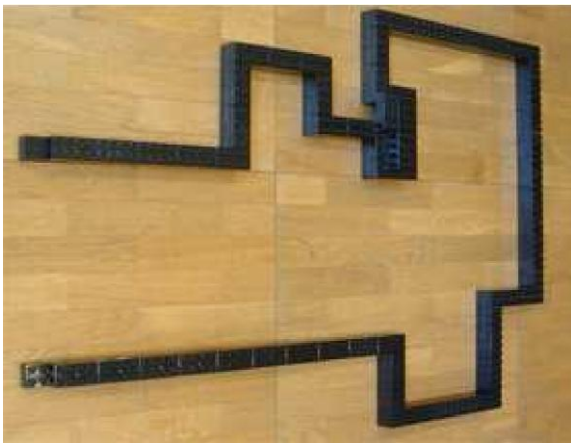
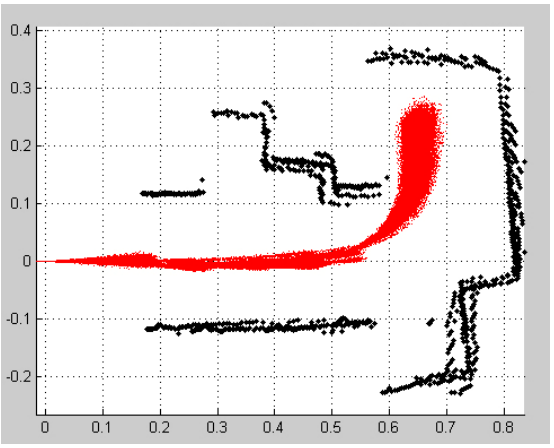
Exercise 4: Particle Filter Based Simultaneous Localization And Mapping

Michael Bloesch, CLA E 32.1, bloeschm@ethz.ch
Margarita Chli, CLA E 16.1, margarita.chli@mavt.ethz.ch

Version 4.0, May 03, 2012



Maze



Particle filter in action

Introduction

In this exercise, you will get introduced to the important topic in mobile robotics of Simultaneous Localization And Mapping (SLAM).
Localization is important for a robot in order to estimate its position with respect to a known environment. When the robot, however, explores a previously unseen environment, it needs to acquire a representation of this space by mapping it.

The SLAM problem, aims to tackle both estimation procedures simultaneously: SLAM is employed on mobile robots to enable map-building of their previously unknown workspace, while keeping track of the current robot position, using only feeds from onboard sensors. The challenge lies in coping with the uncertainties inevitably occurring when dealing with real-world sensor data for acquiring robust estimates of the scene and the robot's movement. As the robot moves away from its start location, the estimate of the trajectory travelled becomes increasingly inaccurate; therefore every feature initialized from the current location and added to the map, in turn, has some uncertainty associated to its estimated position in the 3D SLAM map. Accumulation of large errors can cause irrecoverable drift of estimates, leading to tracking/mapping failures. A good SLAM technique needs to take into account these uncertainties in order to provide an informed estimate of the robot location and its environment.

A standard SLAM pipeline consists of multiple parts; Landmark extraction, data association, state estimation, state update, and landmark update. Nowadays, there are well-known solutions for solving the SLAM problem, including Kalman filters, particle filters and key-frame based solutions – the most important ones are discussed in the associated SLAM lecture.

In this exercise, we will look into particle filter based SLAM (namely the FastSLAM technique) using range data retrieved by the "virtual laser" developed in Exercise 3.

Exercise outline:

This exercise is divided in two parts.

1. **PART A:** Complete and solve a part of the SLAM algorithm in MATLAB.

2. **PART B:** This is a practical SLAM example, where you will record a log of the environment and run the algorithm.

A small summary of the proposed SLAM algorithm is given at the end of this document.

Part A: Working on Pre-recorded Data

Detecting features

The input of the SLAM algorithm is **odometry data** (also called 'control inputs') and observable features (landmarks in the scene). Here, we will use corners as landmarks, detected as endpoint segments of lines extracted from range data retrieved using the "virtual laser" device.

