Simultaneous Localization and Mapping (SLAM)

We just spent some time talking about localization, where we know the map of the world that the robot is interacting with, but do not know where the robot is. But, what if we don’t know that, either? Now our sensor uncertainties are magnified.

Consider Figure 1. In this, we start by setting the robot down in the world, and turning it on. We then take an observation, and observe the green landmark. Now, we don’t know exactly where the green landmark is in reference to our robot, because our sensor is noisy. However, let’s say we can represent our uncertainty with the green cloud in Subfigure 1a.

Suppose we then turn and drive the robot. Now, our odometer is uncertain, so the location of our robot, in reference to the already-uncertain green landmark, has increased. This can be seen in Subfigure 1b.

We then observe the red landmark. Now, we already don’t know where we’re observing it from, because our position is uncertain. In addition, our sensor adds additional error onto the position of our landmark. This can be seen in Subfigure 1c.

So, is there no hope? Well, what if we turned again (increasing uncertainty in the robot’s pose again), and observed the green landmark once more? Now, we’re more sure about the landmark’s position than we are about the robot’s position. Doesn’t this observation then tell us which of the possible robot poses are most likely?

So, we can see that there is the possibility of reducing uncertainty with repeated observations. Because we are both determining what the map looks like, and determining the robot’s location in that map, this is known as Simultaneous Localization and Mapping, or SLAM.

In localization, we were interested in calculating $p(x_t | u_{1:t}, o_{1:t})$, where $x_t$ is the robot’s pose at time $t$. In SLAM, we’re calculating $p(x_t, m | u_{1:t}, o_{1:t})$, where $x_t$ is still the robot’s pose, and $m$ is the map. Notice the map is NOT timestep dependent. The difference, of course, is that the pose will change with time, while the map does not.

Particle Filters and SLAM

As with MCL, SLAM has a lot of different approaches, which make different assumptions. Also as with MCL, the approach that involves the fewest number of assumptions about our distribution is a particle filter. We’ll be talking about an oft-implemented version known as FastSLAM.
In MCL, a particle is a candidate robot pose. In SLAM, a single particle has to consist of both a pose and a map, where our map is represented by a 2D Gaussian for each landmark. We take a brief break to discuss Gaussians in two dimensions.

2D Gaussians: In 1D, the center of a Gaussian is defined by a mean $\mu$ and its shape by a standard deviation $\sigma$. In two dimensions, this is no different. However, the mean becomes a point in 2D space, and the shape is defined by a 2x2 covariance matrix $\Sigma$. The details of how this works isn’t too important for our purposes, but it’s worth seeing some examples:

- (a) $\mu = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$, $\Sigma = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$
- (b) $\mu = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$, $\Sigma = \begin{bmatrix} 5 & 0 \\ 0 & 1 \end{bmatrix}$
- (c) $\mu = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$, $\Sigma = \begin{bmatrix} 5 & 3 \\ 3 & 5 \end{bmatrix}$

Figure 2: Gaussians centered at the origin with a variety of covariance matrices. Notice there can be lots of different shapes attained by changing this matrix. The top row is the same as the bottom row, but viewed from above.

For an $n$-dimensional Gaussian, simply make $\mu$ a $n \times 1$ vector, and $\Sigma$ a $n \times n$ matrix. Notice that our 1D Gaussian is simply this, where $n = 1$.

Back to FastSLAM... So, each particle contains a 2D Gaussian for EACH landmark, indicating its understanding of where each particle is likely to be:

<table>
<thead>
<tr>
<th>Particle</th>
<th>Pose</th>
<th>Landmark 1</th>
<th>Landmark 2</th>
<th>...</th>
<th>Landmark N</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>$x_1 = (x, y, \theta)$</td>
<td>$\mu_1, \Sigma_1$</td>
<td>$\mu_2, \Sigma_2$</td>
<td>...</td>
<td>$\mu_N, \Sigma_N$</td>
</tr>
<tr>
<td>2</td>
<td>$x_1 = (x, y, \theta)$</td>
<td>$\mu_1, \Sigma_1$</td>
<td>$\mu_2, \Sigma_2$</td>
<td>...</td>
<td>$\mu_N, \Sigma_N$</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>M</td>
<td>$x_1 = (x, y, \theta)$</td>
<td>$\mu_1, \Sigma_1$</td>
<td>$\mu_2, \Sigma_2$</td>
<td>...</td>
<td>$\mu_N, \Sigma_N$</td>
</tr>
</tbody>
</table>

The steps of FastSLAM are very similar to the steps of MCL, and are shown in Algorithm 1. We will now discuss each of the steps in Algorithm 1 in more detail:

Moving the particle: The particle’s pose is moved just like in MCL. Given some desired motion $u$, and a motion model, each particle is given a different change in pose drawn from that motion model.

Predict the observations: This is extremely similar to what is done in MCL to provide our $o_p$s. The observation prediction comes from the particle’s belief as to the landmark’s location, and the robot pose.
**Algorithm 1** FastSLAM

for all \( p \) in particles do
    Move the particle according to the motion model and \( u \)
    Predict observations given particle pose and landmark locations
    Given your actual observations, Update \( \mu \) and \( \Sigma \) for each observed landmark
    Weight the particle according to likelihood of those observations given this particle’s predicted observations
end for
Resample particles using weights

**Update the landmarks:** We’re not going to focus much on how this math works. It is important, however, that we have some intuition into what will happen.

Suppose a particle’s belief on a landmark location is somewhat confident, and then the landmark is observed, once again, in exactly that location. This will cause the covariance around the \( \mu \) to tighten up, making the belief even more confident.

However, suppose the landmark is observed far away from the particle’s belief. In that case, \( \mu \) will move in the direction of the observation, and the covariance will increase to indicate decreased confidence.

The degree to which these effects happen is determined in large part by the sensor model, in that an observation from a well-trusted sensor has more effect than an observation from an often-wrong sensor.

Finally, note that if a landmark is NOT observed, there is no effect on the map, even if the landmark was right in front of the sensor, and should have been observed; this type of negative information is not considered by FastSLAM.

**Weighting the particle:** Again, this is very similar to MCL. Given the predicted observations, the actual observations, the sensor model, and the confidence in the landmark’s position (which is the only new part), what is the likelihood this pose and map is correct? This becomes the particle’s weight.

**Resampling:** This is done exactly like in MCL.

And that’s it! The maps for your second project were created using FastSLAM.