

Module Interface Specification for 2D Localizer

Aliyah Jimoh

April 18, 2025

1 Revision History

Date	Version	Notes
2025/03/19	1.0	Initial Draft
2025/04/18	1.1	Implement Feedback

2 Symbols, Abbreviations and Acronyms

See SRS Documentation at <https://github.com/AliyahJimoh/2D-Localizer/blob/main/docs/SRS/SRS.pdf>. The symbols used in this document are mentioned below.

Symbol	Description
map	String of the map image's name
N	Number of beacons used
M	Number of FMs used
p	Number of positions the robot has in its trajectory
path	$\mathbb{R}^{p \times 3}$ matrix of the ground truth trajectory
a	$\mathbb{R}^{N \times 2}$ matrix of beacon coordinates
C	$\mathbb{R}^{2 \times 2}$ CRLB
$\mathcal{I}(\hat{\mathbf{x}})$	$\mathbb{R}^{2 \times 2}$ FIM
\mathbf{T}_{mf}	$\mathbb{R}^{M \times 3}$ pose of fiducial marker in map frame (x, y, θ)
\mathbf{T}_{rf}	\mathbb{R}^3 pose of fiducial marker in robot frame (x, y, θ)
$\tilde{\mathbf{d}}$	\mathbb{R}^N vector of a set of range measurements
$\tilde{\mathbf{D}}$	$\mathbb{R}^{p \times N}$ matrix of range measurements in all positions
$\hat{\mathbf{x}}$	\mathbb{R}^3 estimated robot pose
i	Index of fiducial marker
j	Index of beacon
t	Index of robot position along the trajectory
x	\mathbb{R} x coordinate of robot
y	\mathbb{R} y coordinate of robot
θ	\mathbb{R} orientation of robot (radians)
η	Gaussian sensor noise
ϕ	Angle of FM relative to robot (used for FOV filtering)
σ^2	\mathbb{R}^N vector of range noise variances

Table 1: Symbol Definitions Used in Access Routines

Contents

1	Revision History	i
2	Symbols, Abbreviations and Acronyms	ii
3	Introduction	1
4	Notation	1
5	Module Decomposition	1
6	MIS of Control Module	2
6.1	Module	2
6.2	Uses	2
6.2.1	Exported Constants	2
6.2.2	Exported Access Programs	2
6.3	Semantics	3
6.3.1	State Variables	3
6.3.2	Environment Variables	3
6.3.3	Assumptions	3
6.3.4	Access Routine Semantics	3
7	MIS of GTSAM Module	5
7.1	Module	5
7.2	Uses	5
7.3	Syntax	5
7.3.1	Exported Constants	5
7.3.2	Exported Types	5
7.3.3	Exported Access Programs	5
7.4	Semantics	6
7.4.1	State Variables	6
7.4.2	Environment Variables	6
7.4.3	Assumptions	6
7.4.4	Access Routine Semantics	6
8	MIS of Input Format Module	9
8.1	Module	9
8.2	Uses	9
8.3	Syntax	9
8.3.1	Exported Constants	9
8.3.2	Exported Access Programs	9
8.4	Semantics	9
8.4.1	State Variables	9

8.4.2	Environment Variables	9
8.4.3	Assumptions	10
8.4.4	Access Routine Semantics	10
9	MIS of Simulation Module	11
9.1	Module	11
9.2	Uses	11
9.3	Syntax	11
9.3.1	Exported Constants	11
9.3.2	Exported Access Programs	11
9.4	Semantics	11
9.4.1	State Variables	11
9.4.2	Environment Variables	11
9.4.3	Assumptions	11
9.4.4	Access Routine Semantics	11
10	MIS of Localization Module	13
10.1	Module	13
10.2	Uses	13
10.3	Syntax	13
10.3.1	Exported Constants	13
10.3.2	Exported Access Programs	13
10.4	Semantics	13
10.4.1	State Variables	13
10.4.2	Environment Variables	13
10.4.3	Assumptions	13
10.4.4	Access Routine Semantics	13
11	MIS of Accuracy Evaluation Module	15
11.1	Module	15
11.2	Uses	15
11.3	Syntax	15
11.3.1	Exported Constants	15
11.3.2	Exported Access Programs	15
11.4	Semantics	15
11.4.1	State Variables	15
11.4.2	Environment Variables	15
11.4.3	Assumptions	15
11.4.4	Access Routine Semantics	15
12	MIS of Output Module	17
12.1	Module	17
12.2	Uses	17

