prpy: Probabilistic Robot Localization Python Library

Release 0.1

Pere Ridao

Oct 01, 2023

CONTENTS

Probabilistic Robot Localization is Python Library containing the main algorithms explained in the **Probabilisitic Robot Localization** Book used in the **Probabilisitic Robotics** and the **Hands-on Localization** Courses of the **Intelligent Field Robotic Systems (IFRoS)** European Erasmus Mundus Master.

Note: This documentation is still under construction.

CHAPTER

ONE

API:

1.1 Pose Representation

1.1.1 Pose 3DOF

class prpy.**Pose3D**(*input_array*)

Bases: ndarray

Definition of a robot pose in 3 DOF (x, y, yaw). The class inherits from a ndarray. This class extends the ndarray with the $oplus$ and $ominus$ operators and the corresponding Jacobians.

oplus(*BxC*)

Given a Pose3D object *AxB* (the self object) and a Pose3D object *BxC*, it returns the Pose3D object *AxC*.

$$\mathbf{^A_{X_B}} = \begin{bmatrix} {}^Ax_B & {}^Ay_B & {}^A\psi_B \end{bmatrix}^T$$
$$\mathbf{^B_{X_C}} = \begin{bmatrix} {}^Bx_C & {}^By_C & {}^B\psi_C \end{bmatrix}^T$$

The operation is defined as:

$$\mathbf{^A_{X_C}} = \mathbf{^A_{X_B}} \oplus \mathbf{^B_{X_C}} = \begin{bmatrix} {}^Ax_B +^B x_C \cos({}^A\psi_B) -^B y_C \sin({}^A\psi_B) \\ {}^Ay_B +^B x_C \sin({}^A\psi_B) +^B y_C \cos({}^A\psi_B) \\ {}^A\psi_B +^B \psi_C \end{bmatrix} \tag{1.1}$$

Parameters
BxC – C-Frame pose expressed in B-Frame coordinates

Returns
C-Frame pose expressed in A-Frame coordinates

J_1oplus(*BxC*)

Jacobian of the pose compounding operation (eq. (1.1)) with respect to the first pose:

$$J_{1\oplus} = \frac{\partial {}^Ax_B \oplus^B x_C}{\partial {}^Ax_B} = \begin{bmatrix} 1 & 0 & -{}^Bx_C \sin({}^A\psi_B) -^B y_C \cos({}^A\psi_B) \\ 0 & 1 & {}^Bx_C \cos({}^A\psi_B) -^B y_C \sin({}^A\psi_B) \\ 0 & 0 & 1 \end{bmatrix} \tag{1.2}$$

The method returns a numerical matrix containing the evaluation of the Jacobian for the pose *AxB* (the self object) and the 2^{nd} posepose *BxC*.

Parameters
BxC – 2nd pose

Returns
Evaluation of the $J_{1\oplus}$ Jacobian of the pose compounding operation with respect to the first pose (eq. (1.2))

J_2oplus()

Jacobian of the pose compounding operation ((1.1)) with respect to the second pose:

$$J_{2\oplus} = \frac{\partial {}^{A}x_B \oplus {}^{B}x_C}{\partial {}^{B}x_C} = \begin{bmatrix} \cos({}^{A}\psi_B) & -\sin({}^{A}\psi_B) & 0 \\ \sin({}^{A}\psi_B) & \cos({}^{A}\psi_B) & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{1.3}$$

The method returns a numerical matrix containing the evaluation of the Jacobian for the 1^{st} posepose *AxB* (the self object).

Returns

Evaluation of the $J_{2\oplus}$ Jacobian of the pose compounding operation with respect to the second pose (eq. (1.3))

ominus()

Inverse pose compounding of the *AxB* pose (the self objetc):

$${}^{B}x_A = \ominus {}^{A}x_B = \begin{bmatrix} -{}^{A}x_B \cos({}^{A}\psi_B) - {}^{A}y_B \sin({}^{A}\psi_B) \\ {}^{A}x_B \sin({}^{A}\psi_B) - {}^{A}y_B \cos({}^{A}\psi_B) \\ -{}^{A}\psi_B \end{bmatrix} \tag{1.4}$$

Returns

A-Frame pose expressed in B-Frame coordinates (eq. (1.4))

J_ominus()

Jacobian of the inverse pose compounding operation ((1.1)) with respect the pose *AxB* (the self object):

$$J_{\ominus} = \frac{\partial \ominus {}^{A}x_B}{\partial {}^{A}x_B} = \begin{bmatrix} -\cos({}^{A}\psi_B) & -\sin({}^{A}\psi_B) & {}^{A}x_B \sin({}^{A}\psi_B) - {}^{A}y_B \cos({}^{A}\psi_B) \\ \sin({}^{A}\psi_B) & -\cos({}^{A}\psi_B) & {}^{A}x_B \cos({}^{A}\psi_B) + {}^{A}y_B \sin({}^{A}\psi_B) \\ 0 & 0 & -1 \end{bmatrix} \tag{1.5}$$

Returns the numerical matrix containing the evaluation of the Jacobian for the pose *AxB* (the self object).

Returns

Evaluation of the $J_{\ominus}$ Jacobian of the inverse pose compounding operation with respect to the pose (eq. (1.5))

1.1.2 Pose 4DOF

class prpy.**Pose4D**(*input_array*)

Bases: ndarray

Definition of a robot pose in 4 DOF (x, y, yaw). The class inherits from a ndarray. This class extends the ndarray with the $oplus$ and $ominus$ operators and the corresponding Jacobians.

oplus(*BxC*)

Given a Pose3D object *AxB* (the self object) and a Pose3D object *BxC*, it returns the Pose4D object *AxC*.

$$\begin{aligned} {}^{A}x_B &= \begin{bmatrix} {}^{A}x_B & {}^{A}y_B & {}^{A}z_B & {}^{A}\psi_B \end{bmatrix}^T \\ {}^{B}x_C &= \begin{bmatrix} {}^{B}x_C & {}^{B}y_C & {}^{B}z_C & {}^{B}\psi_C \end{bmatrix}^T \\ {}^{A}x_C &= {}^{A}x_B \oplus {}^{B}x_C = \begin{bmatrix} {}^{A}x_B + {}^{B}x_C \cos({}^{A}\psi_B) - {}^{B}y_C \sin({}^{A}\psi_B) \\ {}^{A}y_B + {}^{B}x_C \sin({}^{A}\psi_B) + {}^{B}y_C \cos({}^{A}\psi_B) \\ {}^{A}z_B + {}^{B}z_C \\ {}^{A}\psi_B + {}^{B}\psi_C \end{bmatrix} \end{aligned} \tag{1.6}$$

Parameters

BxC – C-Frame pose expressed in B-Frame coordinates

Returns
C-Frame pose expressed in A-Frame coordinates

J_1oplus(*BxC*)

Jacobian of the pose compounding operation (eq. (1.6)) with respect to the first pose:

$$J_{1\oplus} = \frac{\partial^A x_B \oplus^B x_C}{\partial^A x_B} = \begin{bmatrix} 1 & 0 & 0 & -^B x_C \sin(^A\psi_B) - ^B y_C \cos(^A\psi_B) \\ 0 & 1 & 0 & ^B x_C \cos(^A\psi_B) - ^B y_C \sin(^A\psi_B) \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{1.7}$$

Parameters

- **AxB** – first pose
- **BxC** – 2nd pose

Returns
$J_{1\oplus}$ Jacobian of the pose compounding operation with respect to the first pose (eq. (1.7))

J_2oplus()

Jacobian of the pose compounding operation ((1.6)) with respect to the second pose:

$$J_{2\oplus} = \frac{\partial^A x_B \oplus^B x_C}{\partial^B x_C} = \begin{bmatrix} \cos(^A\psi_B) & -\sin(^A\psi_B) & 0 & 0 \\ \sin(^A\psi_B) & \cos(^A\psi_B) & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{1.8}$$

Parameters
AxB – first pose

Returns
$J_{2\oplus}$ Jacobian of the pose compounding operation with respect to the second pose (eq. (1.8))

ominus()

Inverse pose compounding of the *AxB* pose (the self object):

$${}^B x_A = \ominus^A x_B = \begin{bmatrix} -^A x_B \cos(^A\psi_B) - ^A y_B \sin(^A\psi_B) \\ ^A x_B \sin(^A\psi_B) - ^A y_B \cos(^A\psi_B) \\ -^A z_B \\ -^A\psi_B \end{bmatrix} \tag{1.9}$$

Parameters
AxB – B-Frame pose expressed in A-Frame coordinates

Returns
A-Frame pose expressed in B-Frame coordinates (eq. (1.9)

J_ominus()

Jacobian of the inverse pose compounding operation ((1.6)) with respect the pose *AxB* (the self object):

$$J_{\ominus} = \frac{\partial \ominus^A x_B}{\partial^A x_B} = \begin{bmatrix} -\cos(^A\psi_B) & -\sin(^A\psi_B) & 0 & ^A x_B \sin(^A\psi_B) - ^A y_B \cos(^A\psi_B) \\ \sin(^A\psi_B) & -\cos(^A\psi_B) & 0 & ^A x_B \cos(^A\psi_B) + ^A y_B \sin(^A\psi_B) \\ 0 & 0 & -1 & ^A z_B \\ 0 & 0 & 0 & -1 \end{bmatrix} \tag{1.10}$$

Parameters
AxB – B-Fram pose expressed in A-Frame coordinates

Returns
$J_{\ominus}$ Jacobian of the inverse pose compounding operation with respect to the pose (eq. (1.10))

1.2 Robot Simulation

class prpy.**SimulatedRobot**(*xs0*, *map=[]*, **args*)

Bases: object

This is the base class to simulate a robot. There are two operative frames: the world N-Frame (North East Down oriented) and the robot body frame body B-Frame. Each robot has a motion model and a measurement model. The motion model is used to simulate the robot motion and the measurement model is used to simulate the robot measurements.

