Assignment 4: Search and rescue

ETH Zurich

Individual Demonstration: Monday, 08.12.2014 at 16:15
Individual Software Due: Monday, 08.12.2014 at 23:00
Group Work Demonstration: Friday, 19.12.2014 at 09:15
Group Work Software Due: Friday, 19.12.2014 at 23:00

1 Localization

1.1 Background

Mobile robot localization, also known as position estimation, is the process of determining a robot's pose with respect to a given map of an environment. The ability to localize is critical to all robots that need to interact with their environment. To address this problem, researchers have developed various algorithms. One of the popular approaches is particle filter localization.

Particle filter localization [1, 2] is a localization method based on particle filter. Particle filter is a nonparametric filter in which a distribution is represented by a set of random samples drawn from the distribution. The samples are called *particles*, $X_t = \{x_t^1, x_t^2, ..., x_t^M\}$, and in localization, they represent concrete instantiations of the robot state $x_t^m = (x, y, \theta)$ at time t. The number of particles, M, influences the accuracy of the localization algorithm.

Algorithm 1: Particle filter localization algorithm

Data: X_{t-1}: a set of particles (robot states) at time $t-1$
u_t: the robot control at time t
z_t: the sensor measurement at time t
Result: X_t: a set of particles at time t
begin
  $\bar{X}_t = X_t = \emptyset$
  for $m = 1$ *to* M **do**
    $x_t^{[m]} =$ **motion_update**$(u_t, x_{t-1}^{[m]})$
    $w_t^{[m]} =$ **sensor_update**$(z_t, x_t^{[m]})$
    $\bar{X}_t = \bar{X}_t + \langle x_t^{[m]}, w_t^{[m]} \rangle$
  for $m = 1$ *to* M **do**
    $x_t^{[i]} =$ **resample**$(\bar{X}_t)$
    $X_t = X_t + x_t^{[i]}$
end

The basic algorithm is shown in Algorithm 1. The algorithm takes a set of particles from the previous time $t-1$, the control input, and current measurement as input and produces a new set of particles. The **motion_update** function updates a previous particle $x_{t-1}^{[m]}$ by moving it according to the control u_t and adding some Gaussian noise. The **sensor_update** function updates the weight of each particle by computing the probability of the current measurement reading given the robot pose. Lastly, the **resample** function takes the updated particles and

their weights and draws new particles such that the probability of a particle $x_t^{[i]}$ being drawn is proportional to its weight $w_t^{[i]}$.

The **motion_update** function can be implemented as a sample odometry motion model, shown in Algorithm 2. The model estimates the robot's motion u_t between the poses x_{t-1} and x_t from the difference in the odometry between $\bar{x}_{t-1} = (\bar{x}, \bar{y}, \bar{z})^T$ $\bar{x}_t = (\bar{x}', \bar{y}', \bar{z}')^T$. Using the relative motion in the odometry, the model then updates the initial pose x_{t-1} and outputs a random pose x_t distributed according to $p(x_t \mid u_t, x_{t-1})$. The model assumes that errors in the robot motion, represented using error parameters α_1 to α_4, are independent and draws a sample perturbed by the error. The **sample**(b^2) function can encode an approximate normal distribution $\frac{1}{2}\sum_{i=1}^{12}$ **rand**$(-b, b)$ or a triangular distribution $\frac{\sqrt{6}}{2}$(**rand**$(-b, b)$ + **rand**$(-b, b)$).

