Robotic Localization: Map_bot

Juil O (horagong@gmail.com)

Abstract—Two different robot models are considered for performance evaluation. The robot models have a camera, a laser sensor and actuators. The move_base and amcl ROS packages are used for AMCL(Adaptive Monte Carlo Localization) algorithm and the parameters are tuned and tuned to get accurate results.

Index Terms—robot, localization, particle filter

1 Introduction

THE robot localization problem is estimating a robots coordinates relative to an given map of the external reference frame. It's pertaining to robot perception and a necessary prerequisite for robot navigation. If the pose of the robot is known through sensor data, it's easy to get the correspondence between the map coordinate system and the robots local coordinate system. But a single sensor measurement is usually insufficient to determine the pose because of the sensor noise. So the robot has to integrate data over time to determine its pose and the algorithm for posterior estimation over the space of robot locations should be used.

2 Background

The most general algorithm for calculating beliefs is given by the Bayes filter algorithm. This algorithm calculates the belief distribution $bel(x_t)$ from measure- ment and control data. The straightforward application of Bayes filters to the localization problem is called Markov localization.

Bayes filter applies the following update rule recursively to calculate the $bel(x_t)$ from the $bel(x_{t-1})$. [1]

Algorithm Bayes_filter($bel(x_{t-1}), u_t, z_t$**):**

for all x_t *do*

$$\overline{bel}(x_t) = \int p(x_t \mid u_t, x_{t-1})\ bel(x_{t-1})\ dx_{t-1}$$

$$bel(x_t) = \eta\ p(z_t \mid x_t)\ \overline{bel}(x_t)$$

endfor

return $bel(x_t)$

Fig. 1. Bayes Filter

The robot executes a control action u_1 first, and then takes a measurement z_1. Here $bel(x_t)$ is abbreviation for the posterior taken after incorporating the measurement z_t, $bel(x_t) = p(x_t|z_{1:t}, u_{1:t})$. $\overline{bel}(x_t)$ is the prediction of the state before incorporating the measurement at time t, $\overline{bel}(x_t) = p(x_t|z_{1:t-1}, u_{1:t})$.

The Bayes filter algorithm possesses two essential steps. The probability $p(x_t|x_{t-1}, u_t)$ is the state transition probability. The probability $p(z_t|x_t)$ is called the measurement probability

- Prediction or Control update
- Correction or Measurement update

In the first step, the belief $\overline{bel}(x_t)$ is the total probability obtained by the integral (sum) of the product of two distributions: the prior assigned to x_{t-1}, and the the state transition probability that control u_t induces from x_{t-1} to x_t.

The second step, the belief $bel(x_t)$ is obtained by bayes rule that multiplies the belief $bel(x_t)$ by the measurement probability that z_t may have been observed.

2.1 Kalman Filters

The algorithm Bayes filter can only be implemented in the form stated here for very simple estimation problems which are possible to carry out the integration in control update and the multiplication in measurement update in closed form, or need to be in finite state spaces so that the integral in control update becomes a (finite) sum.

Kalman filter is the Bayes filter for a linear Gaussian transition and observation model. Posteriors $bel(x_t)$ are Gaussian if the following three properties hold, in addition to the Markov assumptions of the Bayes filter.

- The transition model is a linear in its arguments, with added Gaussian system noise vector, $\epsilon_\mathbf{t} \sim N(\mathbf{0}, \mathbf{R_t})$.
  $\mathbf{x_t} = \mathbf{A_t x_{t-1}} + \mathbf{B_t u_t} + \epsilon_\mathbf{t}$
- The observation model is linear in its arguments, with added Gaussian measurement noise vector, $\delta_\mathbf{t} \sim N(\mathbf{0}, \mathbf{Q_t})$.
  $\mathbf{z_t} = \mathbf{C_t x_t} + \delta_\mathbf{t}$
- The initial belief $bel(\mathbf{x_0}) \sim N(\mu_\mathbf{0}, \mathbf{\Sigma_0})$

If a robot moves with constant translational and rotational velocity typically moves on a circular trajectory, it cannot be described by linear state transitions. State transitions and measurements are rarely linear in practice so Kalman filter is inapplicable to all but the most trivial robotics.