All Robot simulation classes must derive from this class .

dt = 0.1

class attribute containing sample time of the simulation

__init__(*xs0*, *map=[]*, **args*)

Parameters

- **xs0** – initial simulated robot state x_{s_0} used to initialize the the motion model
- **map** – feature map of the environment $M = [{}^N x_{F_1}^T, ..., {}^N x_{F_{nf}}^T]^T$

Constructor. First, it initializes the robot simulation defining the following attributes:

- **k** : time step
- **Qsk** : **To be defined in the derived classes**. Object attribute containing Covariance of the simulation motion model noise
- **usk** : **To be defined in the derived classes**. Object attribute contining the simulated input to the motion model
- **xsk** : **To be defined in the derived classes**. Object attribute contining the current simulated robot state
- **zsk** : **To be defined in the derived classes**. Object attribute contining the current simulated robot measurement
- **Rsk** : **To be defined in the derived classes**. Object attribute contining the observation noise covariance matrix
- **xsk** : current pose is the initial state
- **xsk_1** : previouse state is the initial robot state
- **M** : position of the features in the N-Frame
- **nf** : number of features

Then, the robot animation is initialized defining the following attributes:

- **vehicleIcon** : Path file of the image of the robot to be used in the animation
- **vehicleFig** : Figure of the robot to be used in the animation
- **vehicleAxes** : Axes of the robot to be used in the animation
- **xTraj** : list containing the x coordinates of the robot trajectory
- **yTraj** : list containing the y coordinates of the robot trajectory
- **visualizationInterval** : time-steps interval between two consecutive frames of the animation

PlotRobot()

Updates the plot of the robot at the current pose

fs(*xsk_1*, *uk*)

Motion model used to simulate the robot motion. Computes the current robot state x_k given the previous robot state x_{k-1} and the input u_k. It also updates the object attributes xsk, xsk_1 and usk to be made them available for plotting purposes. *To be overriden in child class*.

Parameters

- **xsk_1** – previous robot state x_{k-1}
- **uk** – model input u_k

Returns
current robot state x_k

SetMap(*map*)

Initializes the map of the environment.

_PlotSample(*x*, *P*, *n*)

Plots n samples of a multivariate gaussian distribution. This function is used only for testing, to plot the uncertainty through samples. :param x: mean pose of the distribution :param P: covariance of the distribution :param n: number of samples to plot

1.2.1 3 DOF Diferential Drive Robot Simulation

class prpy.**DifferentialDriveSimulatedRobot**(*xs0*, *map=[]*, **args*)

Bases: *SimulatedRobot*

This class implements a simulated differential drive robot. It inherits from the *SimulatedRobot* class and overrides some of its methods to define the differential drive robot motion model.

__init__(*xs0*, *map=[]*, **args*)

Parameters

- **xs0** – initial simulated robot state $\mathbf{x_{s_0}} = [{}^N x_{s_0}\ {}^N y_{s_0}\ {}^N \psi_{s_0}]^T$ used to initialize the motion model
- **map** – feature map of the environment $M = [{}^N x_{F_1}, ..., {}^N x_{F_{nf}}]$

Initializes the simulated differential drive robot. Overrides some of the object attributes of the parent class *SimulatedRobot* to define the differential drive robot motion model:

- **Qsk** : Object attribute containing Covariance of the simulation motion model noise.

$$Q_k = \begin{bmatrix} \sigma_u^2 & 0 & 0 \\ 0 & \sigma_v^2 & 0 \\ 0 & 0 & \sigma_r^2 \end{bmatrix} \tag{1.11}$$

- **usk** : Object attribute containing the simulated input to the motion model containing the forward velocity u_k and the angular velocity r_k

$$\mathbf{u_k} = \begin{bmatrix} u_k & r_k \end{bmatrix}^{\mathbf{T}} \tag{1.12}$$

- **xsk** : Object attribute containing the current simulated robot state

$$x_k = \begin{bmatrix} {}^N x_k & {}^N y_k & {}^N \theta_k & {}^B u_k & {}^B v_k & {}^B r_k \end{bmatrix}^T \tag{1.13}$$

where ${}^N x_k$, ${}^N y_k$ and ${}^N \theta_k$ are the robot position and orientation in the world N-Frame, and ${}^B u_k$, ${}^B v_k$ and ${}^B r_k$ are the robot linear and angular velocities in the robot B-Frame.

- **zsk** : Object attribute containing $z_{s_k} = [n_L\ n_R]^T$ observation vector containing number of pulses read from the left and right wheel encoders.
- **Rsk** : Object attribute containing $R_{s_k} = diag(\sigma_L^2, \sigma_R^2)$ covariance matrix of the noise of the read pulses`.
- **wheelBase** : Object attribute containing the distance between the wheels of the robot ($w = 0.5$ m)
- **wheelRadius** : Object attribute containing the radius of the wheels of the robot ($R = 0.1$ m)
- **pulses_x_wheelTurn** : Object attribute containing the number of pulses per wheel turn ($pulseXwheelTurn = 1024$ pulses)
- **Polar2D_max_range** : Object attribute containing the maximum Polar2D range ($Polar2D_max_range = 50$ m) at which the robot can detect features.
- **Polar2D_feature_reading_frequency** : Object attribute containing the frequency of Polar2D feature readings (50 tics -sample times-)
- **Rfp** : Object attribute containing the covariance of the simulated Polar2D feature noise ($R_{fp} = diag(\sigma_\rho^2, \sigma_\phi^2)$)

Check the parent class *prpy.SimulatedRobot* to know the rest of the object attributes.

fs(*xsk_1*, *usk*)

Motion model used to simulate the robot motion. Computes the current robot state x_k given the previous robot state x_{k-1} and the input u_k:

$$
\begin{aligned}
\eta_{s_{k-1}} &= \begin{bmatrix} x_{s_{k-1}} & y_{s_{k-1}} & \theta_{s_{k-1}} \end{bmatrix}^T \\
\nu_{s_{k-1}} &= \begin{bmatrix} u_{s_{k-1}} & v_{s_{k-1}} & r_{s_{k-1}} \end{bmatrix}^T \\
x_{s_{k-1}} &= \begin{bmatrix} \eta_{s_{k-1}}^T & \nu_{s_{k-1}}^T \end{bmatrix}^T \\
u_{s_k} &= \nu_d = \begin{bmatrix} u_d & r_d \end{bmatrix}^T \\
w_{s_k} &= \dot{\nu}_{s_k} \\
x_{s_k} &= f_s(x_{s_{k-1}}, u_{s_k}, w_{s_k}) \\
&= \begin{bmatrix} \eta_{s_{k-1}} \oplus (\nu_{s_{k-1}} \Delta t + \frac{1}{2} w_{s_k}) \\ \nu_{s_{k-1}} + K(\nu_d - \nu_{s_{k-1}}) + w_{s_k} \Delta t \end{bmatrix} \quad ; \quad K = diag(k_1, k_2, k_3) \quad k_i > 0
\end{aligned}
\tag{1.14}
$$

Where $\eta_{s_{k-1}}$ is the previous 3 DOF robot pose (x,y,yaw) and $\nu_{s_{k-1}}$ is the previous robot velocity (velocity in the direction of x and y B-Frame axis of the robot and the angular velocity). u_{s_k} is the input to the motion model contaning the desired robot velocity in the x direction (u_d) and the desired angular velocity around the z axis (r_d). w_{s_k} is the motion model noise representing an acceleration perturbation in the robot axis. The w_{s_k} acceleration is the responsible for the slight velocity variation in the simulated robot motion. K is a diagonal matrix containing the gains used to drive the simulated velocity towards the desired input velocity.

Finally, the class updates the object attributes xsk, xsk_1 and usk to made them available for plotting purposes.

To be completed by the student.

Parameters

- **xsk_1** – previous robot state $x_{s_{k-1}} = \begin{bmatrix} \eta_{s_{k-1}}^T & \nu_{s_{k-1}}^T \end{bmatrix}^T$
- **usk** – model input $u_{s_k} = \nu_d = \begin{bmatrix} u_d & r_d \end{bmatrix}^T$

Returns

current robot state x_{s_k}

ReadEncoders()

Simulates the robot measurements of the left and right wheel encoders.

To be completed by the student.