12.3	Syntax	17
12.3.1	Exported Constants	17
12.3.2	Exported Access Programs	17
12.4	Semantics	17
12.4.1	State Variables	17
12.4.2	Environment Variables	17
12.4.3	Assumptions	17
12.4.4	Access Routine Semantics	18
13	MIS of Plotting Module	19
13.1	Module	19
13.2	Uses	19
13.3	Syntax	19
13.3.1	Exported Constants	19
13.3.2	Exported Access Programs	19
13.4	Semantics	19
13.4.1	State Variables	19
13.4.2	Environment Variables	19
13.4.3	Assumptions	19
13.4.4	Access Routine Semantics	20
13.4.5	Local Functions	20

3 Introduction

The following document details the Module Interface Specifications for 2D Localizer, a program that implements various sensors to help localize mobile robots on a 2D plane in enclosed environments.

Complementary documents include the System Requirement Specifications and Module Guide. The full documentation and implementation can be found at <https://github.com/AliyahJimoh/2D-Localizer>.

4 Notation

The structure of the MIS for modules comes from Hoffman and Strooper (1995), with the addition that template modules have been adapted from Ghezzi et al. (2003). The mathematical notation comes from Chapter 3 of Hoffman and Strooper (1995). For instance, the symbol $:=$ is used for a multiple assignment statement and conditional rules follow the form $(c_1 \Rightarrow r_1 | c_2 \Rightarrow r_2 | \dots | c_n \Rightarrow r_n)$.

The following table summarizes the primitive data types used by 2D Localizer.

Data Type	Notation	Description
boolean	\mathbb{B}	a true or false value True, False
character	char	a single symbol or digit
dictionary	dict	a key-value data structure where keys map to values of different types
integer	\mathbb{Z}	a number without a fractional component in $(-\infty, \infty)$
list	list	An ordered collection of items of different or similar types
real	\mathbb{R}	any number in $(-\infty, \infty)$
string	String	more than one symbol put together

The specification of 2D Localizer uses some derived data types: sequences, strings, and tuples. Sequences are lists filled with elements of the same data type. Strings are sequences of characters. Tuples contain a list of values, potentially of different types. In addition, 2D Localizer uses functions, which are defined by the data types of their inputs and outputs. Local functions are described by giving their type signature followed by their specification.

5 Module Decomposition

The following table is taken directly from the Module Guide document for this project.

Level 1	Level 2
Hardware-Hiding Module	
	Input Format Module
	Simulation Module
	Output Module
Behaviour-Hiding Module	Localization Module
	Control Module
	Accuracy Evaluation Module
Software Decision Module	GTSAM Module
	Plotting Module

Table 2: Module Hierarchy

6 MIS of Control Module

6.1 Module

main

6.2 Uses

- Input Format Module (Section 8)
- Localization Module (Section 10)
- Accuracy Evaluation Module (Section 11)
- Plotting Module (Section 13)
- Output Module (Section 12)
- Python multiprocessing Library (Queue, Process)

6.2.1 Exported Constants

None

6.2.2 Exported Access Programs

Name	In	Out	Exceptions
main	-	-	-

6.3 Semantics

6.3.1 State Variables

- `data_queue`: Queue that stores the estimated pose of the current position and passes it to the Output Module's environmental variable.