The extended Kalman filter, or EKF, relaxes the linearity assumption. Here the assumption is that the state transition probability and the measurement probabilities are governed by nonlinear functions g and h, respectively.

- $\mathbf{x_t} = \mathbf{g}(\mathbf{u_t}, \mathbf{x_{t-1}}) + \epsilon_\mathbf{t}$
- $\mathbf{z_t} = \mathbf{h}(\mathbf{x_t}) + \delta_\mathbf{t}$

Algorithm Kalman_filter($\mu_{t-1}, \Sigma_{t-1}, u_t, z_t$):

$$\bar{\mu}_t = A_t\ \mu_{t-1} + B_t\ u_t$$
$$\bar{\Sigma}_t = A_t\ \Sigma_{t-1}\ A_t^T + R_t$$
$$K_t = \bar{\Sigma}_t\ C_t^T (C_t\ \bar{\Sigma}_t\ C_t^T + Q_t)^{-1}$$
$$\mu_t = \bar{\mu}_t + K_t(z_t - C_t\ \bar{\mu}_t)$$
$$\Sigma_t = (I - K_t\ C_t)\ \bar{\Sigma}_t$$
$$\textit{return}\ \mu_t, \Sigma_t$$

Fig. 2. Kalman Filter

2.2 Particle Filters

Briefly explain what a particle filter is, how it is used, and why it is useful.

A popular alternative to Gaussian techniques are nonparametric filters. Nonparametric filters do not rely on a fixed functional form of the posterior, such as Gaussians. Instead, they approximate posteriors over continuous spaces with finitely many values.

Some nonparametric Bayes filters decompose the state space into finitely many regions and represents the posterior by a histogram which is a piecewise constant approximations to a continuous density. When applied to finite spaces, such filters are known as discrete Bayes filters. The discrete Bayes filter algorithm is popular in many areas of signal processing, where it is often referred to as the forward pass of a hidden Markov model, or HMM. When applied to continuous spaces, they are commonly called histogram filters.

Others represents posteriors by finitely many samples drawn from the posterior distribution. The resulting filter is known as particle filter and has become immensely popular in robotics.

Algorithm Particle_filter($\mathcal{X}_{t-1}, u_t, z_t$):

$\bar{\mathcal{X}}_t = \mathcal{X}_t = \emptyset$
for $m = 1$ *to* M *do*
  sample $x_t^{[m]} \sim p(x_t \mid u_t, x_{t-1}^{[m]})$
  $w_t^{[m]} = p(z_t \mid x_t^{[m]})$
  $\bar{\mathcal{X}}_t = \bar{\mathcal{X}}_t + \langle x_t^{[m]}, w_t^{[m]} \rangle$
endfor
for $m = 1$ *to* M *do*
  draw i *with probability* $\propto w_t^{[i]}$
  add $x_t^{[i]}$ *to* $\mathcal{X}_t$
endfor
return $\mathcal{X}_t$

Fig. 3. particle filter

2.3 Comparison / Contrast

Here particle filter is used for localization. It has the following advantages over EKF.

TABLE 1
EKF vs Particle filter

	EKF	Particle Filter
posterior	Gaussian unimodal	non-Gaussian multimodal
implementation	hard	easy
adaptivity	no	yes

In particle filter, the quality of the approximation depends on the number of parameters used to represent the posterior. Particle filters make it possible to adapt the number of parameters to the (suspected) complexity of the posterior. When the posterior is of low complexity (e.g., focused on a single state with a small margin of uncertainty), they use only small numbers of parameters. For complex posteriors, e.g., posteriors with many modes scattered across the state space, the number of parameters grows larger.

Particle filters can be resource-adaptive if they can adapt the number of parameters online based on the computational resources available for belief computation. They enable robots to make decisions in real time, regardless of the computational resources available.