Return zsk,Rsk

$zk = [n_L\ n_R]^T$ observation vector containing number of pulses read from the left and right wheel encoders. $R_{s_k} = diag(\sigma_L^2, \sigma_R^2)$ covariance matrix of the read pulses.

ReadCompass()

Simulates the compass reading of the robot.

Returns

yaw and the covariance of its noise *R_yaw*

ReadCartesian2DFeature()

Simulates the reading of 2D cartesian features. The features are placed in the map in cartesian coordinates.

Returns

zsk: [[x1 y1],...,[xn yn]]
Cartesian position of the feature observations.

Rsk: block_diag(R_1,...,R_n), where R_i=[[r_xx r_xy],[r_xy r_yy]] is the
2x2 i-th feature observation covariance. Covariance of the Cartesian feature observations. Note the features are uncorrelated among them. They are independent. However, the x and y coordinates of each feature are correlated.

PlotRobot()

Updates the plot of the robot at the current pose

1.2.2 4 DOF AUV Simulation

class prpy.**AUV4DOFSimulatedRobot**(*xs0*, *map=[]*, **args*)

Bases: *SimulatedRobot*

This class simulates an AUV equipped with the following sensors: Gyro, Compass, DVL, Depth, direct USBL and inverted USBL.

dt = 0.1

class attribute containing sample time of the simulation

__init__(*xs0*, *map=[]*, **args*)

Constructor.

Parameters

- **xs0** – initial simulated robot pose $x_{s_k} = [x_{s_k}, y_{s_k}, z_{s_k}, \psi_{s_k}]$ in the N-Frame
- **map** – map of the environment

PlotRobot()

Updates the plot of the robot at the current pose

fs(*xsk_1*, *usk*)

Motion model used to simulate the robot motion. Computes the current robot state x_k given the previous

robot state x_{k-1} and the input u_k:

$$\begin{aligned}
\eta_{s_{k-1}} &= \begin{bmatrix} x_{s_{k-1}} & y_{s_{k-1}} & z_{s_{k-1}} & \psi_{s_{k-1}} \end{bmatrix}^T \\
\nu_{s_{k-1}} &= \begin{bmatrix} u_{s_{k-1}} & v_{s_{k-1}} & w_{s_{k-1}} & r_{s_{k-1}} \end{bmatrix}^T \\
x_{s_{k-1}} &= \begin{bmatrix} \eta_{s_{k-1}}^T & \nu_{s_{k-1}}^T \end{bmatrix}^T \\
u_{s_k} &= \nu_d = \begin{bmatrix} u_d & v_d & w_d & r_d \end{bmatrix}^T \\
w_{s_k} &= \dot{\nu}_{s_k} \\
x_{s_k} &= f_s(x_{s_{k-1}}, u_{s_k}, w_{s_k}) \\
&= \begin{bmatrix} \eta_{s_{k-1}} \oplus (\nu_{s_{k-1}} \Delta t + \frac{1}{2} w_{s_k}) \\ \nu_{s_{k-1}} + K(\nu_d - \nu_{s_{k-1}}) + w_{s_k} \Delta t \end{bmatrix} \quad ; \quad K = diag(k_1, k_2, k_3, k_4) \quad k_i > 0
\end{aligned} \tag{1.15}$$

Where $\eta_{s_{k-1}}$ is the previous 3 DOF robot pose (x,y,yaw) and $\nu_{s_{k-1}}$ is the previous robot velocity (velocity in the direction of x and y B-Frame axis of the robot and the angular velocity). u_{s_k} is the input to the motion model contaning the desired robot velocity in the x direction (u_d) and the desired angular velocity around the z axis (r_d). w_{s_k} is the motion model noise representing an acceleration perturbation in the robot axis. The w_{s_k} acceleration is the responsible for the slight velocity variation in the simulated robot motion. K is a diagonal matrix containing the gains used to drive the simulated velocity towards the desired input velocity.

Finally, the class updates the object attributes xsk, xsk_1 and usk to made them available for plotting purposes.

To be completed by the student.

Parameters

- **xsk_1** – previous robot state $x_{s_{k-1}} = \begin{bmatrix} \eta_{s_{k-1}}^T & \nu_{s_{k-1}}^T \end{bmatrix}^T$
- **usk** – model input $u_{s_k} = \nu_d = \begin{bmatrix} u_d & r_d \end{bmatrix}^T$

Returns
current robot state x_{s_k}

`SetMap`(*map*)

Initializes the map of the environment.

`ReadGyro`()

Simulates the gyro reading of the robot.

Returns
angular velocity [r] and the covariance of the noise [R_r]

`ReadCompass`()

Simulates the compass reading of the robot.

Returns
yaw and the covariance of the noise [R_yaw]

`ReadDVL`()

Simulates the DVL reading of the robot.

Returns
linear velocity [u v w] and the covariance of the noise [R_lin_vel]

`ReadDepth`()

Simulates the depth reading of the robot.

Returns
depth and the covariance of the noise [R_depth]

`ReadUSBL()`

Simulates the USBL reading of the robot. Assumes that the USBL transceiver is placed at the origin of the N_Frame and the transceiver is placed at the origin of the robot's B-Frame.

Returns
zsk: cartesian position [x y z] Rsk: Covariance of the noise [R_usbl_cartesian]

`ReadSpherical3DFeature()`

Simulates the inverted USBL sensor. In this case the transceiver is mounted at the origin of the robot's B-Frame and n transponders are layed out on the seafloor. The Cartesian position of the transponders is stored in the map in Cartesian coordinates.

Returns
zsk=[[r_0 theta_0 varphi_0],...,[r_i theta_i varphi_i],...[r_n theta_n varphi_n]], Rsk=block_diag(R_0,...,R_i,...,R_n)

where [r_i theta_i phi_i] is the spherical position of the i transponder, being, r_i the distance, theta_i the elevation angle and varphi_i the azimuth angle. The covariance is a block diagonal matrix (since the transponder positions are uncorrelated) where each block is a 3x3 matrix corresponding to the covariance of the noise of each transponder within the range of the sensor.

`ReadCartesian2DFeature()`

Simulates the reading of 2D cartesian features. The features are placed in the map in cartesian coordinates.

Returns

zsk: [[x1 y1],...,[xn yn]]
Cartesian position of the feature observations.

Rsk: block_diag(R_1,...,R_n), where R_i=[[r_xx r_xy],[r_xy r_yy]] is the
2x2 i-th feature observation covariance. Covariance of the Cartesian feature observations. Note the features are uncorrelated among them.

`ReadPolar2DFeature()`

Simulates the reading of 2D Polar features. The features are placed in the map in cartesian coordinates.

Returns

zsk: [[x1 y1 z1],...,[xn yn zn]]
Cartesian position of the feature observations.

Rsk: block_diag(R_1,...,R_n), where R_i=[[r_xx r_xy r_xz],[r_xy r_yy r_yz],[r_xz r_yz r_yy]] is the
2x2 i-th feature observation covariance. Covariance of the Polar feature observations. Note the features are uncorrelated among them.

`ReadCartesian3DFeature()`

Simulates the reading of 3D cartesian features. The features are placed in the map in cartesian coordinates.

Returns

zsk: [[x1 y1 z1],...,[xn yn zn]]
Cartesian position of the feature observations.

Rsk: block_diag(R_1,...,R_n), where R_i=[[r_xx r_xy r_xz],[r_xy r_yy r_yz],[r_xz r_yz r_yy]] is the
3x3 i-th feature observation covariance. Covariance of the Cartesian feature observations. Note the features are uncorrelated among them.

_PlotSample(*x*, *P*, *n*)

Plots n samples of a multivariate gaussian distribution. This function is used only for testing, to plot the uncertainty through samples. :param x: mean pose of the distribution :param P: covariance of the distribution :param n: number of samples to plot

1.3 Filters

class prpy.**GaussianFilter**(*x0*, *P0*, **args*)

Bases: `object`

Gaussian Filter Interface

__init__(*x0*, *P0*, **args*)

Constructor of the GaussianFilter class.

Parameters

- **x0** – initial mean state vector
- **P0** – initial covariance matrix

Prediction(*uk*, *Qk*, *xk_1=None*, *Pk_1=None*)

Prediction step of the Gaussian Filter to be overwritten by the child class.

Parameters

- **uk** – input vector
- **Qk** – covariance matrix of the motion model noise
- **xk_1** – previous mean state vector
- **Pk_1** – previous covariance matrix

Return xk_bar, Pk_bar
current mean state vector and covariance matrix

Update(*zk*, *Rk*, *xk_bar=None*, *Pk_bar=None*)

Update step of the Gaussian Filter to be overwritten by the child class.

Parameters

- **zk** – observation vector
- **Rk** – covariance of the observation model noise
- **xk_bar** – mean of the predicted state
- **Pk_bar** – covariance of the predicted state

Returns
xk, Pk: current mean state vector and covariance matrix

class prpy.**KF**(*Ak*, *Bk*, *Hk*, *Vk*, *x0*, *P0*, **args*)

Bases: *GaussianFilter*

Kalman Filter class. Implements the *prpy.GaussianFilter* interface for the particular case of the Kalman Filter.

__init__(*Ak*, *Bk*, *Hk*, *Vk*, *x0*, *P0*, **args*)

Constructor of the KF class.

