Marker-based FastSLAM on the AlphaBot2

Autonomous Systems Project 2023/2024

J. Pessoa
ist198915

J. Gonçalves
ist199995

M. Ribeiro
ist196446

T. Nogueira
ist1100029

Group 27

Instituto Superior Técnico

Presentation Overview

Introduction to FastSLAM

$$\boxed{p(x_{0:t}, m_{1:M} \,|\, z_{1:t}, u_{1:t})}$$

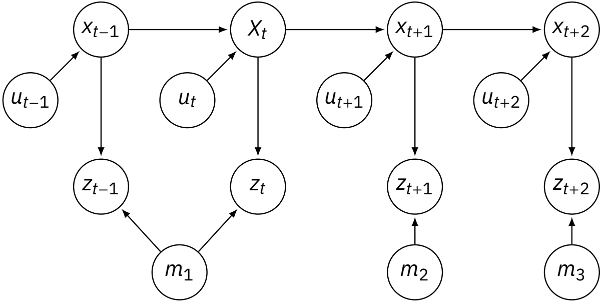


Figure: SLAM graph representation

Particle Filter

Particle Filter

- Non-parametric recursive Bayes filter.
- Posterior represented by a set of weighted samples.
- Can model arbitrary distributions.
- Works well in low-dimensional spaces.
- Three-step procedure:
  1. Sampling from proposal
  2. Importance weighting
  3. Resampling

1. Sample the particles from the proposal distribution

$$x_t^{[k]} \sim \pi(x_t \mid \dots)$$

2. Compute the importance weights

$$w_t^{[k]} = \frac{\text{target distribution}}{\text{proposal distribution}}$$

3. Resampling: Draw sample i with probability $w_t^{[k]}$ and repeat N times.

- A set of weighted samples

$$X = \{\langle x^{[k]}, w^{[k]} \rangle\}_{k=1,\dots,N}$$

- Think about samples as an hypotesis of the state
- For feature-based SLAM:

$$\text{state} = [\ \underbrace{x_{0:t}}_{\text{poses}}, \underbrace{m_1, \dots, m_M}_{\text{env. features}}\ ]^T$$

Dimensionality Problem

- Particle filters are effective in low dimensional spaces as the likely regions of the state space need to be covered with samples

$$\text{state} = [\underbrace{x_{0:t}}_{\text{poses}}, \underbrace{m_1, \ldots, m_M}_{\text{env. features}}]^T$$

high-dimensional

- The number of particles grows exponentially with the dimension of the state space!

- Can we exploit dependencies between the diffent dimensions of the state space?

$$x_{0:t}, m_1, \ldots, m_M$$

Dimensionality Problem

Key Idea

- If we know the **poses** of the robot, mapping is easy!

$$\underline{x_{0:t}}, m_1, \ldots, m_M$$

- If the particle set only models the robot's path, each sample is a path hypotesis.
- For each sample, we can build an individual map of landmarks.

Rao-Blackwellization and FastSLAM

- Factorization to exploit dependencies between variables:

$$p(a, b) = p(b \mid a)\, p(a)$$

- If $p(b \mid a)$ can be computed efficiently, represent only $p(a)$ with samples and compute $p(b \mid a)$ for every sample.

- First introduced for SLAM in 1999 by Murphy.
- Factorization of the SLAM posterior:

$$\begin{aligned} p(x_{0:t}, m_{1:M} \mid z_{1:t}, u_{1:t}) &= p(x_{0:t} \mid z_{1:t}, u_{1:t})\, p(m_{1:M} \mid x_{0:t}, z_{1:t}) \\ &= p(x_{0:t} \mid z_{1:t}, u_{1:t}) \prod_{i=1}^{M} p(m_i \mid x_{0:t}, z_{1:t}) \end{aligned}$$

- Exploited in FastSLAM by Montemerlo et al., 2002.