3 Simulations

The benchmark model and personal model are used for localization task.

3.1 Achievements

They all can move and get to the goal position.

3.2 Model design

TABLE 2
benchmark model vs map_bot model

	benchmark model	map_bot model
chassis	box: ”.4 .2 .1” with two casters	box: ”.4 .2 .1” without casters
wheel	two wheels cylinder: r=”0.1” l=”0.05”	four wheels cylinder: r=”0.1” l=”0.05”
body	N.A.	cylinder: r=”0.1” l=”0.3”
sensors	a camera in front of chassis a laser on top of chassis	a camera in front of chassis a laser on top of chassis

Udacity_bot has a camera and a laser sensor and moves with two wheels like the following.

Map_bot has a cmera and a laser sensor on the cylinder body and moves with four wheels.

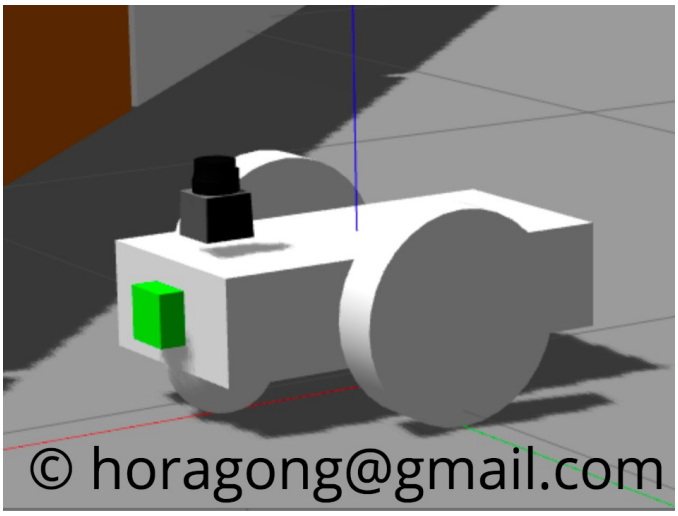

Fig. 4. udacity_bot

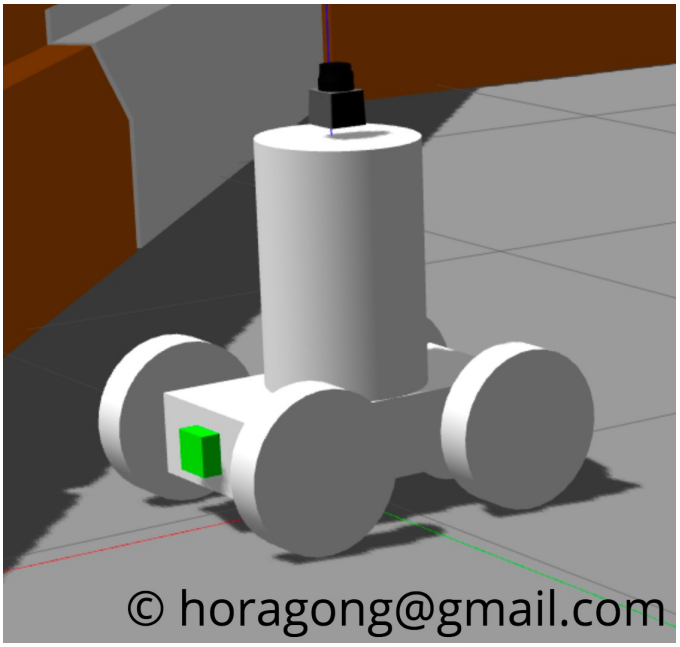

Fig. 5. map_bot

3.3 Packages Used

Both of them use ROS navigation stack which includes move_base, map_server and amcl nodes.