Parameters

- **Ak** – Transition matrix of the motion model
- **Bk** – Input matrix of the motion model
- **Hk** – Observation matrix of the observation model
- **Vk** – Noise projection matrix of the motion model
- **x0** – initial mean of the state vector
- **P0** – initial covariance matrix
- **args** – arguments to be passed to the parent class

Prediction(*uk*, *Qk*, *xk_1=None*, *Pk_1=None*)

Prediction step of the Kalman Filter.

Parameters

- **uk** – input vector
- **Qk** – covariance matrix of the motion model noise
- **xk_1** – previous mean state vector
- **Pk_1** – previous covariance matrix

Return xk_bar, Pk_bar
current mean state vector and covariance matrix

Update(*zk*, *Rk*, *xk_bar=None*, *Pk_bar=None*)

Update step of the Kalman Filter.

Parameters

- **zk** – observation vector
- **Rk** – covariance of the observation model noise
- **xk_bar** – predicted mean state vector
- **Pk_bar** – predicted covariance matrix

Return xk.Pk
current mean state vector and covariance matrix

class prpy.**EKF**(*x0*, *P0*, **args*)

Bases: *GaussianFilter*

Extended Kalman Filter class. Implements the *prpy.GaussianFilter* interface for the particular case of the Extended Kalman Filter.

__init__(*x0*, *P0*, **args*)

Constructor of the EKF class.

Parameters

- **x0** – initial mean state vector
- **P0** – initial covariance matrix
- **args** – arguments to be passed to the parent class

f(*xk_1=None*, *uk=None*)

" Motion model of the EKF to be overwritten by the child class.

Parameters

- **xk_1** – previous mean state vector
- **uk** – input vector

Return xk_bar, Pk_bar
predicted mean state vector and its covariance matrix

Jfx(*xk_1=None*)

Jacobian of the motion model with respect to the state vector. *Method to be overwritten by the child class*.

Parameters
xk_1 – Linearization point. By default the linearization point is the previous state vector taken from a class attribute.

Returns
Jacobian matrix

Jfw(*xk_1=None*)

Jacobian of the motion model with respect to the noise vector. *Method to be overwritten by the child class*.

Parameters
xk_1 – Linearization point. By default the linearization point is the previous state vector taken from a class attribute.

Returns
Jacobian matrix

h(*xk_bar=None*)

Observation model of the EKF. We differentiate two types of observations: 1. **Measurements**: observations that are directly measured by the sensors. For example, the position of the robot, its heading, its speed, etc. 2. **Features**: observations of map features. For example, the position of a landmark.

This method calls the *prpy.EKF.hm()* which implements the measurements observation equation. To implement a standard EKF, the *prpy.EKF.hm* method should be overwritten by the child class to be used as the observation equation for measurements.

Parameters
xk_bar – mean of the predicted state vector. By default it is taken from the class attribute.

Returns
expected observation vector

Jhx(*xk_bar=None*)

Jacobian of the observation model with respect to the state vector. Calls the *prpy.EKF.Jhmx()* method which implements the measurements observation Jacobian. To implement a standard EKF, the *prpy.EKF.Jhmx* method should be overwritten by the child class.

Parameters
xk_bar – linearization point. By default it is taken from the class attribute.

Returns
Jacobian matrix

Jhv(*xk_bar=None*)

Jacobian of the observation model with respect to the noise vector. Calls the *prpy.EKF.Jhmv()* method which should be overwritten by the child class. To implement a standard EKF, the *prpy.EKF.Jhmv* method should be overwritten by the child class.

Parameters
xk_bar – linearization point. By default it is taken from the class attribute.

Returns
Jacobian matrix

hm(*xk_bar=None*)

Observation model related to the measurements observations of the EKF to be overwritten by the child class.

Parameters
xk_bar – mean of the predicted state vector. By default it is taken from the class attribute.

Returns
expected observation vector

Jhmx(*xk_bar=None*)

Jacobian of the measurement observation model with respect to the state vector. *Method to be overwritten by the child class*.

Parameters
xk_bar – linearization point. By default it is taken from the class attribute.

Returns
Jacobian matrix

Jhmv(*xk_bar=None*)

Jacobian of the measurement observation model with respect to the noise vector. *Method to be overwritten by the child class*.

Parameters
xk_bar – linearization point. By default it is taken from the class attribute.

Returns
Jacobian matrix

Prediction(*uk*, *Qk*, *xk_1=None*, *Pk_1=None*)

Prediction step of the EKF. It calls the motion model and its Jacobians to predict the state vector and its covariance matrix.

Parameters

- **uk** – input vector
- **Qk** – covariance matrix of the noise vector
- **xk_1** – previous mean state vector. By default it is taken from the class attribute. Otherwise it updates the class attribute.
- **Pk_1** – covariance matrix of the previous state vector. By default it is taken from the class attribute. Otherwise it updates the class attribute.

Return xk_bar, Pk_bar
predicted mean state vector and its covariance matrix. Also updated in the class attributes.

Update(*zk*, *Rk*, *xk_bar=None*, *Pk_bar=None*)

Update step of the EKF. It calls the observation model and its Jacobians to update the state vector and its covariance matrix.

Parameters

- **zk** – observation vector

- **Rk** – covariance matrix of the noise vector
- **xk_bar** – predicted mean state vector. By default it is taken from the class attribute. Otherwise it updates the class attribute.
- **Pk_bar** – covariance matrix of the predicted state vector. By default it is taken from the class attribute. Otherwise it updates the class attribute.

Return xk,Pk
updated mean state vector and its covariance matrix. Also updated in the class attributes.

1.4 Robot Localization

1.4.1 Robot Localization

class prpy.**Localization**(*index*, *kSteps*, *robot*, *x0*, **args*)

Bases: object

Localization base class. Implements the localization algorithm.

__init__(*index*, *kSteps*, *robot*, *x0*, **args*)

Constructor of the DRLocalization class.

Parameters

- **index** – Logging index structure (prpy.Index)
- **kSteps** – Number of time steps to simulate
- **robot** – Simulation robot object (prpy.Robot)
- **args** – Rest of arguments to be passed to the parent constructor
- **x0** – Initial Robot pose in the N-Frame

GetInput()

Gets the input from the robot. To be overidden by the child class.

Return uk
input variable

Localize(*xk_1*, *uk*)

Single Localization iteration invoked from prpy.DRLocalization.Localization(). Given the previous robot pose, the function reads the inout and computes the current pose.

Parameters
xk_1 – previous robot pose

Return xk
current robot pose

LocalizationLoop(*x0*, *usk*)

Given an initial robot pose x_0 and the input to the *prpy.SimulatedRobot* this method calls iteratively prpy.DRLocalization.Localize() for k steps, solving the robot localization problem.

Parameters
x0 – initial robot pose

Log(*xsk*, *xk*)

Logs the results for later plotting.

Parameters

- **xsk** – ground truth robot pose from the simulation
- **xk** – estimated robot pose

PlotXY()

Plots, in a new figure, the ground truth (orange) and estimated (blue) trajectory of the robot at the end of the Localization Loop.

PlotTrajectory()

Plots the estimated trajectory (blue) of the robot during the localization process.

1.4.2 Dead Reckoning

Dead Reckoning

3 DOF Differential Drive Mobile Robot Example

class prpy.**DR_3DOFDifferentialDrive**(*index*, *kSteps*, *robot*, *x0*, **args*)

Bases: *Localization*

Dead Reckoning Localization for a Differential Drive Mobile Robot.

__init__(*index*, *kSteps*, *robot*, *x0*, **args*)

Constructor of the prlab.DR_3DOFDifferentialDrive class.

Parameters

args – Rest of arguments to be passed to the parent constructor

Localize(*xk_1*, *uk*)

Motion model for the 3DOF ($x_k = [x_k\ y_k\ \psi_k]^T$) Differential Drive Mobile robot using as input the readings of the wheel encoders ($u_k = [n_L\ n_R]^T$).

Parameters

- **xk_1** – previous robot pose estimate ($x_{k-1} = [x_{k-1}\ y_{k-1}\ \psi_{k-1}]^T$)
- **uk** – input vector ($u_k = [u_k\ v_k\ w_k\ r_k]^T$)

Return xk

current robot pose estimate ($x_k = [x_k\ y_k\ \psi_k]^T$)

GetInput()

Get the input for the motion model. In this case, the input is the readings from both wheel encoders.

Returns

uk: input vector ($u_k = [n_l\ n_R]^T$)

4 DOF AUV Robot Example

class prpy.**DR_4DOFAUV_DVLGyro**(*index*, *kSteps*, *robot*, *x0*, **args*)

Bases: *Localization*

Dead Reckoning Localization for a 4DOF AUV with DVL and Gyro sensors

__init__(*index*, *kSteps*, *robot*, *x0*, **args*)

Constructor of the DR_4DOFAUV_DVLGyro class.

