Particle Filter Localization

Monte Carlo method

2016.02.04
Hyun Ho Jeon
SLAM

- SLAM (Simultaneous localization and mapping) : In robotic mapping, **simultaneous localization and mapping (SLAM)** is the computational problem of constructing or updating a map of an unknown environment while simultaneously keeping track of an agent's location within it. ... ... Popular approximate solution methods include the **particle filter** and extended Kalman filter.

2005 DARPA Grand Challenge winner STANLEY performed SLAM as part of its autonomous driving system.
The problem with GPS is its really not very accurate. It’s common for a car to believe to be somewhere but it has error about 2-10 meters.

If you want to reduce error -> Localization
Localization

Measurement

Probability

Location

Belief

Posterior

Prior

Convolution

Uniform = Maximum Confusion

Localization

Probability

Location

Belief

Posterior

Prior

Convolution

Uniform = Maximum Confusion
Sense & Move

Gains Information

Loses Information

Initial Belief
Robot World

(distance)

(x, y, \(\theta\))

Landmark1

L2

L3

L4
Create Particles

- Particle \((x, y, \theta)\)
- \(N = 1000\)
Importance Sampling

Measurement noise
Importance Sampling
Importance Sampling

Actual measurement

Predicted measurement
Importance Sampling

Weight

Actual measurement

Predicted measurement
Importance Sampling

Importance Weights
Importance Sampling

Resampling

Importance Weights
Resampling

Particles

Weights

Normalized Weight

\[ (x_1, y_1, \theta_1) \] \[ \omega_1 \] \[ \alpha_1 = \frac{\omega_1}{W} \]

\[ (x_2, y_2, \theta_2) \] \[ \omega_2 \] \[ \alpha_2 = \frac{\omega_2}{W} \]

\[ \vdots \] \[ \vdots \]

\[ (x_N, y_N, \theta_N) \] \[ \omega_N \] \[ \alpha_N = \frac{\omega_N}{W} \]

\[ W = \sum_{i=1}^{N} \omega_i \]

\[ \sum_{i=1}^{N} \alpha_i = 1 \]
# Quiz 1-1

<table>
<thead>
<tr>
<th>Particles</th>
<th>Weights</th>
<th>Normalized Weight</th>
</tr>
</thead>
<tbody>
<tr>
<td>$p_1$</td>
<td>$\omega_1 = 0.6$</td>
<td>$\alpha_1 =$</td>
</tr>
<tr>
<td>$p_1$</td>
<td>$\omega_2 = 1.2$</td>
<td>$\alpha_1 =$</td>
</tr>
<tr>
<td>$p_1$</td>
<td>$\omega_2 = 2.4$</td>
<td>$\alpha_1 =$</td>
</tr>
<tr>
<td>$p_1$</td>
<td>$\omega_2 = 0.6$</td>
<td>$\alpha_1 =$</td>
</tr>
<tr>
<td>$p_1$</td>
<td>$\omega_2 = 1.2$</td>
<td>$\alpha_1 =$</td>
</tr>
</tbody>
</table>

$N = 5$

$W = 6.0$

Is it possible that $p_1$ is **NEVER** sampled?  

[ ] YES  

[ ] NO
## Quiz 1-2

<table>
<thead>
<tr>
<th>Particles</th>
<th>Weights</th>
<th>Normalized Weight</th>
</tr>
</thead>
<tbody>
<tr>
<td>$p_1$</td>
<td>$\omega_1 = 0.6$</td>
<td>$\alpha_1 = 0.1$</td>
</tr>
<tr>
<td>$p_1$</td>
<td>$\omega_2 = 1.2$</td>
<td>$\alpha_1 = 0.2$</td>
</tr>
<tr>
<td>$p_1$</td>
<td>$\omega_2 = 2.4$</td>
<td>$\alpha_1 = 0.4$</td>
</tr>
<tr>
<td>$p_1$</td>
<td>$\omega_2 = 0.6$</td>
<td>$\alpha_1 = 0.1$</td>
</tr>
<tr>
<td>$p_1$</td>
<td>$\omega_2 = 1.2$</td>
<td>$\alpha_1 = 0.2$</td>
</tr>
</tbody>
</table>

$W = 6.0$

Is it possible that $p_3$ is NEVER sampled?

- **YES**
- **NO**

What is the probability of NEVER sampling $p_3$?

0.0777
Resampling Wheel

\[ \text{index} = U[1 \ldots N] \]
\[ \beta = 0 \]

for \( i = 1 \ldots N \)
\[ \beta = \beta + U[0 \ldots 2 \cdot \omega_{\text{max}}] \]

while \( \omega_{\text{index}} < \beta \)
\[ \beta = \beta - \omega_{\text{index}} \]
index = index + 1

Pick \( P_{\text{index}} \)
index = U[1 ... N]

\[ \beta = 0 \]

for i = 1...N

\[ \beta = \beta + U[0 ... 2 \times \omega_{\text{max}}] \]

while \( \omega_{\text{index}} < \beta \)

\[ \beta = \beta - \omega_{\text{index}} \]

index = index + 1

Pick \( P_{\text{index}} \)
Implementation

```cpp
void main()
{
    srand((unsigned)time(NULL));
    std::cout << "OpenCV Version: " << CV_VERSION << std::endl;
    viz::Viz3d mywindow("test");
    mywindow.showWidget("MyCoordinate", viz::WCoordinateSystem(100.0));

    robot myrobot;
    myrobot = myrobot.move(0.1, 5.0);
    mywindow.spinOnce();

    vector<double> z = myrobot.sense();
    int N = 1000;
    int T = 1; // 10;

    Mat point_particle(1, N, CV_32FC3);
    Mat point_robot(1, 1, CV_32FC3);

    point_robot.at<cv::Vec3f>(0, 0)[0] = myrobot.x;
    point_robot.at<cv::Vec3f>(0, 0)[1] = myrobot.y;
    point_robot.at<cv::Vec3f>(0, 0)[2] = 0;

    mywindow.showWidget("robot", viz::WCloud(point_robot, viz::Color::red()));
    mywindow.spinOnce();
}```
Implementation