Algorithm 2: Motion update

Data: $u_t = (\bar{x}_{t-1}, \bar{x}_t)^T$: the robot control at time t
$x_{t-1} = (x, y, \theta)^T$: initial pose at time $t-1$
Result: x_t: robot pose at time t
begin
– Recover relative motion parameters $(\delta_{\text{rot1}}, \delta_{\text{trans}}, \delta_{\text{rot2}})^T$ from u_t.
$\delta_{\text{rot1}} = \text{atan2}(\bar{y}' - \bar{y}, \bar{x}' - \bar{x}) - \bar{\theta}$
$\delta_{\text{trans}} = \sqrt{(\bar{x} - \bar{x}')^2 + (\bar{y} - \bar{y}')^2}$
$\delta_{\text{rot2}} = \bar{\theta}' - \bar{\theta} - \delta_{\text{rot1}}$
– Perturb the motion parameters by noise in robot motion.
$\hat{\delta}_{\text{rot1}} = \delta_{\text{rot1}} - \textbf{sample}(\alpha_1\delta_{\text{rot1}}^2 + \alpha_2\delta_{\text{trans}}^2)$
$\hat{\delta}_{\text{trans}} = \delta_{\text{trans}} - \textbf{sample}(\alpha_3\delta_{\text{trans}}^2 + \alpha_4\delta_{\text{rot1}}^2 + \alpha_4\delta_{\text{rot2}}^2)$
$\hat{\delta}_{\text{rot2}} = \delta_{\text{rot2}} - \textbf{sample}(\alpha_1\delta_{\text{rot2}}^2 + \alpha_2\delta_{\text{trans}}^2)$
– Update the output pose x_t using the sample motion parameters.
$x' = x + \hat{\delta}_{\text{trans}} \cos(\theta + \hat{\delta}_{\text{rot1}})$
$y' = y + \hat{\delta}_{\text{trans}} \sin(\theta + \hat{\delta}_{\text{rot1}})$
$\theta' = \theta + \hat{\delta}_{\text{rot1}} + \hat{\delta}_{\text{rot2}}$
$x_t = (x', y', \theta')^T$
end

After predicting a new pose for every particle, the localization algorithm updates their weight in the **sensor_update** step. One way to implement the **sensor_update** function is map matching. The core idea behind map matching is to compute the correlation between a global map – given as input – and a local map – computed by transforming the scans into an occupancy map. Given a robot at x_t, let $m_{x,y,\text{local}}(x_t)$ be its location in the local map and (x, y) be the corresponding location in the global map. Assuming that both maps have the same reference frame, the map correlation can be computed as

$$\rho_{m,m_{\text{local}},x_t} = \frac{\sum_{x,y}(m_{x,y} - \bar{m})(m_{x,y,\text{local}}(x_t) - \bar{m})}{\sqrt{\sum_{x,y}(m_{x,y} - \bar{m})^2 \sum_{x,y}(m_{x,y,\text{local}}(x_t) - \bar{m})^2}} \tag{1}$$

Here $\bar{m}$ is the average map value over the elements (N) that overlap between the maps, i.e.,

$$\bar{m} = \frac{1}{2N}\sum_{x,y}(m_{x,y} + m_{x,y,\text{local}}) \tag{2}$$

From the correlation $\rho_{m,m_{\text{local}},x_t}$, we can compute the probability of the local map conditioned on the global map m and the robot pose x_t as

$$p(m_{\text{local}}|x_t, m) = \max\{\rho_{m,m_{\text{local}},x_t}, 0\} \tag{3}$$

Taking the probability from the **sensor_update** step as the particle's weight w_t, we can resample the particles based on their importance. A popular **resample** function is stochastic universal sampling. Stochastic universal sampling draws samples evenly such that the new sample set is not biased. Given the mean weight $\bar{w}_t = \frac{1}{M}\sum_M w_t^{[m]}$ and an initial random selection $w_t^{\text{initial}} \in [0, \bar{w}_t)$, a new i'th particle $x_t^{[i]}$, where i ranges from 1 to M, is set to the previous m'th particle $x_t^{[m]}$ such that $\sum_{m'=1}^{m} w_t^{[m']} \leq ((i-1)\bar{w}_t + w_t^{\text{initial}})$ and $\sum_{m'=1}^{m+1} w_t^{[m']} > ((i-1)\bar{w}_t + w_t^{\text{initial}})$ [1].

Particle filter localization is able to represent wide range of distributions and thus solve global localization problem. In addition, if some particles are replaced by random samples, it is able to handle kidnapped robot problem in which a robot can find its way out of a wrong pose prediction.

Some useful tutorials on particle filter include:

- http://robots.stanford.edu/papers/fox.mcmc-book.ps.gz
- http://www.cim.mcgill.ca/~yiannis/particletutorial.pdf

1.2 Task

Implement a particle filter localization algorithm. Your robot will start with an unknown pose and will go to a goal location. You are free to implement any sensible prediction and update methods. Play with the number of particles and sample distributions.

1.3 Practical Note

1.3.1 depthimage_to_laserscan

You can get "laser scans" from the Carmine 1.09 sensor using `depthimage_to_laserscan`. The package takes a depth image and generates a 2D laser scan. More information on the package is found here: http://wiki.ros.org/depthimage_to_laserscan.