- map_server
  - publish: map
- Sensor sources publish the following topics
  - laser publish: /udacity_bot/laser/scan
  - camera publish: image_raw, camera_info
- Odometry source
  - publish: odom
  - subscribe: cmd_vel
- move_base
  - publish: path, cmd_vel
  - subscribe: map, sensors/odometry topics

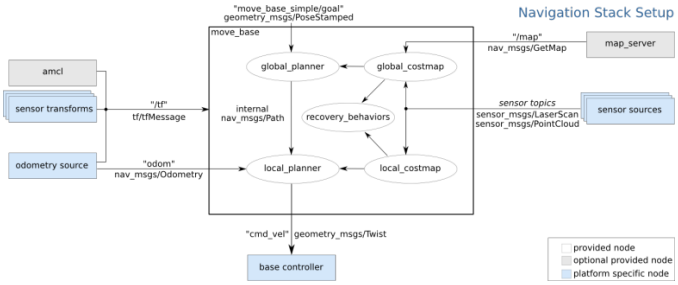

Fig. 6. packages

3.4 Parameters

Both of the robots use the same parameter setting. The customized parameters are the followings.

- move_base node
  - costmap_common_params.yaml
    - obstacle_range: 3.0 #determines the maximum range sensor reading that will result in an obstacle being put into the costmap.
    - raytrace_range: 3.0 #determines the range to which we will raytrace freespace given a sensor reading.
    - robot_radius: 0.3
    - inflation_radius: 0.6
  - base_local_planner_params.yaml
    - holonomic_robot: false #Determines whether velocity commands are generated for a holonomic or non-holonomic robot.
    - controller_frequency: 0.5 #The frequency at which this controller will be called in Hz.
    - meter_scoring: true #Whether the gdist_scale and pdist_scale parameters should assume that goal_distance and path_distance are expressed in units of meters or cells.
    - pdist_scale: 1.5 #The weighting for how much the controller should stay close to the path it was given, maximal possible value is 5.0
    - gdist_scale: 1.0 #The weighting for how much the controller should attempt to reach its local goal, also controls speed, maximal possible value is 5.0
    - occdist_scale: 0.1 #The weighting for how much the controller should attempt to avoid obstacles
    - heading_lookahead: 3.5 #How far to look ahead in meters when scoring different in-place-rotation trajectories
  - global and local_costmap_params.yaml
    - global_frame: map #The global frame for the costmap to operate in.
    - robot_base_frame: robot_footprint #The name of the frame for the base link of the robot.

    - transform_tolerance: 0.5 #Specifies the delay in transform (tf) data that is tolerable in seconds. If the tf transform between the coordinate frames specified by the global_frame and robot_base_frame parameters is transform_tolerance seconds older than ros::Time::now(), then the navigation stack will stop the robot.
    - update_frequency: 0.5 #The frequency in Hz for the map to be updated.
    - publish_frequency: 5.0 #The frequency in Hz for the map to be publish display information.
    - static_map: true #determines whether or not the costmap should initialize itself based on a map served by the map_server. false in local.
    - rolling_window: false #Whether or not to use a rolling window version of the costmap. true in local.
    - width: 40.0 #The width of the map in meters. 5.0 in local.
    - height: 40.0 #The height of the map in meters. 5.0 in local.

- amcl node

    - update_min_d: 0.1 #Translational movement required before performing a filter update.
    - update_min_a: 0.314 #Rotational movement required before performing a filter update.
    - transform_tolerance: 0.5 #Time with which to post-date the transform that is published, to indicate that this transform is valid into the future.
    - recovery_alpha_slow: 0.001 #Exponential decay rate for the slow average weight filter, used in deciding when to recover by adding random poses. A good value might be 0.001.
    - recovery_alpha_fast: 0.1 #Exponential decay rate for the fast average weight filter, used in deciding when to recover by adding random poses. A good value might be 0.1.
    - use_map_topic: true #When set to true, AMCL will subscribe to the map topic rather than making a service call to receive its map.
    - odom_frame_id: ”odom” #Which frame to use for odometry.
    - base_frame_id: ”robot_footprint” #Which frame to use for the robot base
    - global_frame_id: ”map” #The name of the coordinate frame published by the localization system
    - odom_model_type: ”diff-corrected” #Which model to use, either ”diff”, ”omni”, ”diff-corrected” or ”omni-corrected”.
    - odom_alpha1 4: 0.1 #Specifies the expected noise in odometry’s rotation estimate from the rotational component of the robot’s motion.

