SLAM 2D – BE

Joan Solà [1] Ellon Paiva Mendes [2]

[1] Institut de Robòtica i Informàtica Industrial

[2] EasyMile

Way to proceed

- Get the `slam2d_BE.zip` from the LMS
- Unpack `slam2d_BE.zip`
- Launch Matlab
- Set Matlab path to `slam2d_BE` folder
- Launch `slam2d_BE.m`

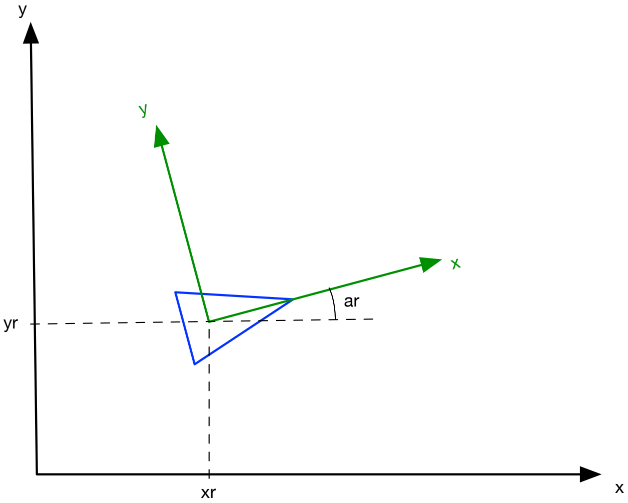
Notice that it **is not** complete: it only simulates a robot, but it does not do any SLAM.

It is your task to implement the SLAM code.