Parameters
args – Rest of arguments to be passed to the parent constructor

Localize(*xk_1*, *uk*)

Motion model for the 4DOF ($[x_k\ y_k\ z_k\ \psi_k]^T$) AUV using as input the lineal velocity read from the DVL senosr and angular velocity read from the Gyro sensor

Parameters

- **xk_1** – previous robot pose estimate ($[x_{k-1}\ y_{k-1}\ z_{k-1}\ \psi_{k-1}]^T$)
- **uk** – input vector ($[u_k\ v_k\ w_k\ r_k]^T$)

Return xk
current robot pose estimate ($[x_k\ y_k\ z_k\ \psi_k]^T$)

GetInput()

Gets the input vector and the input noise covariance matrix from the robot. The input vector contains the linear read from the DVL and angular velocity read from the Gyro of the robot.

$$u_k = \begin{bmatrix} \nu_{DVL}^T & r_{Gyro}^T \end{bmatrix}^T = \begin{bmatrix} u_{DVL} & v_{DVL} & w_{DVL} & r_{Gyro} \end{bmatrix}^T$$
$$Q_k = \begin{bmatrix} Q_{DVL} & 0 \\ 0 & \sigma_{Gyro}^2 \end{bmatrix} \quad \text{where} \quad Q_{DVL} = \begin{bmatrix} \sigma_{u_{DVL}}^2 & \sigma_{uv_{DVL}} & \sigma_{uw_{DVL}} \\ \sigma_{vu_{DVL}} & \sigma_{v_{DVL}}^2 & \sigma_{vw_{DVL}} \\ \sigma_{wu_{DVL}} & \sigma_{wv_{DVL}} & \sigma_{w_{DVL}}^2 \end{bmatrix} \tag{1.16}$$

Returns
input vector u_k and input noise covariance matrix Q_k defined in eq. (1.16).

1.4.3 Localization using Gaussian Filters

Gaussian Filter Localization

class prpy.**GFLocalization**(*index*, *kSteps*, *robot*, *x0*, *P0*, **args*)

Bases: *Localization*, *GaussianFilter*

Map-less localization using KF and EKF filters.

__init__(*index*, *kSteps*, *robot*, *x0*, *P0*, **args*)

Constructor.

Parameters

- **x0** – initial state
- **P0** – initial covariance
- **index** – Named tuple used to map the state vector, the simulation vector and the observation vector (`prpy.IndexStruct`)

- **kSteps** – simulation time steps
- **robot** – Simulated Robot object
- **args** – arguments to be passed to the parent constructor

`GetInput()`

Get the input from the robot. Relates to the motion model as follows:

$$\begin{aligned} x_k &= f(x_{k-1}, u_k, w_k) \\ w_k &= N(0, Q_k) \end{aligned} \tag{1.17}$$

To be overidden by the child class .

Return uk, Qk
input and covariance of the motion model

`GetMeasurements()`

Get the measurements from the robot. Corresponds to the observation model:

$$\begin{aligned} z_k &= h(x_k, v_k) \\ v_k &= N(0, R_k) \end{aligned} \tag{1.18}$$

To be overidden by the child class .

Return zk, Rk
observation vector and covariance of the observation noise.

`Localize(xk_1, Pk_1)`

Localization iteration. Reads the input of the motion model, performs the prediction step, reads the measurements, performs the update step and logs the results. The method also plots the uncertainty ellipse of the robot pose.

Parameters

- **xk_1** – previous state vector
- **Pk_1** – previous covariance matrix

Return xk, Pk
updated state vector and covariance matrix

`LocalizationLoop(x0, P0, usk)`

Localization loop. Durinf *self.kSteps* it calls the *Localize()* method for each time step.

Parameters

- **x0** – initial state vector
- **P0** – initial covariance matrix

`Log(xsk, xk, Pk, xk_bar, zk)`

Logs the results for later plotting.

Parameters

- **xsk** – ground truth robot pose from the simulation
- **xk** – estimated robot pose

`PlotState()`

Plot the results of the localization For each state DOF s -si[s] is the corresponding simulated stated -x1[s] is the corresponding observation

PlotXY(*estimation=True*)

Plot the x-y trajectory of the robot simulation: True if the simulated XY robot trajectory is available

PlotRobotUncertainty(*xk*, *Pk*)

PlotUncertainty(*xk*, *Pk*)

KF Localization

KF Localization

EKF Localization

EKF Localization

3 DOF Diferential Drive Mobile Robot

Input Displacement Motion Model

3 DOF EKF Localization with Input Displacement Motion Model and Yaw Observation Model

class prpy.**EKF_3DOFxyYaw_DisplacementMM_YawOM**(*kSteps*, *robot*, **args*)

Bases: *GFLocalization*, *DR_3DOFDifferentialDrive*, *EKF*

This class implements an EKF localization filter for a 4 DOF AUV using an input velocity motion model incorporating DVL linear velocity mesurements, a gyro angular speed measurement, as well as depth and yaw measurements. Inherit from `prpy.PoseCompounding4DOF` first, to ensure it uses the overridden $\oplus$ and $\ominus$ methods. Then, it inherits from *prpy.GFLocalization* to implement a localization filter and, finally, it inherits from *prpy.EKF* to use the EKF Gaussian filter implementation for the localization.

__init__(*kSteps*, *robot*, **args*)

Constructor.

Parameters
args – arguments to be passed to the base class constructor

GetInput()

Get the input from the robot. Relates to the motion model as follows:

$$\begin{aligned} x_k &= f(x_{k-1}, u_k, w_k) \\ w_k &= N(0, Q_k) \end{aligned} \tag{1.19}$$

To be overidden by the child class .

Return uk, Qk
input and covariance of the motion model

f(*xk_1*, *uk*)

Non-linear motion model using as input the DVL linear velocity and the gyro angular speed:

$$\begin{aligned} x_k &= f(x_{k-1}, u_k, w_k) = x_{k-1} \oplus (u_k + w_k)\Delta t \\ x_{k-1} &= [x_{k_1}^T, y_{k_1}^T, z_{k_1}^T, \psi_{k_1}^T]^T \\ u_k &= [u_k, v_k, w_k, r_k]^T \end{aligned}$$

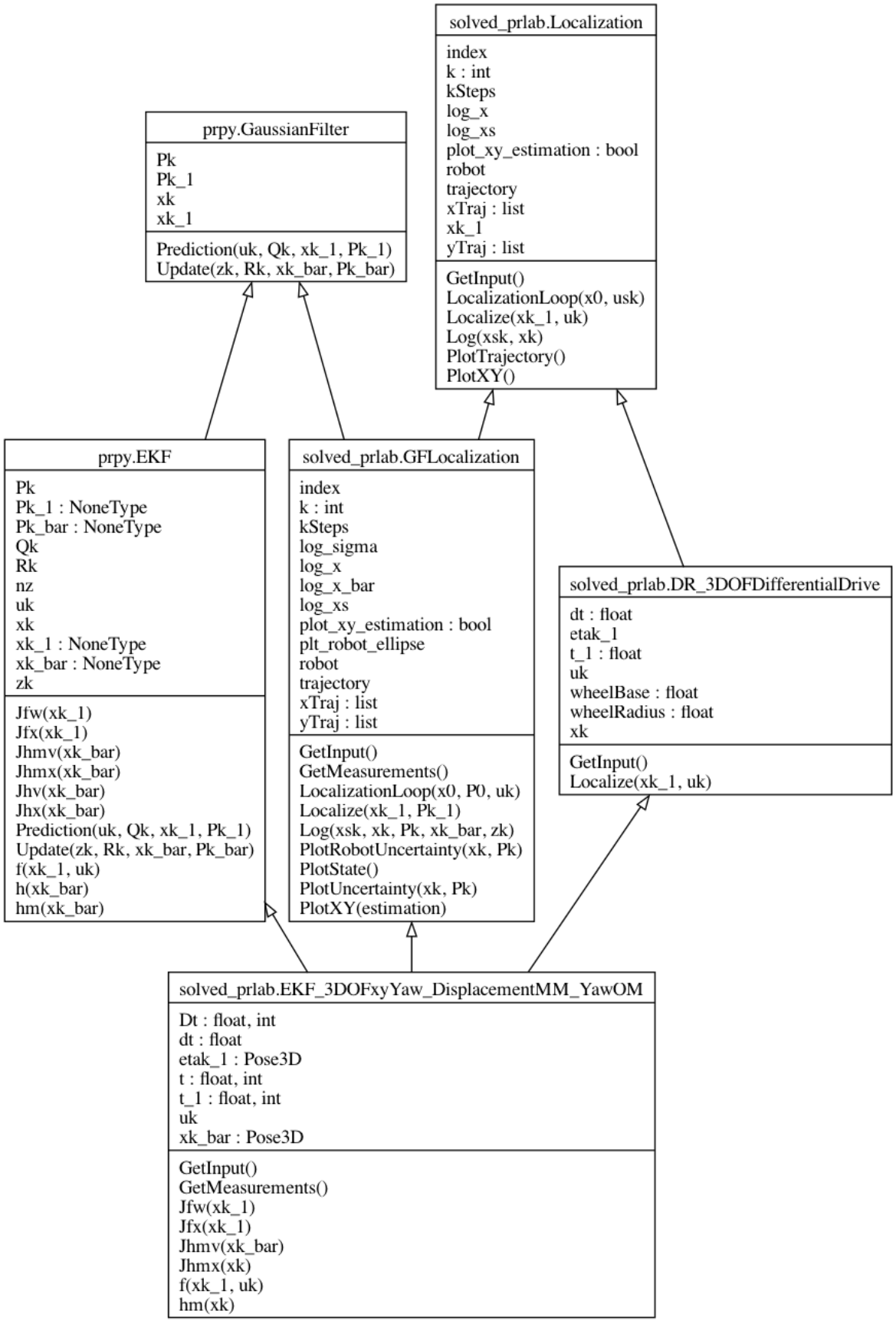


Fig. 1: EKF_3DOFxyYaw_DisplacementMM_YawOM Class Diagram.

