Guidance for Project 4 SLAM

ESE650 University of Pennsylvania

1 Preparation

1. Read the assignment description and *docs/config_slam.pdf* carefully. Try to understand (a) components of the given data, (b) how and where (frame) the data is collected (in order to understand the model), (c) how to use the given codes, and (d) which steps you need to follow to complete the project.

2. Try mapping from the first scan and plot the map

3. Try dead-reckoning and plot the robot trajectory

4. Try prediction only and plot the robot trajectories (100 for $N = 100$ particles)

5. Try the update step with only 3-4 particles and see if the weight update makes sense

2 Notations

- $x^{(i)}$ represents a vector or a scalar x in frame i. $(g), (b), (h)$ mean global, body, and head frames respectively.

- $R(\theta)$ is a rotation matrix:
  For $SO(2)$ (*special orthonormal group*),
$$\begin{bmatrix} \cos(\theta) & -\sin(\theta) \\ \sin(\theta) & \cos(\theta) \end{bmatrix}$$
  For $SO(3)$, rotations with respect to z, y, x-axis, respectively,
$$\begin{bmatrix} \cos(\theta_z) & -\sin(\theta_z) & 0 \\ \sin(\theta_z) & \cos(\theta_z) & 0 \\ 0 & 0 & 1 \end{bmatrix} \quad \begin{bmatrix} \cos(\theta_y) & 0 & \sin(\theta_y) \\ 0 & 1 & 0 \\ -\sin(\theta_y) & 0 & \cos(\theta_y) \end{bmatrix} \quad \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos(\theta_x) & -\sin(\theta_x) \\ 0 & \sin(\theta_x) & \cos(\theta_x) \end{bmatrix}$$

- Homogeneous transform matrix between Frame 0 and Frame 1, A_1^0, is defined as:
$$A_1^0 = \begin{bmatrix} R_1^0 & \mathbf{d}_1^0 \\ \mathbf{0}^T & 1 \end{bmatrix}$$
  where $R_1^0 \in SO(3)$ is a rotatio matrix of Frame 1 with respect to Frame 0, $\mathbf{d}_1^0 \in \mathbb{R}^3$ is a vector of the origin of Frame 1 with respect to Frame 0, and $\mathbf{0}^T = [0, 0, 0]$. Therefore, A_1^0 belongs to the *special Euclidean group* $SE(3)$.

- Coordinate transformation of Frame n with respect to Frame 0, T_n^0:
$$T_n^0 = A_1^0 A_2^1 \cdots A_n^{n-1}$$
  and
$$\mathbf{v}^{(0)} = T_n^0 \mathbf{v}^{(n)}$$
  for a vector $\mathbf{v}$.

- Smart Plus

$$x_{t+1} \oplus x_t \equiv \begin{bmatrix} p_t + R(\theta_t)p_{t+1} \\ \theta_t + \theta_{t+1} \end{bmatrix}$$

$$x_{t+1} \ominus x_t \equiv x_t^{-1} \oplus x_{t+1} = \begin{bmatrix} R^T(\theta_t)(p_{t+1} - p_t) \\ \theta_{t+1} - \theta_t \end{bmatrix}$$

* Note that some of notations may be different from notations in your reference materials.

3 Mapping

Input

1. laser scan z_t (laser.scan)
2. transform from head to body frame: T_h^b (head and neck angles)
3. robot pose x_t determining transform from body to global frame: T_b^g (current best particle and could use laser.rpy but be careful with transform from IMU to body frame)
4. current log-odds map m_t

Output

1. Updated log-odds map m_{t+1}

Pseudo-code

1. Transform $z_t = z_t^{(h)}$ via $T_b^g * T_h^b$ to the global frame
2. Remove scan points that are too close, too far, or hit the ground
3. Use `MapUtils/bresenham2D.py/getMapCellsFromRay` to obtain the cell locations y_t^o that are occupied according to the laser and y_t^f that are free according to the laser
4. Increase the log-odds in m_t of the occupied cells y_t^o and decrease the odds on the free cells y_t^f to obtain m_{t+1}

4 Dead-reckoning

Input

1. Current robot pose $p_t \in SE(2)$
2. Global frame odometry (laser.odom) o_t and o_{t+1}

Output

1. Updated robot pose $p_{t+1} \in SE(2)$

Pseudo-code

1. $p_{t+1} = p_t \oplus (o_{t+1} \ominus o_t)$

5 Localization Prediction

Input

1. Current particles: $p_t^n \in SE(2), \quad n = 1, \ldots, N$
2. Global frame odometry (laser.odom) o_t and o_{t+1}

Output

1. Updated particles: $p_{t+1}^n \in SE(2), n = 1, \ldots, N$

Pseudo-code

1. $p_{t+1}^n = p_t^n \oplus (o_{t+1} \ominus o_t) \oplus w_t^n, \qquad w_t^n \sim \mathcal{N}(0, W_{3\times 3})$

6 Localization Update

6.1 Algorithm

Input

1. Current particle positions and weights: $(p_t^n, a_t^n), n = 1, \ldots, N$
2. Laser scan z_t
3. Current map m_t
4. Transform from head to body: T_h^b (head and neck angles)