Algorithm overview

We can reduce the corner detection algorithm in two steps:

- Line extraction
- Endpoint collection

Line extraction

A range scan describes a 2D slice of the environment. Points in a range scan are specified in the polar coordinate system `(ρi, θi)` with the origin at the location of the sensor. It is common in literature to assume that the noise on measurements follows a Gaussian distribution with zero mean, some range variance and negligible angular uncertainty.
We choose to express a line in polar coordinates (see Fig 2):

$$x \cos \alpha + y \sin \alpha = r \quad (1)$$

where $-\pi < \alpha <= \pi$ is the angle between the x-axis and `r >= 0` is the distance of the line to the origin (perpendicular to the line); `(x, y)` are the Cartesian coordinates of a point lying on the line.

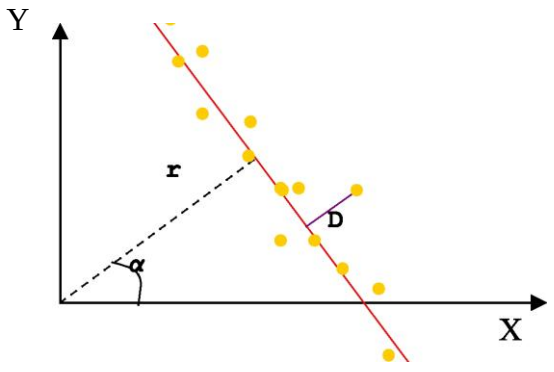


Fig 2: Fitting line parameters: D is the fitting error we aim to minimize expressing a line with polar parameters `(r, α)`

We employ the popular "Spit-and-Merge" line extraction algorithm to identify the best line segments for the obtained range measurements (points).

Algorithm 1: *Split-and-Merge*
1. Initially: set s_1 consists of all N points. Insert s_1 to the list $\mathcal{L}$. Set index $i=1$
2. Fit a line to the next set s_i in $\mathcal{L}$
3. Detect the point P with the maximum distance D to the line
4. **if** D is less than a threshold **then** continue to step 2
5. **else** split s_i at P into s_{i1} and s_{i2}, replace s_i , in $\mathcal{L}$, by s_{i1} and s_{i2}. Continue to step 2
6. When all sets (segments) in $\mathcal{L}$ have been checked, merge collinear segments.

EXERCISE 1: Algorithm in MATLAB

A test script has been written to test corner extraction using the pre-recorded data:

TEST_CORNER.M

Edit the file and change the filename log you need to load.

The most important part is the call to the function `extract_beacon4`,

```
z = extract_beacon4 (rho, theta);
```

which takes as input the range measurements in polar coordinates and returns corner coordinates.

The Split-and-Merge algorithm is implemented inside the function. A crucial part of this function is the line fitting step.

Edit FITLINE.M and follow the instructions to complete the mathematical formula for computing line regression (line fitting) using a set of points in Cartesian coordinates. The aim of the function is to minimize the mean squared error of:

```
D^2 = sum_i{ r - x_i cos α - y_i sin α }
```

Where `(x_i ,y_i)` are the input points in Cartesian coordinates. The solution of `(r, α)` can be found imposing:

```
d(D^2) / dr = 0   and   d (D^2) / da = 0
```

That is:

```
nom   = -2 sum_i{ (x_i - x_c) * (y_i - y_c)}
denom = sum_i{ (y_i - y_c) ^2 - (x_i - x_c)^2}
α = 0.5 * atan2 (nom, denom)
```

Where `(x_i ,y_i)` are the input points in Cartesian coordinates, `(x_c, y_c)` are the coordinates of the centroid (see instructions in M-file).
In order to solve for `r` consider the equation (1) and a point that will surely lie on the line (which one is it?).

Now that the lines are correctly fitted, the algorithm performs Split-and-Merge and extracts the endpoints of each segment.

Then **run** `TEST_CORNER.m` to check if the code is correctly completed (see the figure below to have a qualitative reference of what the line extractor does).