6.3.2 Environment Variables

None

6.3.3 Assumptions

None

6.3.4 Access Routine Semantics

`main()`:

- `transition`: Modifies the shared queue ('`data_queue`') with each set of range and camera measurements.

Get Data

`input = InputData()` *# Abstract Data Type from Input Format Module*

Start the Output Data

`data_queue = Queue()` *# Part of multiprocessing library*

`process = Process(target=run_gui, args=[data_queue])` *# Runs the GUI (Output Module). It is created with run_gui as the target function and data_queue as its input argument.*

`process.start()`

`m = p` *# Number of positions the robot has in the map*

Getting estimated pose for each set of measurements

`for t in range(1,m):`

`$\hat{\mathbf{x}}$:= localize(a, T_{mf} , $\tilde{\mathbf{D}}[t, :]$, path[$t, :$])`

Computing FIM & CRLB

```

fim = compute_fim( $\hat{\mathbf{x}}$ ,  $\mathbf{a}$ , variances( $\boldsymbol{\sigma}^2$ ))

crlb = compute_crlb(fim) # Will be printed

update_trajectory( $\hat{\mathbf{x}}$ )

data_queue.put((t,  $\hat{\mathbf{x}}$ .x(),  $\hat{\mathbf{x}}$ .y(),  $\hat{\mathbf{x}}$ .theta()))

# Plot on the map

plot_localization_live( $\mathbf{a}$ ,  $T_{mf}$ , map, path, map_size)

```

7 MIS of GTSAM Module

The GTSAM Module provides wrapper access the Georgia Tech Smoothing and Mapping (GTSAM) library using simplified and consistent Python interfaces. GTSAM is used for solving estimation problems using factor graphs which are a way to represent relationships between variables using "factors" (pieces of information gotten from sensors). More information about the API can be found at <https://gtsam.org/dxygen/index.html>

7.1 Module

gtsam_wrapper

7.2 Uses

None

7.3 Syntax

7.3.1 Exported Constants

None

7.3.2 Exported Types

Data Type	Notation	Description
factor	Factor	a constraint in a factor graph that relates variables
factor graph	Graph	a collection of factors defining an optimization problem
noise model	Model	a model that defines uncertainty in a measurement
values	Values	a container that stores variable estimates in a factor graph

7.3.3 Exported Access Programs

Name	In	Out	Exceptions
Pose2	$x : \mathbb{R}, y : \mathbb{R}, \theta : \mathbb{R}$	\mathbb{R}^3	-
Point2	$x : \mathbb{R}, y : \mathbb{R}$	\mathbb{R}^2	-
symbol	char: char, int: \mathbb{Z}	String	-
NonlinearFactorGraph	-	Graph	-
PriorFactorPose2	$key : \mathbb{Z}, \mathbf{pose} : \mathbb{R}^3, noise : Model$	Factor	-
PriorFactorPoint2	$key : \mathbb{Z}, \mathbf{pose} : \mathbb{R}^2, noise : Model$	Factor	-
RangeFactor2D	$key1 : \mathbb{Z}, key2 : \mathbb{Z}, d : \mathbb{R}, noise : Model$	Factor	-
BetweenFactor	$key1 : \mathbb{Z}, key2 : \mathbb{Z}, between : Pose2, noise : Model$	Factor	-
noiseModel_Isotropic_Sigma	$dim : \mathbb{Z}, \sigma : \mathbb{R}$	Model	-
LevenbergMarquardtOptimizer	$graph : Graph, values : Values$	Values	-
Values	-	Values	-
insert	$values : Values, key : \mathbb{Z}, value : Pose2 \text{ or } Point2$	-	-
atPose2	$result : Values, key : \mathbb{Z}$	\mathbb{R}^3	-
compose	$T_{mf} : \mathbb{R}^3, T_{rf} : \mathbb{R}^3$	\mathbb{R}^3	-
inverse	$T_{rf} : \mathbb{R}^3$	\mathbb{R}^3	-