Parameters

- **xk_1** – previous mean state vector ($x_{k-1} = [x_{k-1}^T, y_{k-1}^T, z_{k-1}^T, \psi_{k-1}^T]^T$) containing the robot position and heading in the N-Frame
- **uk** – input vector $u_k = [u_k^T, v_k^T, w_k^T, r_k^T]^T$ containing the DVL linear velocity and the gyro angular speed, both referenced in the B-Frame

Returns

current mean state vector containing the current robot position and heading ($x_k = [x_k^T, y_k^T, z_k^T, \psi_k^T]^T$) represented in the N-Frame

Jfx(*xk_1*)

Jacobian of the motion model with respect to the state vector:

$$J_{fx} = \frac{\partial f(x_{k-1}, u_k, w_k)}{\partial x_{k-1}} = \frac{\partial x_{k-1} \oplus (u_k + w_k)}{\partial x_{k-1}} = J_{1\oplus} \tag{1.20}$$

Parameters

xk_1 – Linearization point. By default the linearization point is the previous state vector taken from a class attribute.

Returns

Jacobian matrix

Jfw(*xk_1*)

Jacobian of the motion model with respect to the motion model noise vector:

$$J_{fx} = \frac{\partial f(x_{k-1}, u_k, w_k)}{\partial w_k} = \frac{\partial x_{k-1} \oplus (u_k + w_k)}{\partial w_k} = J_{2\oplus} \tag{1.21}$$

Parameters

xk_1 – Linearization point. By default the linearization point is the previous state vector taken from a class attribute.

Returns

Jacobian matrix

hm(*xk*)

Observation model related to the measurements observations of the EKF to be overwritten by the child class.

Parameters

xk_bar – mean of the predicted state vector. By default it is taken from the class attribute.

Returns

expected observation vector

Jhmx(*xk*)

Jacobian of the measurement model with respect to the state vector:

$$J_{hmx} = H_{m_k} = \frac{\partial h_m(x_k, v_k)}{\partial x_k} = \frac{\partial [z_{depth}^T, \psi_{compass}^T]^T}{\partial x_k} = \begin{bmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{1.22}$$

Parameters

xk – mean state vector containing the robot position and heading ($x_k = [x_k^T, y_k^T, z_k^T, \psi_k^T]^T$) represented in the N-Frame

Returns

observation matrix (Jacobian) matrix eq. (1.22).

Jhmv(*xk_bar*)

Jacobian of the measurement model with respect to the measurement noise vector:

$$J_{hmv} = V_{m_k} = \frac{\partial h_m(x_k, v_k)}{\partial v_k} = I_{2\times 2} \tag{1.23}$$

Parameters

xk – mean state vector containing the robot position and heading ($x_k = [x_k^T, y_k^T, z_k^T, \psi_k^T]^T$) represented in the N-Frame

Returns

observation noise (Jacobian) matrix eq. (1.23).

GetMeasurements()

Gets the measurement vector and the measurement noise covariance matrix from the robot. The measurement vector contains the depth read from the depth sensor and the heading read from the compass sensor.

Returns

observation vector z_k and observation noise covariance matrix R_k defined in eq. eq-zk-EKF_3DOFxyYaw_DisplacementMM_YawOM.

Constant Velocity Motion Model

3 DOF EKF Localization with Constant Velocity Motion Model and Yaw and velocity Observation Model

class prpy.**EKF_3DOFxyYaw_CtVelocityMM_YawVelocityOM**(*kSteps*, *robot*, **args*)

Bases: *GFLocalization*, *DR_3DOFDifferentialDrive*, *EKF*

__init__(*kSteps*, *robot*, **args*)

Constructor.

Parameters

- **x0** – initial state
- **P0** – initial covariance
- **index** – Named tuple used to map the state vector, the simulation vector and the observation vector (prpy.IndexStruct)
- **kSteps** – simulation time steps
- **robot** – Simulated Robot object
- **args** – arguments to be passed to the parent constructor

f(*xk_1*, *uk*)

” Motion model of the EKF to be overwritten by the child class.

Parameters

- **xk_1** – previous mean state vector
- **uk** – input vector

Return xk_bar, Pk_bar

predicted mean state vector and its covariance matrix

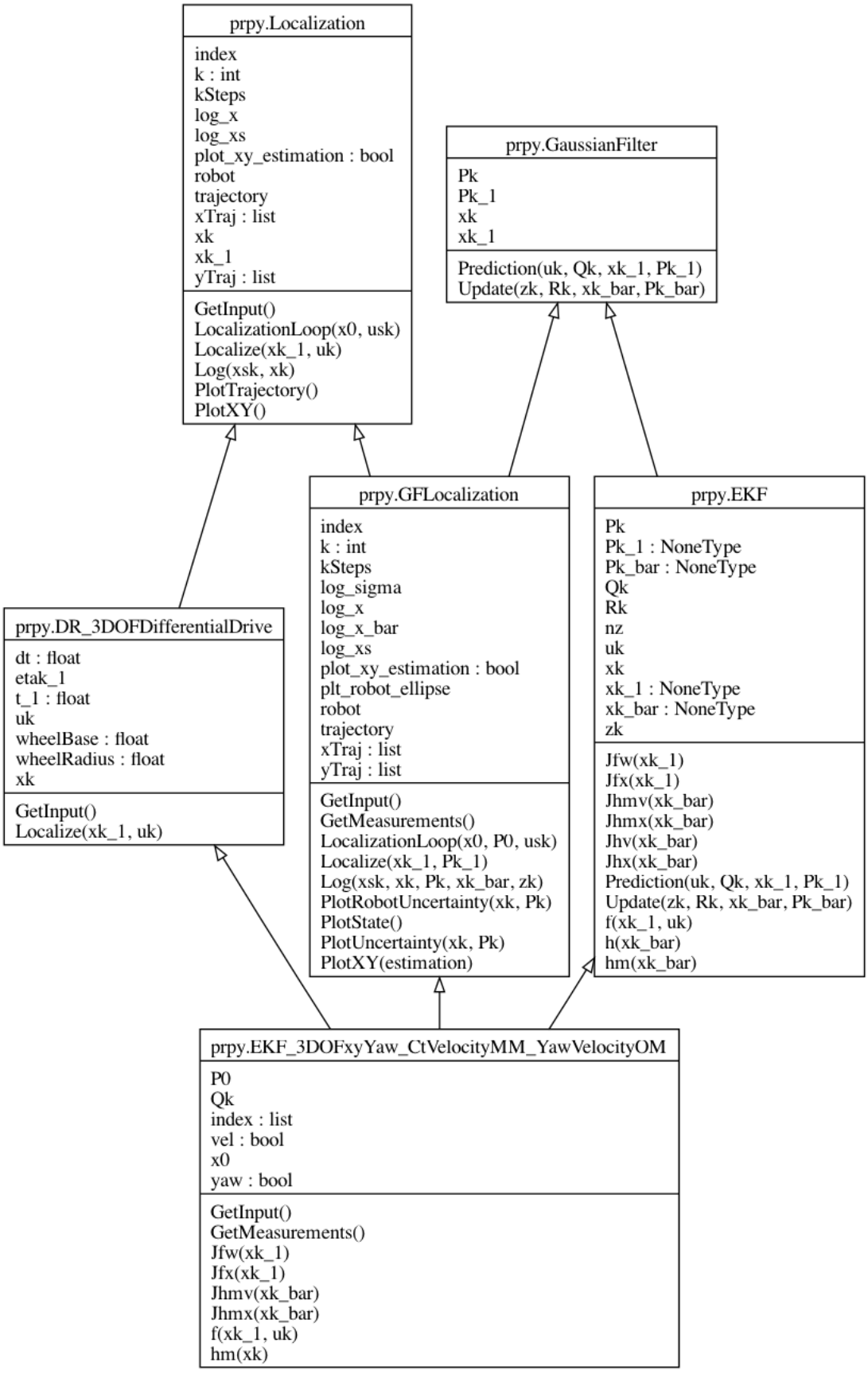


Fig. 2: EKF_3DOFxyYaw_CtVelocityMM_YawVelocityOM Class Diagram.

Jfx(*xk_1*)

Jacobian of the motion model with respect to the state vector. *Method to be overwritten by the child class*.

Parameters
xk_1 – Linearization point. By default the linearization point is the previous state vector taken from a class attribute.

Returns
Jacobian matrix

Jfw(*xk_1*)

Jacobian of the motion model with respect to the noise vector. *Method to be overwritten by the child class*.