Rao-Blackwellization for SLAM

SLAM Graph Revisited

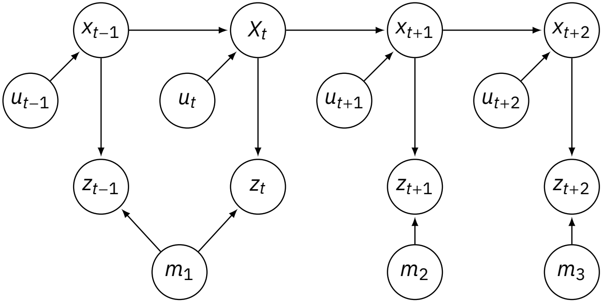


Figure: SLAM graph revisited

- Generative probabilistic model (dynamic Bayes network).
- The landmarks are independent of each other given we know the poses of the system (i.e., landmarks are all disconnected).

$$
\begin{aligned}
p(x_{0:t}, m_{1:M} \mid z_{1:t}, u_{1:t}) &= p(x_{0:t} \mid z_{1:t}, u_{1:t})\, p(m_{1:M} \mid x_{0:t}, z_{1:t}) \\
&= p(x_{0:t} \mid z_{1:t}, u_{1:t}) \prod_{i=1}^{M} \underline{p(m_i \mid x_{0:t}, z_{1:t})}
\end{aligned}
$$

- This dramatic simplification allows us to compute each landmark individually.
- Instead of a $2M \times 2M$ covariance matrix, we use M 2×2 covariance matrixes!

$$p(x_{0:t}, m_{1:M} \mid z_{1:t}, u_{1:t}) = p(x_{0:t} \mid z_{1:t}, u_{1:t})\, p(m_{1:M} \mid x_{0:t}, z_{1:t})$$
$$= \underline{p(x_{0:t} \mid z_{1:t}, u_{1:t})} \prod_{i=1}^{M} p(m_i \mid x_{0:t}, z_{1:t})$$

- Robot path posterior (localization problem).
- Particle filter similar to Monte Carlo Localization (MCL).

- Sample based representation for

$$p(x_{0:t} \mid z_{1:t}, u_{1:t}).$$

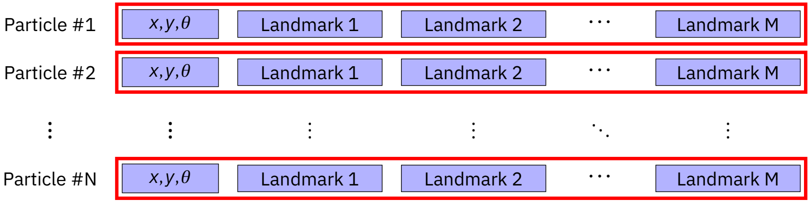
- Each sample is a path hypotesis:

$$\underbrace{x_0}_{\substack{\text{starting location,} \\ \text{typically } (0,0,0)}} \qquad \underbrace{x_1}_{\substack{\text{pose hypotesis} \\ \text{at time } t=1}} \qquad x_2 \qquad \dots$$

- Past samples of a pose are not revised or kept (always estimates the next trajectory of the robot).
- Utilizes the **motion model** to draw the next pose.

Principles of FastSLAM

- Proposed by Montemerlo et al. in 2002.
- Each sample maintains M 2-dimensional EKFs and estimates the location of each landmark.



1. **Do, for all particles** ($k = 1, \dots, N$):
   - **Retrieval:** retrieve a pose $x_{t-1}^{[k]}$ from the particle set Y_{t-1}.
   - **Prediction:** sample a new pose $x_t^{[k]} \sim p(x_t \mid x_{t-1}^{[k]}, u_t)$.
   - **Measurement Update:** for each observed feature identify the correspondence j for the measurement z_t^i and incorporate the measurement z_t^i into the corresponding EKF, by updating the mean $\mu_{j,t}^{[k]}$ and covariance $\Sigma_{j,t}^{[k]}$ of the jth landmark.
   - **Importance Weight:** calculate the importance weight $w_t^{[k]}$ for the new particle.
2. **Resampling:** sample, with replacement, N particles. Each particle is sampled with a probability proportional to $w_t^{[k]}$.