7.4 Semantics

7.4.1 State Variables

None

7.4.2 Environment Variables

None

7.4.3 Assumptions

None

7.4.4 Access Routine Semantics

Pose2(x, y, θ):

- output: $out := [x, y, \theta]$ (A 2D pose with orientation)
- exception: None

Point2(x, y):

- output: $out := [x, y]$ (2D position)

- exception: None

`symbol(char, int):`

- output: *out* := $x1(pose), a1, a2, a3(beacons)$
- exception: None

`NonlinearFactorGraph():`

- output: *out* := An empty factor graph
- exception: None

`PriorFactorPose2(key, pose, noise_model):`

- output: *out* := Factor (A prior factor on a 2D pose)
- exception: None

`PriorFactorPoint2(key, point, noise_model):`

- output: *out* := Factor (A prior factor on a 2D point)
- exception: None

`RangeFactor2D(key1, key2, measured, noise_model):`

- output: *out* := Factor (A range factor between two keys)
- exception: None

`BetweenFactor(key1, key2, measured, noise_model):`

- output: *out* := Factor (A factor that enforces a relative transformation between two variables)
- exception: None

`noiseModel_Isotropic_Sigma(dim, σ):`

- output: *out* := Model (An isotropic noise model)
- exception: None

`LevenbergMarquardtOptimizer(graph, values):`

- output: *out* := Values (Optimized results from factor graph)
- exception: None

`Values():`

- output: $out := \text{Values}$ (An empty values container)
- exception: None

$\text{insert}(\text{Values}, key, value):$

- transition: Adds point/pose into a Values variable according to its id (key)
- exception: None

$\text{atPose2}(\text{result}, key):$

- output: $out := \hat{\mathbf{x}}$
- exception: None

$\text{compose}(T_{mf}, T_{rf}):$

- output: $out := T_{mr}$ (The composition of two poses)
- exception: None

$\text{inverse}(T_{rf}):$

- output: $out := T_{fr}$
- exception: None

8 MIS of Input Format Module

8.1 Module

input_format

8.2 Uses

- Simulation Module (Section 9)

8.3 Syntax

8.3.1 Exported Constants

None

8.3.2 Exported Access Programs

These functions are methods of the ‘InputData’ class instance, which must be initialized before use (example shown in section 6.3.4).

Name	In	Out	Exceptions
load_input	self	-	FileNotFoundError, ValueError
get_beacons	self	$\mathbb{R}^{N \times 2}$	-
get_fmMap	self	\mathbb{R}^3	-
get_mapSize	self	\mathbb{R}^2	-
get_trajectory	self	$\mathbb{R}^{p \times 3}$	-
get_map	self	String	-
get_ranges	self	\mathbb{R}^N	-
get_variances	self	\mathbb{R}^N	-

8.4 Semantics

8.4.1 State Variables

- input_file: String (represents the path to the user input file ‘user_input.yaml’)
- data: dict (storing parsed input data from YAML file).

8.4.2 Environment Variables

None

8.4.3 Assumptions

- The module will call on a pre-existing YAML file

8.4.4 Access Routine Semantics

load_input():

- transition: Reads the YAML input file and stores it in ‘self.data’.
- exception: FileNotFoundError if the input file is not detected and ValueError if the YAML file is formatted incorrectly

input.get_beacons():

- output: $out := \mathbf{a}$
- exception: None

get_fmMap():

- output: $out := \mathbb{R}^{M \times 3}$
- exception: None

get_map():

- output: $out :=$ String of picture’s name
- exception: None

get_mapSize():

- output: $out := [\text{length}, \text{width}]$ of map in meters (from user input)
- exception: None

get_trajectory():

- output: $out := \mathbb{R}^{p \times 3}$ matrix containing the robot’s full trajectory over each position (x, y, θ)
- exception: None

get_ranges():

- output: $out := \tilde{\mathbf{D}}$
- exception: None

get_variances():

