Chapter 1

FastSLAM

A Incorporating Observations

When incorporating observations, we must reweight the particles according to the observation probability. Note that in this section, since every distribution is implicitly conditioned on previous feature observations and robot commands, we’ll omit the $u$ and $z$ terms that might otherwise appear as conditional terms.

$$p(z|x)$$  

(1.1)

Here, we use $x$ to mean the trajectory of the robot, which is the quantity that a given particle is conditioned upon. It’s not immediately clear how to compute $p(z|x)$; it is not the same our observation model. Our observation model, $p(z|x, f)$, requires us to know where the feature $f$ is!

In our Rao-Blackwellized FastSLAM filter, our particles do not sample over the positions of the landmarks. Instead, each particle maintains an EKF that estimates the position of the landmark conditioned on the robot’s trajectory. (In other words, our EKF is estimating the quantity $p(f|x)$.)

Let us take a closer look at $p(z|x)$. We can re-introduce $f$ by “un-marginalizing” (note that this is always true; it doesn’t exploit any special property of our problem):

$$p(z|x) = \int p(z, f|x)df$$  

(1.2)

We can apply the product rule, which again is always true:

$$p(z|x) = \int p(z, f|x)df = \int p(z|f, x)p(f|x)df$$  

(1.3)

We’re now in good shape: the quantity we want is an integral of the product of two quantities that we know: $p(z|f, x)$ is our observation model and $p(f|x)$ is our estimate of the feature location prior to observing $z$. This equation makes good intuitive sense: we don’t know $f$ with certainty, so we sum over all possible values of $f$. Of course, the question is: can we compute this integral?

We could compute it numerically, easily enough. For simplicity, let’s consider a 2D robot whose location is given by $x$ and $y$, but no $\theta$. A real-world analog would be a robot with a perfect compass. Further, we’ll suppose that our robot’s sensor returns the relative location (in global coordinates) of the feature. These simplifications make the observation equations linear, which makes the math simpler. However, as you’ll see, we still have all the machinery needed to handle the non-linear case.
Specifically, assume that the state vector is $x$, and that the first two elements contain the position of the robot in the $x$ and $y$ directions. Let $\mu_f$ and $P_{ekf}$ be the state and covariance of the feature position as maintained by the EKF. Finally, let us suppose that the covariance associated with observations is $P_{obs}$.

When numerically performing the integration, we must limit the area over which we integrate; the range parameter controls this. It needs to be large enough so that we integrate over all of the high-probability areas. We must also choose how finely to sample points; this is controlled by step. Also note that since most programming languages can’t use the $\Sigma$ character, we’ll use the letter $P$ instead.

```c
acc = 0;
step = 0.01;
range = 10;

for (fx = uf[0] - range; fx < uf[0] + range; fx += step) {
    for (fy = uf[1] - range; fy < uf[1] + range; fy += step) {
        zpred = [ fx - x[0] ; fy - y[0] ];

        acc += gaussian_prob(zpred - zobs, Pobs) *
            gaussian_prob([fx ; fy] - uf, Pekf) * step^2;
    }
}

return acc;
```

Numerical integration is illustrative but not very fast. Perhaps surprisingly, it turns out that the integration can be computed in closed form. The derivation that follows requires a few steps that would only be obvious to an experienced practitioner: do not worry too much if you don’t think you would be able to repeat this derivation on your own. However, you should be able to follow and check the derivation at every step.

Let us first expand the integral by substituting in the expressions for Gaussian probabilities. To do this, let us assume that $z(x, f)$ is the sensor model which predicts the value of a sensor reading in terms of the robot’s position $x$ and the landmark’s position $f$:

$$p(z|x) = \int p(z|f, x)p(f|x)df$$

$$= \int \frac{1}{K_z} e^{-\frac{1}{2}(z_{obs} - z(x,f))^T \Sigma_z^{-1}(z_{obs} - z(x,f))} \frac{1}{K_f} e^{-\frac{1}{2}(f - \mu_f)^T \Sigma_f^{-1}(f - \mu_f)} df$$

The $1/K$ terms represent the normalization constants associated with the respective Gaussian distributions.

