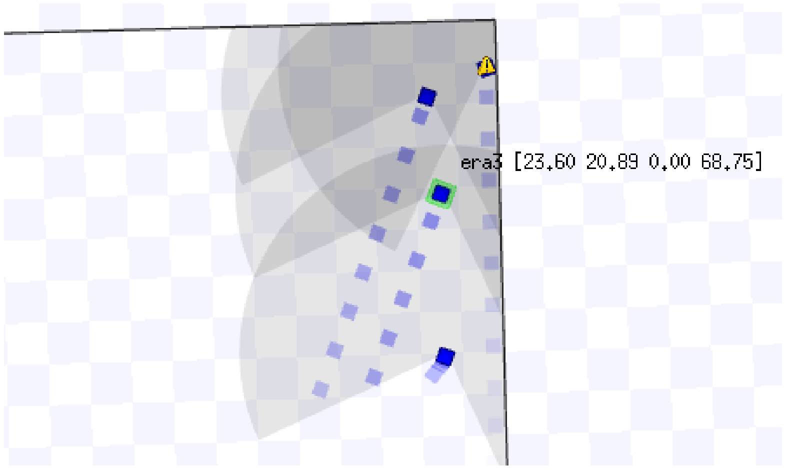

Figure 2: Swarm heading toward leader after slight collision

Conclusion:

While the swarm found some success following the leader, there are several improvements that can be made to this implementation. The leader successfully undergoes a random walk and the swarm attempts to follow the leader while avoiding obstacles. It should be noted that while the swarm moves towards the leader, it often gets out of range. One idea to fix this is to increase the speed of the swarm but also increase the repulsive force to avoid obstacles and fellow robots. While there are undoubtably improvements to be made, the general concept of the application is present. Moving forward, the repulsive and attractive forces need to be tuned in order to stay within a closer radius of the leader and the safe distance for the leader needs to be adjusted to ensure there is zero collision.