- output: $out := \sigma^2$
- exception: None

9 MIS of Simulation Module

9.1 Module

simulation

9.2 Uses

None

9.3 Syntax

9.3.1 Exported Constants

None

9.3.2 Exported Access Programs

Name	In	Out	Exceptions
noisy_range	beacons: $\mathbb{R}^{N \times 2}$, variances: \mathbb{R}^N , trajectory: $\mathbb{R}^{p \times 3}$	$\tilde{d} : \mathbb{R}^{p \times N}$	None
fm_robot	trajectory: \mathbb{R}^3 , fm_map: \mathbb{R}^3	$T_{rf} : \mathbb{R}^3$	None
visible_fms	robot_pose: \mathbb{R}^3 , fm_map: $\mathbb{R}^{M \times 3}$	list (i : \mathbb{Z} , $T_{rf} : \mathbb{R}^3$)	None

9.4 Semantics

9.4.1 State Variables

None

9.4.2 Environment Variables

None

9.4.3 Assumptions

None

9.4.4 Access Routine Semantics

noisy_range(beacons, variances, trajectory):

extract robot positions

path := trajectory[:, :2] *# x, y positions only*

compute distances and apply Gaussian noise

$r = \text{path}[:, \text{None}, :] - \text{beacons}[\text{None}, :, :]$

$\eta = \sqrt{\text{variances}} * \mathcal{N}(0, 1)$

$\tilde{d} = \|r\| + \eta$

- output: $out := \tilde{d} \in \mathbb{R}^{p \times N}$ (noisy range measurements from robot to each beacon)
- exception: None

$\text{fm_robot}(\text{trajectory}, \text{fm_map})$:

extract robot and FM pose in map frame

$(x_r, y_r, \theta_r) = \text{trajectory}$

$(x_f, y_f, -) = \text{fm_map}$

transform FM pose to robot frame

$dx := x_f - x_r$

$dy := y_f - y_r$

$x_{rel} := \cos(\theta_r) \cdot dx + \sin(\theta_r) \cdot dy$

$y_{rel} := -\sin(\theta_r) \cdot dx + \cos(\theta_r) \cdot dy$

$\phi := \arctan 2(y_{rel}, x_{rel})$

- output: $out := T_{rf} \in \mathbb{R}^3$ (relative FM pose in robot frame)
- exception: None

$\text{visible_fms}(\text{robot_pose}, \text{fm_map}, \text{max_range}, \text{fov_angle})$:

initialize list of visible FMs

$\text{visible} := []$

check distance and FOV for each FM

for each (i, fm) in fm_map :

$(x_{rel}, y_{rel}, \phi) := \text{fm_robot}(\text{robot_pose}, fm)$

if $\|[x_{rel}, y_{rel}]\| \leq \text{max_range}$ and $0 \leq \phi \leq \text{fov_angle}$:

$\text{visible.append}((i, (x_{rel}, y_{rel}, \phi)))$

- output: $out := \text{visible of } (index, T_{rf} \in \mathbb{R}^3)$ (visible FMs in robot frame)
- exception: None

10 MIS of Localization Module

10.1 Module

localization

10.2 Uses

- GTSAM Module (Section 7)
- Simulation Module (Section 9)

10.3 Syntax

10.3.1 Exported Constants

None

10.3.2 Exported Access Programs

Name	In	Out	Exceptions
localize	$\mathbf{a} : \mathbb{R}^{N \times 2}, \mathbf{T}_{mf} : \mathbb{R}^3, \mathbf{T}_{rf} : \mathbb{R}^3, \tilde{\mathbf{d}} : \mathbb{R}^N$	\mathbb{R}^3	-