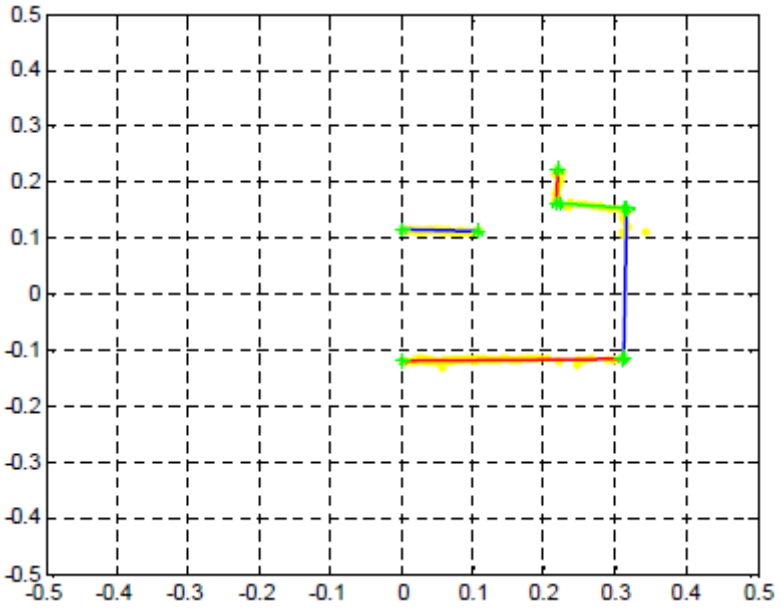


Fig 3: Corners in green are endpoints of extracted lines (various colors) on range data

Particle filter SLAM in MATLAB

An implementation of the particle filter SLAM, FastSLAM, is readily provided. Two reference systems are used: the robot reference system `(X_r, Y_r)` and the world reference system `(X, Y)`:

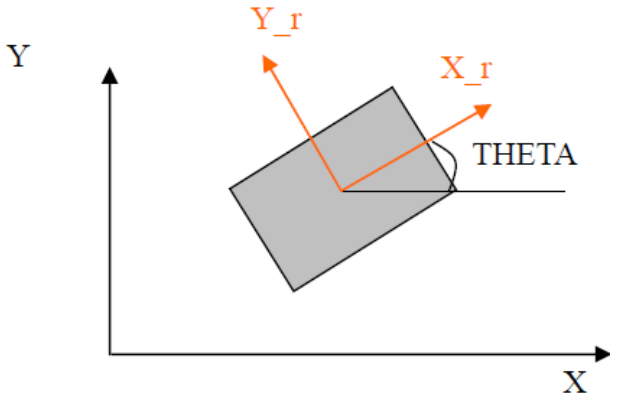


Fig:Reference systems

The range measurements are given with respect to the robot reference system `(X_r, Y_r)`. The odometry values represent the displacement in the time interval `dt` of the robot with respect to the world reference frame. The SLAM algorithm is solved in the world reference frame `(X, Y)` using as input odometry and laser measurements.

EXERCISE 2: Algorithm in MATLAB

The core cycle of the algorithm is developed in the file `fastslam1_sim.m`
A brief explanation of the main loop (line 38) is the following:

for every line of the log file

- **if** the logfile line is an odometry entry **then**
  - `predict` the new state of each particle
    [Diffuse the particle pose in the <x ,y, theta> space given the odometry covariance matrix]
- **end**
- **if** the logfile line is a laser rangefinder entry **then**
  - **for** each particle in the particle set,
    - extract corners from the range data
    - perform data association
      [Decide if the corner detected is a new feature for the particle or an observation of a previous one]
    - **if** the corner is an observation of a feature **then**
      - compute the weight of the particle given the observation
      - update the feature estimate
        [The feature estimate is the position <xf, yf> and its covariance matrix to define the uncertainty]
    - **end**
    - **if** the corner is a new feature **then**
      - Add the feature to the particle
    - **end**
    - Resample the particle set if the particle set has enough *confidence*
  - **end**
- **end**

end

The first step of this algorithm is used to make predictions on how the robot has moved

given the odometry information since the last estimation iteration.