The presence of $z(x, f)$ is problematic from an integration stand-point. (Can you explain why?) We will assume that a linear approximation is satisfactory. (For our simple example, such a linearization is exact). Given a constant $z_0$ and a Jacobian matrix $\bar{J}$, we can write:

$$z(x, f) \approx z_0 + J f$$
We can now substitute our linearized version of \(z(x_0, f)\) into Eqn. 1.5. Let’s focus our attention on the exponent, noting that we can add together the exponents of the two terms. We’ll also pull out the factor of \(-1/2\). We’ll reintroduce these a bit later. Our exponent now becomes:

\[(z_{\text{obs}} - z_0 - J f)^T \Sigma_z^{-1} (z_{\text{obs}} - z_0 - J f) + (f - \mu_f)^T \Sigma_f^{-1} (f - \mu_f)\]  \(1.7\)

Let us substitute \(y = z_{\text{obs}} - z_0\):

\[(y - J f)^T \Sigma_z^{-1} (y - J f) + (f - \mu_f)^T \Sigma_f^{-1} (f - \mu_f)\]  \(1.8\)

And now expand:

\[y^T \Sigma_z^{-1} y - 2y^T \Sigma_z^{-1} J f + f^T J^T \Sigma_z^{-1} J f + f^T \Sigma_f^{-1} f - 2f^T \Sigma_f^{-1} \mu_f + \mu_f^T \Sigma_f^{-1} \mu_f\]  \(1.9\)

Our goal is to re-arrange these terms to represent a Gaussian involving \(f\). If we can do that, the integral of that portion will be 1, and the value of our integral will be greatly simplified. In other words, we want to massage terms containing \(f\) into the form:

\[\Sigma_q^{-1} = J^T \Sigma_z^{-1} J + \Sigma_f^{-1}\]  \(1.13\)

What value of \(\mu_q\) should we pick? A bit of thought shows that we must have:

\[\mu_q = \Sigma_q(J^T \Sigma_z^{-1} y + \Sigma_f^{-1} \mu_f)\]  \(1.14\)

Note that the leading term is \(\Sigma_q\), not \(\Sigma_q^{-1}\). (It’s needed to cancel out the \(\Sigma_q^{-1}\) that would otherwise prevent us from obtaining the second term in Eqn. 1.12. Can you see why?) Having found \(\Sigma_q^{-1}\) and \(\mu_q\), we can now replace the first two terms of Eqn. 1.12 with:

\[(f - \mu_q)^T \Sigma_q^{-1} (f - \mu_q) - \mu_q^T \Sigma_q^{-1} \mu_q\]  \(1.15\)

Finally, we can write our simplified version of Eqn. 1.12:

\[(f - \mu_q)^T \Sigma_q^{-1} (f - \mu_q) - \mu_q^T \Sigma_q^{-1} \mu_q + y^T \Sigma_z^{-1} y + \mu_f^T \Sigma_f^{-1} \mu_f\]  \(1.16\)

We’ve achieved our goal of combining all terms containing \(f\) into a form that looks like a Gaussian probability. Things are looking good! Let’s put this stuff back inside an exponent and re-introduce the factor of \(-1/2\) that we dropped earlier:
\begin{align*}
p(z|x) &= \int p(z|f, x)p(f|x)df \\
&= \int \frac{1}{K_f} e^{-\frac{1}{2}(f-\mu_q)^T\Sigma_q^{-1}(f-\mu_q)} e^{-\frac{1}{2}(\mu_f^T\Sigma_f^{-1}\mu_f+y^T\Sigma_z^{-1}y+\mu_f^T\Sigma_f^{-1}\mu_f)/df}
\end{align*}

Let’s introduce a factor of \(K_q\frac{1}{K_q} = 1\), where \(\frac{1}{K_q}\) corresponds to the normalization constant needed for a Gaussian variable with covariance \(\Sigma_q\). Let’s also pull the other terms not involving \(f\) outside of the integral:

\begin{align*}
p(z|x) &= \int p(z|f, x)p(f|x)df \\
&= \frac{1}{K_z} \frac{1}{K_f} e^{-\frac{1}{2}(\mu_q^T\Sigma_q^{-1}\mu_q+y^T\Sigma_z^{-1}y+\mu_f^T\Sigma_f^{-1}\mu_f)} K_q \int \frac{1}{K_q} e^{-\frac{1}{2}(f-\mu_q)^T\Sigma_q^{-1}(f-\mu_q)} df
\end{align*}

Splendid! The integral contains a Gaussian random variable for \(f\) integrated over the domain of \(f\); by definition, this integral is 1. Thus, the probability that we were after is simply:

\begin{align*}
p(z|x) &= \int p(z|f, x)p(f|x)df \\
&= \frac{K_q}{K_z K_f} e^{-\frac{1}{2}(\mu_q^T\Sigma_q^{-1}\mu_q+y^T\Sigma_z^{-1}y+\mu_f^T\Sigma_f^{-1}\mu_f)}
\end{align*}

All of these quantities are known and can be plugged in without any integration over \(f\). Note that the \(K\) terms will generally be different for each particle since they will have different covariances.

This is a critical result that makes incorporating observation evidence in FastSLAM tractable. Note that this probability is used to update both the weight of the particle and the particle’s \(\chi^2\) error.