10.4 Semantics

10.4.1 State Variables

None

10.4.2 Environment Variables

None

10.4.3 Assumptions

None

10.4.4 Access Routine Semantics

localize($\mathbf{a}, \mathbf{T}_{mf}, \tilde{\mathbf{d}}, \text{initial_guess}$):

```
# initialize factor graph and robot symbol  
graph = NonlinearFactorGraph()  
robot_id = symbol("x", 1)
```

```

# add prior on robot pose
graph.add(PriorFactorPose2(robot_id, x0, prior_noise))

# add range factors to visible beacons
for each (j, position, d_j) in visible_beacons:
graph.add(RangeFactor2D(robot_id, symbol("a", j+1), d_j, range_noise))

# fix one beacon position for stability
graph.add(PriorFactorPoint2(symbol("a", 1), position_1, beacon_noise))

# add between factors from visible fiducial markers
for each fiducial i:
graph.add(BetweenFactor(robot_id, symbol("f", i+1), T_rf, pose_noise))
graph.add(PriorFactorPose2(symbol("f", i+1), T_mf, pose_noise))

# initialize estimates
insert robot, beacon, and fiducial guesses into Values()

# optimize with LM
result = LevenbergMarquardtOptimizer(graph, initial_estimates)
 $\hat{x} := \text{result.atPose2}(\text{robot\_id})$ 

```

- output: $out := \hat{\mathbf{x}} \in \mathbb{R}^3$
- exception: `ValueError` if estimation fails or result is not computable

11 MIS of Accuracy Evaluation Module

11.1 Module

accuracy

11.2 Uses

- Localization Module (Section [10](#))

11.3 Syntax

11.3.1 Exported Constants

None

11.3.2 Exported Access Programs

Name	In	Out	Exceptions
compute_fim	$\hat{\mathbf{x}} : \mathbb{R}^3, \mathbf{a} : \mathbb{R}^{N \times 2}, \boldsymbol{\sigma}^2 : \mathbb{R}^N$	$\mathbb{R}^{2 \times 2}$	-
compute_crlb	$\mathcal{I}(\hat{\mathbf{x}}) : \mathbb{R}^{2 \times 2}$	$\mathbb{R}^{2 \times 2}$	-

11.4 Semantics

11.4.1 State Variables

None

11.4.2 Environment Variables

None

11.4.3 Assumptions

- Noise variances are positive

11.4.4 Access Routine Semantics

compute_fim($\hat{\mathbf{x}}, \mathbf{a}, \boldsymbol{\sigma}^2$):

- output: $out := \mathcal{I}(\hat{\mathbf{x}})$ where $\mathcal{I}(\hat{\mathbf{x}})$ is a 2×2 Fisher Information Matrix (FIM) of the estimated pose, computed as:

$$\mathcal{I}(\hat{\mathbf{x}}) = \sum_{j=1}^N \frac{1}{\sigma_j^2} \frac{(\hat{\mathbf{x}} - \mathbf{a}_j)(\hat{\mathbf{x}} - \mathbf{a}_j)^T}{\|\hat{\mathbf{x}} - \mathbf{a}_j\|^2}$$

where $\hat{\mathbf{x}}$ only contains x and y (making it \mathbb{R}^2 so it can subtract)

- exception: None

`compute_crlb($\mathcal{I}(\hat{\mathbf{x}})$):`

- output: *out* := A 2×2 CRLB matrix, computed as:

$$\mathbf{C} = \mathcal{I}^{-1}$$

- exception: None

12 MIS of Output Module

12.1 Module

output

12.2 Uses

- Localization Module (Section [10](#))
- Python multiprocessing Library (Queue)

12.3 Syntax

12.3.1 Exported Constants

None

12.3.2 Exported Access Programs

Name	In	Out	Exceptions
update_table	-	-	-
run_gui	queue: Queue	-	-

12.4 Semantics

12.4.1 State Variables

- root: Tkinter root window
- tree: Treeview widget for table display
- data_queue: Shared queue received from Control Module

12.4.2 Environment Variables

- data_queue: A queue storing tuples of estimated pose data (time, x, y, theta).
- display_env: The OS-level display environment variable required to render the Tkinter GUI (e.g., ‘\$DISPLAY’ for Unix-based systems).