Parameters
xk_1 – Linearization point. By default the linearization point is the previous state vector taken from a class attribute.

Returns
Jacobian matrix

hm(*xk*)

Observation model related to the measurements observations of the EKF to be overwritten by the child class.

Parameters
xk_bar – mean of the predicted state vector. By default it is taken from the class attribute.

Returns
expected observation vector

Jhmx(*xk_bar*)

Jacobian of the measurement observation model with respect to the state vector. *Method to be overwritten by the child class*.

Parameters
xk_bar – linearization point. By default it is taken from the class attribute.

Returns
Jacobian matrix

Jhmv(*xk_bar*)

Jacobian of the measurement observation model with respect to the noise vector. *Method to be overwritten by the child class*.

Parameters
xk_bar – linearization point. By default it is taken from the class attribute.

Returns
Jacobian matrix

GetInput()

Get the input from the robot. Relates to the motion model as follows:

$$\begin{aligned} x_k &= f(x_{k-1}, u_k, w_k) \\ w_k &= N(0, Q_k) \end{aligned} \tag{1.24}$$

To be overidden by the child class .

Return uk, Qk
input and covariance of the motion model

GetMeasurements()

Get the measurements from the robot. Corresponds to the observation model:

$$\begin{aligned} z_k &= h(x_k, v_k) \\ v_k &= N(0, R_k) \end{aligned} \tag{1.25}$$

To be overidden by the child class .

Return zk, Rk

observation vector and covariance of the observation noise.

4 DOF AUV Robot

Input Displacement Motion Model

4 DOF EKF Localization with Input Velocity Motion Model and Yaw Observation Model

class prpy.**EKF_4DOFAUV_VelocityMM_DVLDepthYawOM**(*kSteps*, *robot*, **args*)

Bases: *GFLocalization*, *DR_4DOFAUV_DVLGyro*, *EKF*

This class implements an EKF localization filter for a 4 DOF AUV using an input velocity motion model incorporating DVL linear velocity measurements, a gyro angular speed measurement, as well as depth and yaw measurements. Inherits from GFLocalization because it is a Localization method using Gaussian filtering, and from EKF because it uses an EKF. It also inherits from DR_4DOFAUV_DVLGyro to reuse its motion model `solved_prlab.DR_4DOFAUV_DVLGyro.Localize()` and the model input `solved_prlab.DR_4DOFAUV_DVLGyro.GetInput()`.

__init__(*kSteps*, *robot*, **args*)

Constructor.

Parameters

args – arguments to be passed to the base class constructor

f(*xk_1*, *uk*)

Non-linear motion model using as input the DVL linear velocity and the gyro angular speed:

$$\begin{aligned} x_k &= f(x_{k-1}, u_k, w_k) = x_{k-1} \oplus (u_k + w_k)\Delta t \\ x_{k-1} &= [x_{k_1}^T, y_{k_1}^T, z_{k_1}^T, \psi_{k_1}^T]^T \\ u_k &= [u_k, v_k, w_k, r_k]^T \end{aligned}$$

Parameters

- **xk_1** – previous mean state vector ($x_{k-1} = [x_{k-1}^T, y_{k-1}^T, z_{k-1}^T, \psi_{k-1}^T]^T$) containing the robot position and heading in the N-Frame
- **uk** – input vector $u_k = [u_k^T, v_k^T, w_k^T, r_k^T]^T$ containing the DVL linear velocity and the gyro angular speed, both referenced in the B-Frame

Returns

current mean state vector containing the current robot position and heading ($x_k = [x_k^T, y_k^T, z_k^T, \psi_k^T]^T$) represented in the N-Frame

Jfx(*xk_1*)

Jacobian of the motion model with respect to the state vector:

$$J_{fx} = \frac{\partial f(x_{k-1}, u_k, w_k)}{\partial x_{k-1}} = \frac{\partial x_{k-1} \oplus (u_k + w_k)}{\partial x_{k-1}} = J_{1\oplus} \tag{1.26}$$

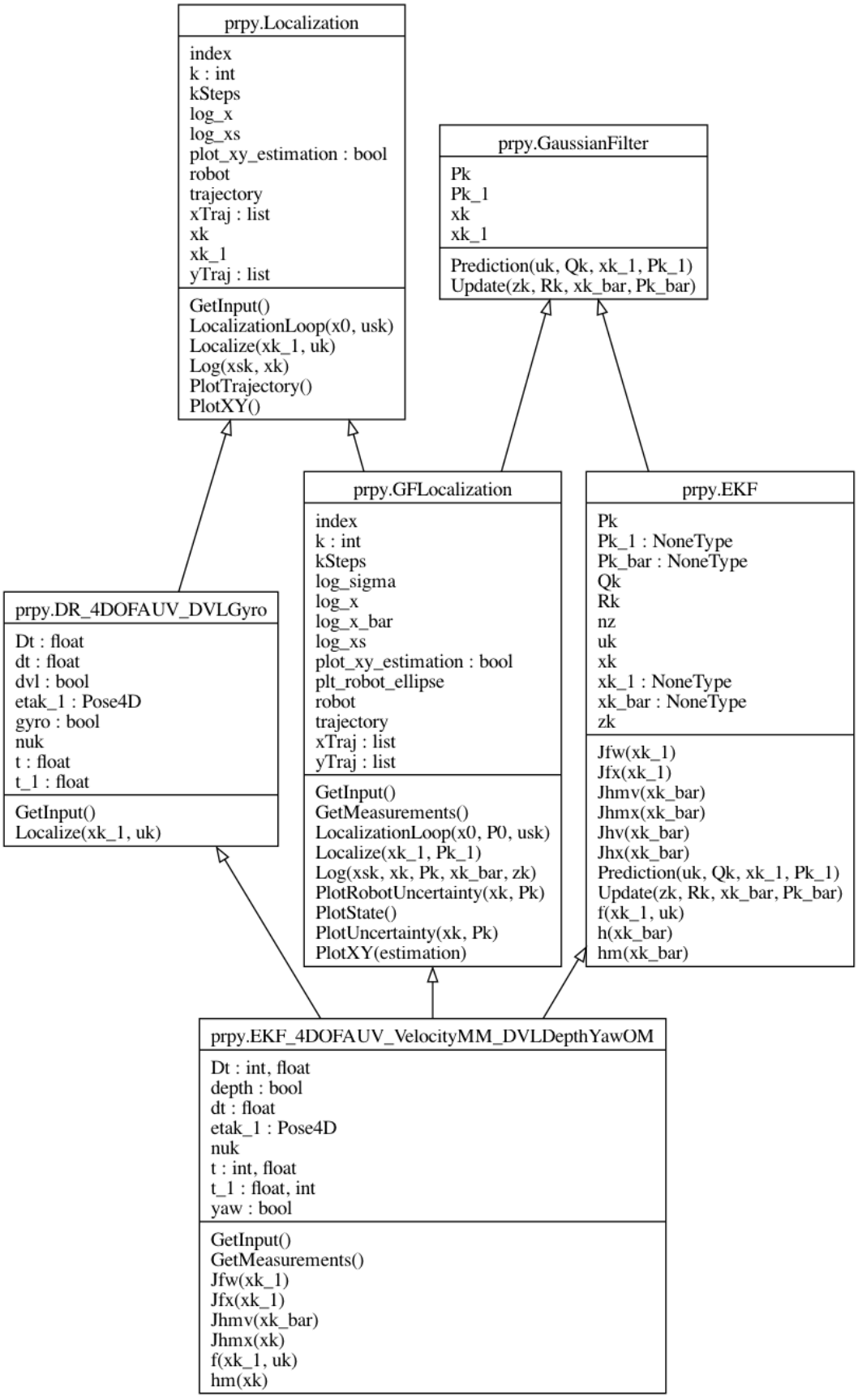


Fig. 3: EKF_4DOFAUV_VelocityMM_DVLDepthYawOM Class Diagram.

Parameters
xk_1 – Linearization point. By default the linearization point is the previous state vector taken from a class attribute.

Returns
Jacobian matrix

Jfw(*xk_1*)

Jacobian of the motion model with respect to the motion model noise vector:

$$J_{fx} = \frac{\partial f(x_{k-1}, u_k, w_k)}{\partial w_k} = \frac{\partial x_{k-1} \oplus (u_k + w_k)}{\partial w_k} = J_{2\oplus} \tag{1.27}$$

Parameters
xk_1 – Linearization point. By default the linearization point is the previous state vector taken from a class attribute.

Returns
Jacobian matrix

hm(*xk*)

Observation model related to the measurements observations of the EKF to be overwritten by the child class.

Parameters
xk_bar – mean of the predicted state vector. By default it is taken from the class attribute.

Returns
expected observation vector

Jhmx(*xk*)

Jacobian of the measurement model with respect to the state vector:

$$J_{hmx} = H_{m_k} = \frac{\partial h_m(x_k, v_k)}{\partial x_k} = \frac{\partial [z_{depth}^T, \psi_{compass}^T]^T}{\partial x_k} = \begin{bmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{1.28}$$

Parameters
xk – mean state vector containing the robot position and heading ($x_k = [x_k^T, y_k^T, z_k^T, \psi_k^T]^T$) represented in the N-Frame

Returns
observation matrix (Jacobian) matrix eq. (1.28).