4 RESULTS

4.1 Localization Results

Both of the robots can move and get to the goal position successively. They followed the generated smooth path well and took about 70s to get to the goal.

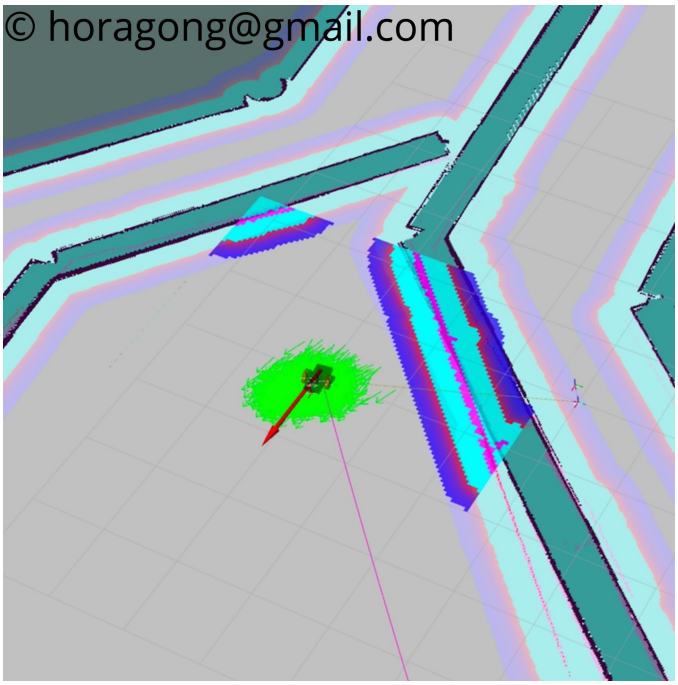

Fig. 7. udacity_bot

Fig. 8. map_bot

4.2 Technical Comparison

Benchmark robot has two wheels and it uses differential_drive_controller to drive wheels. Map_bot has four

wheels. It uses skid_steer_drive_controller to drive the four wheels. It also uses object_controller to publish odometry topic. But they use the same parameters.

5 Discussion

Both of the robots performed well but they seem to be tuned more to get better performance. The benchmark robot goes forward and backward at start to turn for the path direction and the map_bot doesn't start immediately calculating something.

odom_alpha parameters are used to specify the expected noise in odometry's rotation estimate. If it is reduced the accuracy will be better. But the value wasn't reduced for practical test.

Mobile robots in an industry domain might need to navigate in the workspace. Localization has three types. In the local localization problem, the robot knows it's initial pose. In the global localization problem, the robot's initial pose is not known. If the robot can get kidnapped and teleported to some other location during operation, the robot might believe it knows where it is while it does not. It's the kidnapped robot problem.

Most localization algorithms cannot be guaranteed never to fail. The ability to recover from failures is essential for truly autonomous robots. Particle filter would be able to recover from global localization failures using larger sampling.

6 Conclusion / Future work

Two different robot models are considered for performance evaluation. The robot models have a camera, a laser sensor and actuators. The move_base and amcl ROS packages are used for AMCL algorithm and the parameters are tuned and tuned to get accurate results. They could get to the goal in about 70s.

6.1 Modifications for Improvement

They might be tuned more to get better performance. Map_bot is desinged to be a SLAM robot. For future work, the algorithm for SLAM should be added

6.2 Hardware Deployment

The map_bot robot model should be deployed in the robot hardware. If jetson board is used, the algorithm for this localization could work well in real time.

References

[1] S. Thrun, W. Burgard, and D. Fox, *Probabilistic Robotics*. MIT press, 2006.