Edit the file `predict_odo.m` and complete the mathematical operations following the instructions. Consider that the variable `xv` contains the last state of the particle (X, Y, THETA) in world reference frame.
Furthermore, consider that the odometry gives information about:

- the distance traveled `dr` in the time `dt` since the last position estimate
- the angle change `dth during` the time `dt` since the last orientation estimate

Changing SLAM parameters (Read the Particle filter based SLAM overview)

It's important to note that the FastSLAM approach estimates independently the pose of the robot and the features using a particle filter. Intuitively, each particle represents a "possible" robot position and its map of the environment.
For this section, you will change the main parameters of the SLAM algorithm using the file `control_panel.m`. To execute FastSLAM with the desired recorded data, **modify** the line that loads the dataset and then run `control_panel`: a window will appear with the map retrieved from the best particles with the best estimates. The red dots represent the actual particles, while the black dots correspond to the range measurements. Note that the particle set shrinks when a re-sampling occurs (only the best particles are maintained and weights are reset).
If the robot proceeds without observation, the uncertainty increases. What will happen to the spatial distribution of the particle set?

a) *Number of particles* (in `PARAMS.NPARTICLES`)
   Set the parameter to **5, 100, 1000, etc**. What is the influence of the number of particles on the final result? What is the trade-off you have to think about when choosing a value for this parameter?

b) *Variance of the odometry* (in `sigmaX` in `sigmaTH`)
   Try to increase / decrease the variance. What effects does it have on the final map? Which are the possible sources of the robot's odometric uncertainty?

c) *Variance of the range sensor* (in `sigmaR sigmaB`)
   Try to increase / decrease the variance. What effects does it have on the final map? Which are the possible sources of the sensor uncertainty?

Part B: Working on real data

In order to get a better feel of the algorithm, here, you will work with real data that you grab with your range sensor. Firstly, you have to **calibrate** the omnidirectional camera, as you did in Exercise 3, adjusting the focus and tilt angle of the camera carefully. Make sure that the cabling of the robot is not present in the field of view of the sensor to avoid interference with the obtained range data.
As soon as these basic steps are completed, please follow the information in Annex A for steering the robot and logging data.

Now you are ready to take a robot, mount a small maze on the paper sheet, record a log and then run FastSLAM on your data.

Modifying the number of scan points

By modifying the number of scan points, you influence the amount of data input to the corner extractor and to the SLAM algorithm to build a map, using the datalogger (Annex A). What are your comments?

Free play with the robot

You can now play freely with the robot and record data.

Try to turn in the maze or try to modify the maze configuration (wider, narrower, longer straight line before an intersection). Can you identify some situations where the resulting map looks really "bad"? Can you find some explanation?

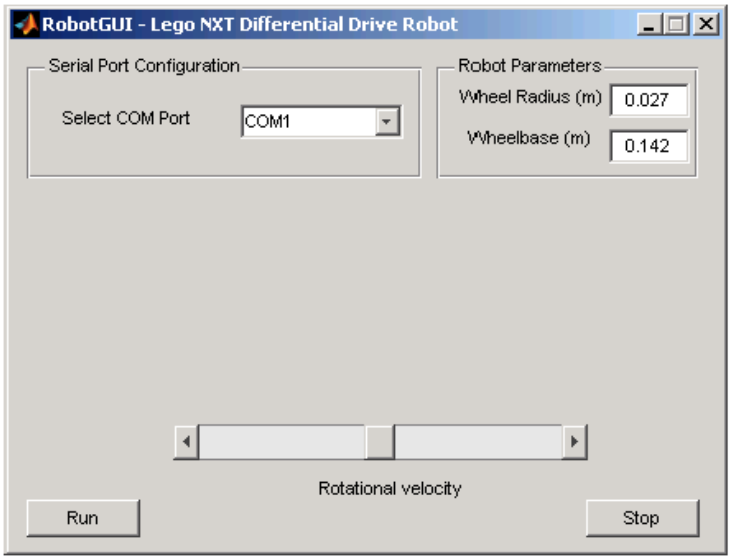
Annex A: Using the logger

When you unzip the file `logger.zip` you will find in the folder "logger" some files and some folders.
The most relevant files are `RobotGUI.m` and `Main.m`.