Calibration of the AlphaBot2

Calibration of the AlphaBot2

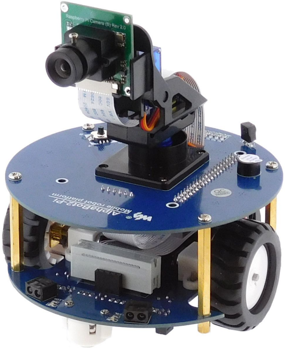


Figure: AlphaBot2

Velocity Motion Model

Given the absence of wheel odometry, we make use of the **velocity motion model**:

The velocity motion model assumes that we can control a robot through two velocities: a **rotational** and a **translational** velocity.

$$u_t = \begin{pmatrix} v_t \\ \omega_t \end{pmatrix}$$

- Positive rotational velocities induce a **left rotation**.
- Positive translational velocities correspond to **forward motion**.

Following the relationship between v_t and ω_t for an object moving on a circular trajectory, the resulting motion model is derived as follows:

$$\begin{pmatrix} x' \\ y' \\ \theta' \end{pmatrix} = \begin{pmatrix} x \\ y \\ \theta \end{pmatrix} + \begin{pmatrix} -\frac{\hat{v}}{\hat{\omega}} \sin\theta + \frac{\hat{v}}{\hat{\omega}} \sin(\theta + \hat{\omega}\Delta t) \\ \frac{\hat{v}}{\hat{\omega}} \cos\theta - \frac{\hat{v}}{\hat{\omega}} \cos(\theta + \hat{\omega}\Delta t) \\ \hat{\omega}\Delta t + \hat{\gamma}\Delta t \end{pmatrix}$$

Where $\hat{v}$ and $\hat{\omega}$ model **real motion!**

Robot motion is subject to **noise**. The true velocity, equals the commanded velocity plus some small, additive error:

$$\begin{pmatrix} \hat{v} \\ \hat{\omega} \end{pmatrix} = \begin{pmatrix} v \\ \omega \end{pmatrix} + \begin{pmatrix} \varepsilon_{\alpha_1 |v| + \alpha_2 |\omega|} \\ \varepsilon_{\alpha_3 |v| + \alpha_4 |\omega|} \end{pmatrix}$$

Here, ε_b is a zero-mean error variable with variance b, given by **robot-specific error parameters**: $\alpha_1, \alpha_2, \alpha_3, \alpha_4$.

To ensure the model accounts for **non-degenerate** pose estimation, we assume a rotation $\hat{\gamma}$ occurs upon the robot reaching its destination:

$$\theta' = \theta + \hat{\omega}\Delta t + \hat{\gamma}\Delta t \quad \rightarrow \quad \hat{\gamma} = \varepsilon_{\alpha_5|v|+\alpha_6|\omega|}$$

Where α_5 and α_6 are robot-specific error parameters.

Camera Calibration

Purpose

Essential for precision in robotics and computer vision to enhance image-based measurement accuracy.

- **Camera Matrix:** Focal lengths and optical center for 2D to 3D point conversion.
- **Distortion Coefficients:** Correct lens distortion, improving image accuracy.
- **Rectification Matrix:** Aligns stereo images for consistent point correspondence.
- **Projection Matrix:** Transforms 3D coordinates to 2D image planes.

Calibration of the AlphaBot2

Camera Calibration Overview

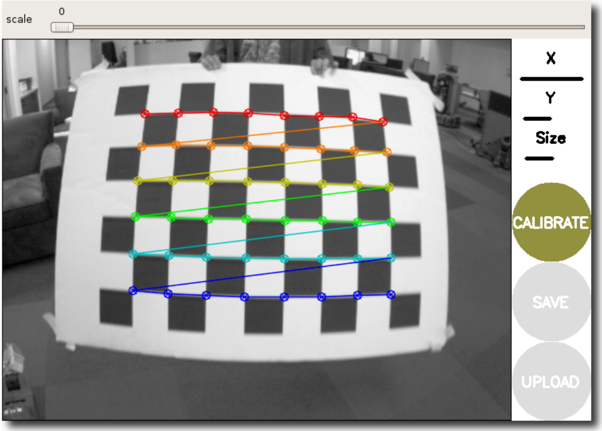


Figure: Camera calibration procedure (taken from the ROS wiki)

Acquired Sensor Data

Acquired Sensor Data

ArUco's relative position to the robot (`aruco_detect`):

- https://youtu.be/1Z3sY5O3PsA

Extra

Figure: AlphaBot2 components

Figure: Building the robot

Questions

Thank You!