Output

1. Updated particle positions and weights: $(p_{t+1}^n, a_{t+1}^n), n = 1, \ldots, N$

Pseudo-code

1. For each particle $n = 1, \ldots, N$:
   - Transform z_t via $T_b^g * T_h^b$ to the global frame, where T_b^g is determined from p_t^n (and optionally laser.rpy but be careful with transform from IMU to robot center of mass)
   - Remove scan points that are too close, too far, or hit the ground
   Find the cells y_t corresponding to the global-frame scan z_t. For speed you can just use the points that were hit by the laser instead of `MapUtils/bresenham2D.py/getMapCellsFromRay`
   - Compute $corr(m_t, y_t)$ using mapCorrelation in MapUtils.py. Call mapCorrelation with a grid of values (e.g., 9x9) around the current particle position to get a good correlation. See test_MapUtils.py for an example variation in x and y. You can also consider adding variation in yaw for the particle.
2. Update the particle weights (see the below section)
3. If $N_{eff} < N_{threshold}$, re-sample the particles

6.2 Updating Weights

There are two ways to compute the weights: one easy but slightly incorrect and one easy but correct. We define the measurement likelihood as follows:

$$p_h(z_t|x, m) = \frac{exp(corr(z_t, m))}{\sum_z exp(corr(z, m))}$$

We are interested in the following in the particle filter update step:

$$a_{t+1|t+1}^{(k)} = \eta_{t+1} a_{t+1|t}^{(k)} exp(corr(z_{t+1}, m)$$

where η_{t+1} is the normalization due to $\sum_z exp(corr(z, m))$ and $\sum_j a_{t+1|t}^{(j)} p_h(z_{t+1}|\mu_{t+1|t}^{(j)}, m)$.

1. The easy, slightly incorrect way:
   a) say that $p_h(z_t|x, m) \propto corr(z_t, m)$
   b) update weights: $a_{t+1|t+1}^{(k)} = a_{t+1|t}^{(k)} * corr(z_{t+1}, m)$
   c) normalize: $a_{t+1|t+1}^{(k)} \leftarrow \frac{a_{t+1|t+1}^{(k)}}{\sum_j a_{t+1|t+1}^{(j)}}$

2. The easy, correct way
   a) say that $p_h(z_t|x, m) \propto exp(corr(z_t, m))$ and define $w_{t|t}^{(k)} := log(a_{t|t}^{(k)})$
   b) update weights: $w_{t+1|t+1}^{(k)} = w_{t+1|t}^{(k)} + corr(z_{t+1}, m)$
   c) normalize: $w_{t+1|t+1}^{(k)} \leftarrow w_{t+1|t+1}^{(k)} - \max_j w_{t+1|t+1}^{(j)} - \log \sum_i exp(w_{t+1|t+1}^{(i)} - \max_j w_{t+1|t+1}^{(j)})$

The function $logsumexp(\mathbf{x}, b) := b + \log \sum_j exp(x_j - b)$ is translation invariant in the second argument, so it can be computed robustly via: $lse(\mathbf{x}, \max_j \mathbf{x}_j) = \max_j \mathbf{x}_j + \log \sum_i exp(x_i - \max_j \mathbf{x}_j)$ as suggested above.

7 SLAM

Initialize $p_0^n = (0, 0, 0), \;\; a_0^n = \frac{1}{N}, \;\; n = 1, \ldots, N$
Input

1. Current particle positions and weights: $(p_t^n, a_t^n), n = 1, \ldots, N$
2. Laser scan z_t (laser.scan)
3. Current map m_t
4. Transform from head to body: T_h^b (head and neck angles)
5. Global frame odometry o_t and o_{t+1} (laser.odom)

Output

1. Updated particle positions and weights: $(p_t^n, a_t^n), n = 1, \ldots, N$
2. Updated log-odds map m_{t+1}

Pseudo-code

1. Find particle p_t^* with highest weight from $(p_t^n, a_t^n), n = 1, \ldots, N$
2. $m_{t+1} \leftarrow Mapping(z_t, p_t^*, T_h^b, m_t)$
3. $p_{t+1}^n \leftarrow LocalizationPrediction(p_t^n, o_t, o_{t+1})$
4. $(p_{t+1}^n, a_{t+1}^n) \leftarrow LocalizationUpdate(p_{t+1}^n, a_t^n, z_t, m_{t+1}, T_h^b)$

8 Texture Mapping

8.1 Algorithm

Input

1. rgb image I_t and depth image d_t
2. transform from head to body: T_h^b (head and neck angles, note: the kinect is at different height from the laser)
3. robot pose x_t determining transform from body to global: T_b^g (current best particle and laser.rpy)
4. Current texture map: tm_t

Output

1. Updated texture map: tm_{t+1}

Pseudo-code

1. Transform I_t and d_t via $T_b^g * T_h^b$
2. Find the ground plane in the transformed data via RANSAC or simple thresholding on the height
3. Color the cells in tm_t using the points from I_t and d_t that belong to the ground plane

8.2 Ground Detection

See *docs/ground_detection.pdf* file.

- You need only 'fc' and 'cc' from *IRcam_Calib_result.pkl* and *RGBcamera_Calib_result.pkl* files.
- For 'fc' (focal length), the first element corresponds to f_u and the second element corresponds to f_v in the pdf file page 6.
- Find the parameters a_0, a_1, a_2, a_3 as the page 10 instruction, choose an appropriate threshold value for ϵ and detect ground points.
- As in the page 2, u corresponds to x-axis and columns and v corresponds to y-axis and rows. Note that you need to compute appropriate pixel positions using the principal point values before applying the equations in the document:
- Let (u', v') be a pixel position in a given RGB (or depth) image matrix and $c = (c_x, c_y)$ be the RGB (or depth) principal point provided in the calibration pickle files. Then (u, v) used in the document is: $u = u' - c_x$ and $v = v' - c_y$. For example, the correct (u,v) you will use for the pixel at (0,0) in a depth image = $(-258.42, -202.49)$.