In `Main.m`, you will find some variables you can change :

- `N`: number of scans taken around the robot
- `T_b_S`: time lapse between two snapshots of scan values
- `F_O_U`: frequency at which the odometry is updated.
- `Log_name`: name of the log file

When you are satisfied with the variables, follow the detailed instruction in the `README.txt and` launch the GUI interface by typing `RobotGUI` on the Matlab command. You will get an interface similar to:



Typically, you must not change the wheel radius and wheelbase parameters.

Once everything is ready, you can press the Run button (the robot is connected via USB-cable, refer to the second exercise if you have some problems). The program will connect to the robot and launch the camera. You will be asked to click on the center of the image and on the outermost edge of the mirror for the camera calibration (in a reduced form compared to Exercise 3. If you are not satisfied with the output of the calibration, press 'n' and you will be asked for a new calibration. Otherwise, press 'y' and you will start to log the information the robot is gathering. WARNING: DO NOT CLOSE THE CALIBRATION WINDOW BEFORE ANSWERING y OR n!

In order for you to drive the robot, the two sliders on the GUI interface control the translational velocity (only forward) and the rotational velocity (in both directions).

When you are finished with the data logging, press the Stop button. The robot will stop and you will be able to process you log file for SLAM purposes.

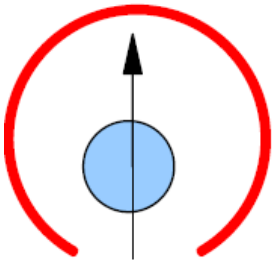
Annex B: The Log files

The log file is composed as follows:

- Each line corresponds to one set of data the robot recorded. The first value is the time stamp. The second value is a boolean indicating the nature of the data: 0 stands for odometry values (dX and dTheta) and 1 stands for scan values. Then comes the data.

- The length of a line depends on the number of scan values you wanted. If you wanted 180 scan values (i.e. one value each 2°), a line will be composed of 180 + 2 (time stamp & nature) values. The lines corresponding to the odometry values are filled with -1 values in order to have the same length as the lines corresponding to scan values.

Within the scan values, a 0 corresponds to a value being out of range of the "visual scanner". In the log file, the scans are listed as follows:



The span angle corresponds to 360/N (the number of scans taken around the robot).

Particle fitler SLAM

Overview

The SLAM problem, aims to tackle both estimation procedures simultaneously: SLAM is employed on mobile robots to enable map-building of their previously unknown workspace, while keeping track of the current robot position, using only feeds from onboard sensors. The challenge lies in coping with the uncertainties inevitably occurring when dealing with real-world sensor data for acquiring robust estimates of the scene and the robot's movement. As the robot moves away from its start location, the estimate of the trajectory travelled becomes increasingly inaccurate; therefore every feature initialized from the current location and added to the map, in turn, has some uncertainty associated to its estimated position in the 3D SLAM map. Accumulation of large errors can cause irrecoverable drift of estimates, leading to tracking/mapping failures. A good SLAM technique needs to take into account these uncertainties in order to provide an informed estimate of the robot location and its environment.
A standard SLAM pipeline consists of multiple parts; Landmark extraction, data association, state estimation, state update, and landmark update. Nowadays, there are well-known solutions for solving the SLAM problem, including Kalman filters, particle filters and key-frame based solutions – the most important ones are discussed in the associated SLAM lecture.

Sensor characteristics

A sensor is characterized principally by:

1. Noise
2. Dimensionality of the output
    a. Planar laser range finder (2D points)
    b. 3D laser range finder (3D point cloud)
    c. Camera features..
3. Frame of reference
    a. Laser/camera in robot frame
    b. GPS in earth coord. Frame
    c. Accelerometer/Gyros in inertial coord. frame

SLAM problem

The approach to solve the SLAM problem is addressed using probabilities. SLAM is usually explained by the conditional probability:

$$p(x_t, m \mid z_{1:t}, u_{1:t})$$

x_t = State of the robot at time t
m = Map of the environment t

$z_{1:t}$ = Sensor inputs from time *1* to *t*
$u_{1:t}$ = Control inputs from time *1* to *t*