Jhmv(*xk_bar*)

Jacobian of the measurement model with respect to the measurement noise vector:

$$J_{hmv} = V_{m_k} = \frac{\partial h_m(x_k, v_k)}{\partial v_k} = I_{2\times 2} \tag{1.29}$$

Parameters
xk – mean state vector containing the robot position and heading ($x_k = [x_k^T, y_k^T, z_k^T, \psi_k^T]^T$) represented in the N-Frame

Returns
observation noise (Jacobian) matrix eq. (1.29).

GetInput()

Get the input from the robot. Relates to the motion model as follows:

$$\begin{aligned} x_k &= f(x_{k-1}, u_k, w_k) \\ w_k &= N(0, Q_k) \end{aligned} \tag{1.30}$$

To be overidden by the child class .

Return uk, Qk
input and covariance of the motion model

`GetMeasurements()`

Gets the measurement vector and the measurement noise covariance matrix from the robot. The measurement vector contains the depth read from the depth sensor and the heading read from the compass sensor.

Returns
observation vector z_k and observation noise covariance matrix R_k defined in eq. `eq-zk-EKF_4DOFAUV_VelocityMM_DVLDepthYawOM`.

Constant Velocity Motion Model

4 DOF EKF Localization with Constant Velocity Motion Model and Yaw and velocity Observation Model

class prpy.**EKF_4DOFAUV_CtVelocityMM_DVLDepthYawOM**(*kSteps*, *robot*, **args*)

Bases: *GFLocalization*, *DR_4DOFAUV_DVLGyro*, *EKF*

__init__(*kSteps*, *robot*, **args*)

Constructor.

Parameters

- **x0** – initial state
- **P0** – initial covariance
- **index** – Named tuple used to map the state vector, the simulation vector and the observation vector (`prpy.IndexStruct`)
- **kSteps** – simulation time steps
- **robot** – Simulated Robot object
- **args** – arguments to be passed to the parent constructor

f(*xk_1*, *uk*)

" Motion model of the EKF to be overwritten by the child class.

Parameters

- **xk_1** – previous mean state vector
- **uk** – input vector

Return xk_bar, Pk_bar
predicted mean state vector and its covariance matrix

Jfx(*xk_1*)

Jacobian of the motion model with respect to the state vector. *Method to be overwritten by the child class*.

Parameters
xk_1 – Linearization point. By default the linearization point is the previous state vector taken from a class attribute.

Returns
Jacobian matrix

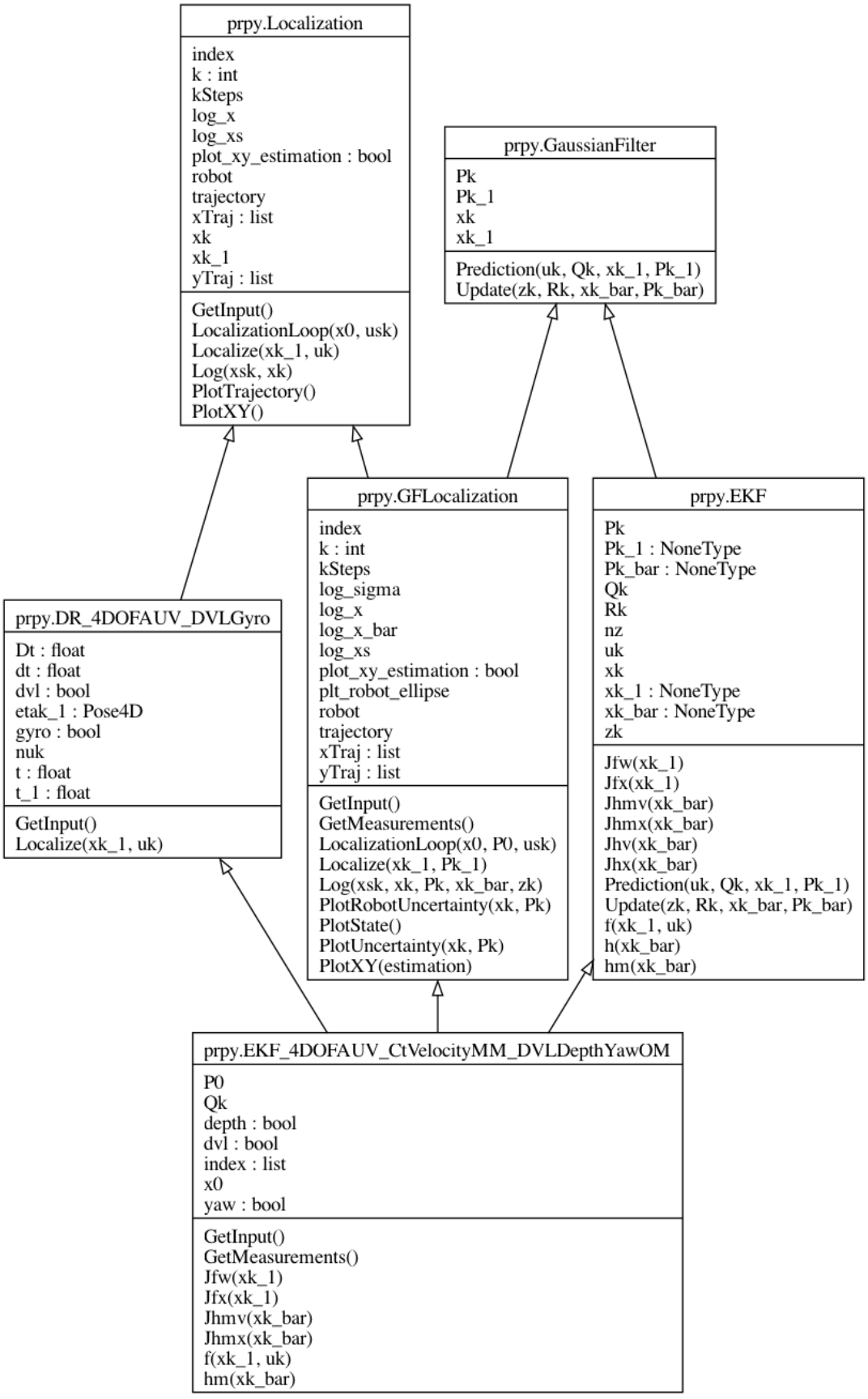


Fig. 4: EKF_4DOFAUV_CtVelocityMM_DVLDepthYawOM Class Diagram.

`Jfw`(*xk_1*)

Jacobian of the motion model with respect to the noise vector. *Method to be overwritten by the child class*.

Parameters
xk_1 – Linearization point. By default the linearization point is the previous state vector taken from a class attribute.

Returns
Jacobian matrix

`hm`(*xk_bar*)

Observation model related to the measurements observations of the EKF to be overwritten by the child class.

Parameters
xk_bar – mean of the predicted state vector. By default it is taken from the class attribute.

Returns
expected observation vector

`Jhmx`(*xk_bar*)

Jacobian of the measurement observation model with respect to the state vector. *Method to be overwritten by the child class*.

Parameters
xk_bar – linearization point. By default it is taken from the class attribute.

Returns
Jacobian matrix

`Jhmv`(*xk_bar*)

Jacobian of the measurement observation model with respect to the noise vector. *Method to be overwritten by the child class*.

Parameters
xk_bar – linearization point. By default it is taken from the class attribute.

Returns
Jacobian matrix

`GetInput`()

Get the input from the robot. Relates to the motion model as follows:

$$\begin{aligned} x_k &= f(x_{k-1}, u_k, w_k) \\ w_k &= N(0, Q_k) \end{aligned} \tag{1.31}$$

To be overidden by the child class .

Return uk, Qk
input and covariance of the motion model

`GetMeasurements`()

Get the measurements from the robot. Corresponds to the observation model:

$$\begin{aligned} z_k &= h(x_k, v_k) \\ v_k &= N(0, R_k) \end{aligned} \tag{1.32}$$

To be overidden by the child class .

Return zk, Rk
observation vector and covariance of the observation noise.

1.4.4 EKF Feature Map Based Localization

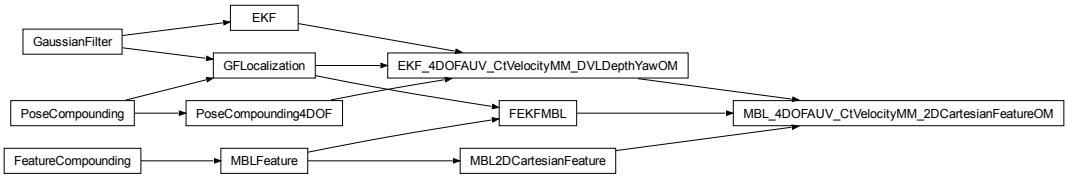
1.4.5 EKF Feature Based SLAM

1.4.6 EKF Pose Based SLAM

CHAPTER

TWO

CLASS DIAGRAM



CHAPTER

THREE

INDICES AND TABLES

- genindex
- modindex
- search

INDEX

Symbols

A

D

E

F

G

R

S

U