```cpp
vector<robot> p;
for (int i = 0; i < N; i++)
{
    robot r;
    r.set_noise(0.05, 0.05, 5.0);
    p.push_back(r);
    point_particle.at<cv::Vec3f>(0, i)[0] = r.x;
    point_particle.at<cv::Vec3f>(0, i)[1] = r.y;
    point_particle.at<cv::Vec3f>(0, i)[2] = 0;
    mywindow.showWidget("particle", viz::WCloud(point_particle, viz::Color(192, 192, 192)));
    mywindow.spinOnce();
    // Sleep(1);
}
```

Create Particle

```
for (int t = 0; t < T; t++)
{
    myrobot = myrobot.move(0.1, 5.0);
    Z = myrobot.sense();
    point_robot.at<cv::Vec3f>(0, 0)[0] = myrobot.x;
    point_robot.at<cv::Vec3f>(0, 0)[1] = myrobot.y;
    point_robot.at<cv::Vec3f>(0, 0)[2] = 0;
    mywindow.showWidget("robot", viz::WCloud(point_robot, viz::Color::red()));
    vector<robot> p2;
    for (int i2 = 0; i2 < N; i2++)
    {
```

Robot Motion & measurement Update
Implementation

Particle Motion Update

```
    p2.push_back(p[i2].move(0.1, 5.0));
    point_particle.at<cv::Vec3f>(0, i2)[0] = p2[i2].x;
    point_particle.at<cv::Vec3f>(0, i2)[1] = p2[i2].y;
    point_particle.at<cv::Vec3f>(0, i2)[2] = 0;
    myWindow.showWidget("particle", viz::WCloud(point_particle, viz::Color(192, 192, 192)));
    myWindow.spinOnce();
    // Sleep(1);
}
p = p2;
```

Importance Weight

```
    vector<double> w;
    for (int 13 = 0; 13 < N; 13++)
    {
        w.push_back(p[i3].measurement_prob(z));
    }
```
Implementation - Weight

double robot::Gaussian(double mu, double sigma, double r_x)
{
   // calculates the probability of x for 1-dim Gaussian with mean mu and var. sigma
   return exp(-((mu - r_x), 2.0)) / pow(sigma, 2.0) / 2.0) / sqrt(2.0 * pi * pow(sigma, 2.0));
}

double robot::measurement_prob(vector<double> &measurement)
{
   // calculate the correct measurement
   double prob = 1.0;
   for (int i = 0; i < 4; i++)
   {
      double dist = sqrt(pow((x - landmarks[1][0]), 2.0) + pow((y - landmarks[1][1]), 2.0));
      prob *= Gaussian(dist, sense_noise, measurement[i]);
   }
   return prob;
}
Implementation

```cpp
vector<robot> p3;
int index = int((rand() / RAND_MAX) * N);
double beta = 0.0;
double mw = *max_element(w.begin(), w.end());
for (int i4 = 0; i4 < N; i4++)
{
    beta += ((double)rand() / (double)RAND_MAX) * 2.0 * mw;
    while (beta > w[index])
    {
        beta -= w[index];
        index = (index + 1) % N;
    }
    p3.push_back(p[index]);
    point_particle.at<cv::Vec3f>(0, i4)[0] = p3[i4].x;
    point_particle.at<cv::Vec3f>(0, i4)[1] = p3[i4].y;
    point_particle.at<cv::Vec3f>(0, i4)[2] = 0;
```
Implementation

```cpp
mywindow.showWindow("particle", viz::WCloud(point_particle, viz::Color(192, 192, 192)));
mywindow.spinOnce();

// Sleep(1);
cout << p3[14].x << " , " << p3[14].y << endl;
}
p = p3;
cout << "p3.size() = " << p3.size() << endl;
get_position(p);
cout << t << "." << endl;
cout << "Ground truth : " << myrobot.x << " " << myrobot.y << " " << myrobot.orientation << endl;
cout << "eval : " << eval(myrobot, p) << endl;
if (check_output(myrobot, estimated_position))
{
    cout << "Code check : " << "True" << endl;
}
else cout << "Code check : " << "False" << endl;
}
while (!mywindow.wasStopped())
{
    mywindow.spinOnce();
}
```
Result
Q&A
<table>
<thead>
<tr>
<th></th>
<th>State space</th>
<th>Belief</th>
<th>Efficiency</th>
<th>In robotics</th>
</tr>
</thead>
<tbody>
<tr>
<td>Histogram Filter</td>
<td>Discrete</td>
<td>Multimodal</td>
<td>Exponential</td>
<td>Approximate</td>
</tr>
<tr>
<td>Kalman Filter</td>
<td>Continuous</td>
<td>Unimodal</td>
<td>Quadratic</td>
<td>Approximate</td>
</tr>
<tr>
<td>Particle Filter</td>
<td>Continuous</td>
<td>Multimodal</td>
<td>?</td>
<td>Approximate</td>
</tr>
</tbody>
</table>
Mathematical Representation

**Measurement Update**

\[ P(X|Z) \propto P(Z|X)P(X) \]

**Motion Update**

\[ P(X') = \sum P(X'|X)P(X) \]