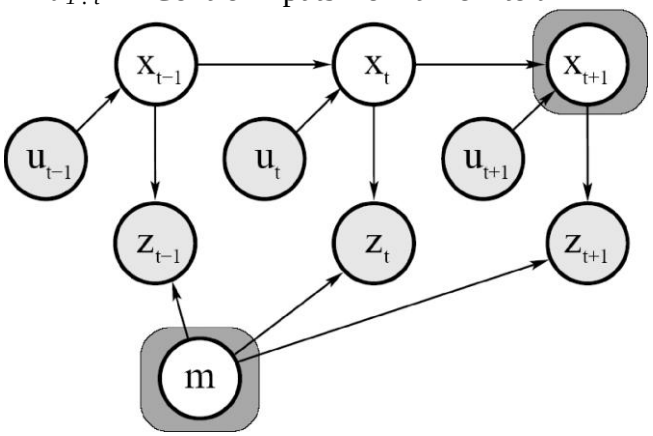


Fig l : Diagram of a SLAM technique

An online SLAM algorithm factorizes this formula to estimate the robot state at current time *t.*

$$p(x_t, m \mid z_{1:t}, u_{1:t}) = \int\int \ldots \int p(x_{1:t}, m \mid z_{1:t}, u_{1:t})\, dx_1\, dx_2 \ldots dx_{t-1}$$

FastSLAM approach

It solves the SLAM problem using particle filters. Particle filters are mathematical models that represent probability distribution as a set of discrete particles which occupy the state space.

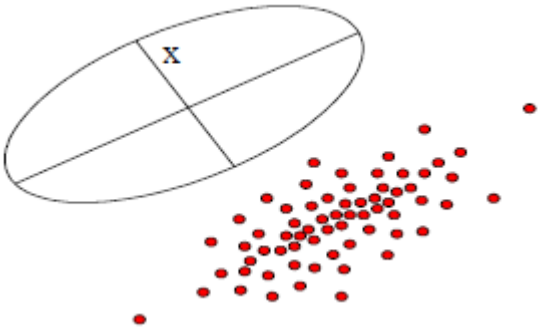


Fig 2: probability distribution (ellipse) as particle set (red dots)

Particle filter update

In the update step a new particle distribution is generated given motion model and controls applied.

a) For each particle:

1. Compare particle's prediction of measurements with actual measurements
2. Particles whose predictions match the measurements are given a high weight

b) Filter resampling:

Resample particles based on weight

Filter resampling

Assign each particle a weight depending on how well its estimate of the state agrees with the measurements and randomly draw particles from previous distribution based on weights creating a new distribution.

Formulation

Particle filter SLAM (or FastSLAM) decouples map of features from pose:

1. Each particle represents a robot pose
2. Feature measurements are correlated thought the robot pose

If the robot pose was known all of the features would be uncorrelated so it treats each pose particle as if it is the true pose, processing all of the feature measurements independently.

SLAM posterior — Robot path posterior

$$p(x_{1:t}, l_{1:m} \mid z_{1:t}, u_{0:t-1}) = p(x_{1:t} \mid z_{1:t}, u_{0:t-1}) \cdot p(l_{1:m} \mid x_{1:t}, z_{1:t})$$

Landmarks position

x: State of the robot

l: Landmarks in the map

u: Controls

z: Observations

It is possible to factorize the previous formula (Murphy l999) as:

$$p(x_{1:t}, l_{1:m} \mid z_{1:t}, u_{0:t-1})$$
$$= p(x_{1:t} \mid z_{1:t}, u_{0:t-1}) \cdot p(l_{1:m} \mid x_{1:t}, z_{1:t})$$
$$= p(x_{1:t} \mid z_{1:t}, u_{0:t-1}) \cdot \prod_{i=1}^{m} p(l_i \mid x_{1:t}, z_{1:t})$$

Robot path posterior (localization problem)

Conditionally independent landmark positions

Particle set

The robot posterior is solved using a Rao Blackwellized particle filtering using landmarks. Each landmark estimate is represented by a 2x2 EKF (Extended Kalman Filter). Each particle is "independent" (due the factorization) from the others and maintains the estimate of *M* landmark positions.