Carefully read the HELP lines at the beginning of `slam2d_BE.m`

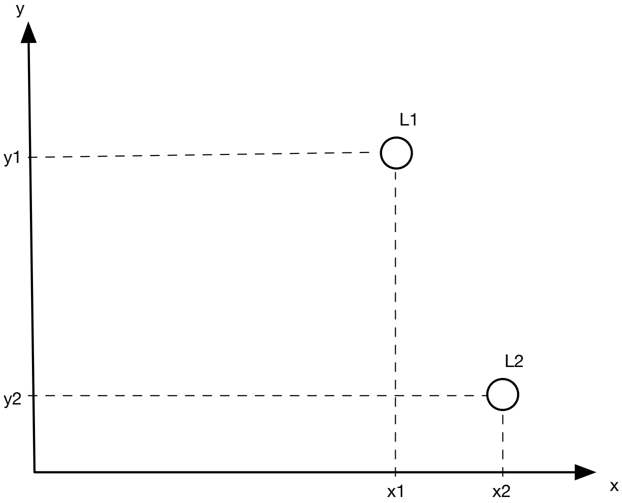
Robot state

$$x(r) = [xr, yr, ar]^\top$$



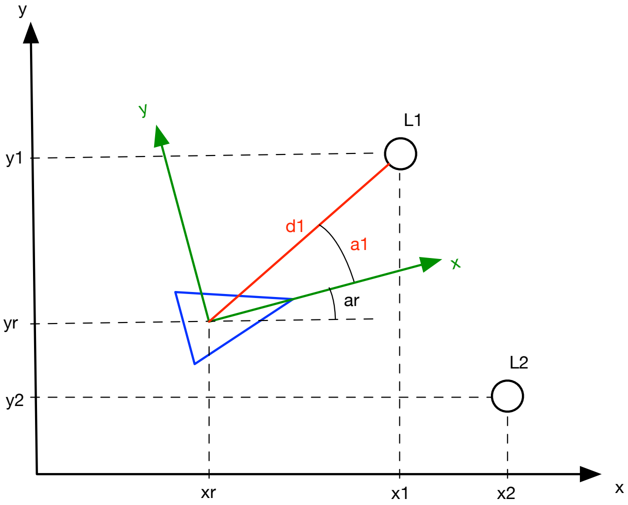
Landmark states

$$x(li) = [xi, yi]^\top$$



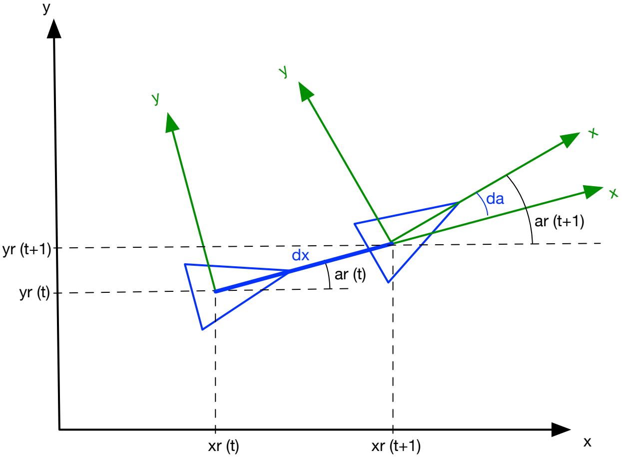
Landmark observations

$$Yi = [di, ai]^\top$$



Robot motion

$$u = [dx, da]^\top$$



States

- robot state $= \begin{bmatrix} x_r & y_r & a_r \end{bmatrix}^T$
- landmark state $= \begin{bmatrix} x_i & y_i \end{bmatrix}^T$
- map state $= \begin{bmatrix} x_r & y_r & a_r & x_1 & y_1 & x_2 & y_2 & \cdots & x_n & y_n \end{bmatrix}^T$
- estimated Gaussian map: $\mathbf{x} = \begin{bmatrix} x_r \\ \vdots \\ y_n \end{bmatrix}^T$ and $\mathbf{P} = \begin{bmatrix} P_{x_r x_r} & \cdots & P_{x_r y_n} \\ \vdots & \ddots & \vdots \\ P_{y_n x_r} & \cdots & P_{y_n y_n} \end{bmatrix}$

Pointers

Access the Gaussian map through *pointers*:

- Pointes are row-vectors of integers used as indices
- Ex: if $\mathbf{r} = \begin{bmatrix} 1 & 2 & 3 \end{bmatrix}$ is a pointer, then:
  $$\mathbf{x}(\mathbf{r}) = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} \text{ and } \mathbf{P}(\mathbf{r},\mathbf{r}) = \begin{bmatrix} P_{11} & P_{12} & P_{13} \\ P_{21} & P_{22} & P_{23} \\ P_{31} & P_{32} & P_{33} \end{bmatrix}$$
- Use the following pointer names:
  - Robot : ‘`r`’ (size 3),
    ex. `r = [1,2,3]`
  - Landmark : ‘`l`’ for one landmark (size 2),
    ex. `l1 = [4,5], l2 = [6,7]`
  - Map : ‘`m`’ for all known landmarks,
    ex. `m = [l1,l2,...,ln] = [4,5,6,7]`
  - Robot and one landmark : ‘`rl`’,
    ex. if `l = [6,7]` then `rl = [r,l] = [1,2,3,6,7]`
  - Full state : ‘`rm`’ for robot and all known landmarks,
    ex. `rm = [r,m] = [r,l1,l2,...,ln] = [1,2,3,4,5,6,7]`

Map manager

Helps you to manage the map space

- `mm_query_space(n)` $\rightarrow$ returns pointer `fs` to `n` free spaces
- `mm_block_space(fs)` $\rightarrow$ block positions indicated in pointer `fs`
- `mm_free_space(fs)` $\rightarrow$ liberate positions indicated in pointer `fs`

Landmark manager

Helps you to manage the landmarks

- `lm_find_non_mapped_lmk()` → look for one non-mapped landmark
- `lm_associate_pointer_to_lmk(fs,i)` → associate free space pointer `fs` to landmark `i`
- `lm_lmk_pointer(i)` → recover pointer `l` for a landmark `i`
- `lm_all_lmk_pointers()` → recover pointers to all known landmarks
- `lm_forget_lmk(i)` → forget landmark `i`
- `lm_all_lmk_ids()` → recover the id of all known landmarks

Control Signal and Measurements

- Control signal $\mathbf{U} = \begin{bmatrix} \delta_x \\ \delta_a \end{bmatrix}$
- Motion perturbation:
  - std dev vector $\mathbf{q} = \begin{bmatrix} q_x \\ q_a \end{bmatrix}$ and covariance matrix $\mathbf{Q}$

- Landmark measurement $\mathbf{Y_i} = \begin{bmatrix} d_i \\ a_i \end{bmatrix}$
- Measurement noise:
  - std dev vector $\mathbf{s} = \begin{bmatrix} s_d \\ s_a \end{bmatrix}$ and covariance matrix $\mathbf{S}$

Simulator

`sim_simulate_one_step()` perform one step of simulation. It is already used where it should be called. <u>Do NOT use it again</u>!

Interface functions:

- `sim_get_control_signal()` $\rightarrow$ recover the current control signal $\mathbf{U}$
- `sim_get_lmk_measurement(i)` $\rightarrow$ recover measurement $\mathbf{Y_i}$ to landmark `i`
- `sim_get_initial_robot_pose()` $\rightarrow$ recover the initial pose of the robot