12.4.3 Assumptions

- The function ‘run_gui()’ is executed in a separate process to prevent a stalled execution.

12.4.4 Access Routine Semantics

update_table():

- transition: Retrieves the latest pose estimates from the queue and updates the Graphical User Interface (GUI) table.

run_gui(queue):

- transition: Initializes and runs the Tkinter GUI while continuously checking for pose updates.

Start GUI loop

root = tk.Tk() # Root GUI window

tree = ttk.Treeview(...) # Table structure for displaying (pose_num, x, y, θ)

Periodically check the shared queue

root.after(100, update_table) # 100 ms update interval

root.mainloop()

13 MIS of Plotting Module

13.1 Module

plot

13.2 Uses

- Localization Module (Section 10)

13.3 Syntax

13.3.1 Exported Constants

None

13.3.2 Exported Access Programs

Name	In	Out	Exceptions
plot_localization_live	$\mathbf{a} : R^{N \times 2}, \mathbf{T}_{mf} : R^3$, map: String (filepath), show: \mathbb{B}	-	-
update_trajectory	$\hat{\mathbf{x}} : R^3$	-	-

13.4 Semantics

13.4.1 State Variables

- trajectory: Global sequence updated by ‘update_trajectory()’ from Control Module (section 6).

13.4.2 Environment Variables

- plot_env: The Matplotlib interactive rendering backend required to run real-time plotting.

13.4.3 Assumptions

- ‘plot_localization_live()’ is run in an interactive Matplotlib session.
- ‘update_trajectory()’ is only called when valid estimated poses exist.
- ‘plot_env’ supports ‘plt.ion()’ and ‘plt.pause()’ for animation updates.
- localize()(Section 10) either returns a valid pose or raises an exception.

13.4.4 Access Routine Semantics

```
plot_localization_live(a, Tmf, map, path, map_size, show=True):  
# render map background and plot fixed elements  
img = plt.imread(map)  
plt.imshow(img, extent=[0, map_size[0], 0, map_size[1]])  
plt.plot(a[:, 0], a[:, 1], 'o') # beacons  
plt.plot(Tmf[:, 0], Tmf[:, 1], 'x') # fiducial markers  
plt.plot(path[:, 0], path[:, 1], '-') # ground truth trajectory  
  
# initialize plot elements  
robot_trajectory, = plt.plot([], [], 'r-')  
triangle_patch = Polygon([], closed=True, color='red')  
plt.gca().add_patch(triangle_patch)  
  
# begin animation loop  
ani = FuncAnimation(fig, update, interval=200)  
if show: plt.show()
```

- transition: Initializes and continuously updates a real-time localization plot.
- output: None (renders a live plot in Matplotlib)
- exception: RuntimeError if the Matplotlib backend does not support animation

update_trajectory($\hat{\mathbf{x}}$):

- transition: Called repeatedly by Matplotlib's 'FuncAnimation' to retrieve and render the latest estimated pose from the global 'trajectory'

13.4.5 Local Functions

```
update(frame):  
# update robot path and triangle orientation  
x, y,  $\theta$  := trajectory[frame]  
robot_trajectory.set_data(... up to frame ...)  
triangle_patch.set_xy(...) # rotated triangle
```

- transition: Retrieves the latest estimated pose from the trajectory and updates the visualization.

References

- Carlo Ghezzi, Mehdi Jazayeri, and Dino Mandrioli. *Fundamentals of Software Engineering*. Prentice Hall, Upper Saddle River, NJ, USA, 2nd edition, 2003.
- Daniel M. Hoffman and Paul A. Strooper. *Software Design, Automated Testing, and Maintenance: A Practical Approach*. International Thomson Computer Press, New York, NY, USA, 1995. URL <http://citeseer.ist.psu.edu/428727.html>.