1.3.2 static_transform_publisher

Static transform publisher publishes a static coordinate transform to tf between a frame and its child frame. This tool is useful when the relationship two frames is static, e.g. between a robot and its camera.

1.3.3 rosbag

Take advantage of repeatable executions! Record robot's odometry information in a real setup and use it each time you want to recreate the testing environment for the localization process. Explore useful `rosbag` commands here: http://wiki.ros.org/rosbag/Commandline.

1.3.4 Design patterns

Always think about reusability and extendability of your code. Current assignment is a good place to apply Object-Oriented approach and design patterns you have learned in the course.

[1] Note that the algorithm assumes that particles are indexed from 1 to M, not 0 to $M-1$.

2 Grading

2.1 In-class demonstration

The goal of this assignment is to demonstrate how your robot can, given a map, search the environment and rescue "people".

2.1.1 Individual In-class demonstration (10 points): Monday, 08.12.2014 at 16:15

In the individual portion, you will demonstrate how your system can localize in an unknown environment. In `data/localization`, you will find a bag file and an image that shows the robot's path.

- Predict (2 points)
- Update (2 points)
- Resample (1 point)
- Localize (5 points)
  - Localized within 80 seconds (2 points)
  - Stayed localized/re-localized in 120 seconds (1 point)
  - Stayed localized/re-localized in 160 seconds (1 point)
  - Stayed localized/re-localized in 200 seconds (1 point)

The grading scheme allows you to get partial credit even if your localization system as a whole does not work. In other words, you can demonstrate predict, update, and/or resample individually and get the corresponding points. Naturally, if your system can localize, you will get the points for predict, update, and resample in addition to the points for localization.

2.1.2 Group In-class demonstration (10 points): Friday, 19.12.2014 at 09:15

In the group portion, your robot will travel to a goal from an unknown pose while looking for "people" on the way. You will only be given the goal location; you are free to plan a path to the goal once your robot localizes. The robot will begin in a place without any object in the proximity, but on its way to the goal, it will encounter different objects. The environment will have a couple of non-humans as well, and your robot must be able to tell which of the objects are "humans". The environment will also have an unknown obstacle.

The grading scheme is as follows:

- Object recognition (4 points)
  - Correct recognition of "human" (2 objects, 2 points each)
    - Label indicated by the robot (sound, LEDs, etc.) (1 point)
    - Correctly-located bounding box in the map coordinate frame (1 point)
  - Recognition of non-human as "human" (-1 point per object)
  - Note: Recognized objects should be displayed in RViz in the map coordinate frame. After a run, the RViz should only display the map and one bounding box per recognized "human". If there are multiple bounding boxes over an object, the grade will be reduced by the number of bounding boxes.
- Speed of search (2 points)

  - Fastest group plus 10 seconds (2 points)
  - Every 10 seconds thereafter (-0.1 point)

- Accuracy (4 points)
  - Closest to the goal plus $1cm$ (4 points)
  - Every $1cm$ thereafter (-0.1 point)

Each group will get two attempts. Every extra attempt will cost 1 point. Bumping into obstacles will also cost 1 point.

2.2 Software quality (20 points)

On the due date at 23:00, we will collect your code through your SVN repository. Every file that should be considered for grading must be in the repository at that time. Note that EIFGENs folder in your project contains auxiliary files and binaries after compilation. Please, DO NOT include EIFGENs folder into your svn repository.

2.2.1 Individual evaluation (10 points): Monday, 08.12.2014 at 23:00

- Choice of abstraction and relations (3 points)
- Correctness of implementation (4 points)
- Extendibility and reusability (2 points)
- Comments and documentation, including "README" (1 points)

2.2.2 Group evaluation (10 points): Friday, 19.12.2014 at 23:00

- Choice of abstraction and relations (3 points)
- Correctness of implementation (4 points)
- Extendibility and reusability (2 points)
- Comments and documentation, including "README" (1 points)

References

[1] Thrun, S., Burgard, W., Fox, D. 2005. Probabilistic Robotics MIT Press. Chapter 8.

[2] Dellaert, F., Fox, D., Burgard, W., Thrun, S., 1999. Monte Carlo Localization for Mobile Robots, ICRA99.