Odisseus Documentation

Alexandros Giavaras

Contents

1 Kinematics Model

The Extended Kalman Filter discussed in section 2.1, requires as motion model as input in order to make a predictions about the pose of the robot. This section discusses the kinematics model used by Odisseus.

1.1 Unicycle model

Odisseus is using the following unicycle model in order to capture the kinematics of the robot motion

$$\frac{dx}{dt} = vcos(\theta) \tag{1}$$

$$\frac{dy}{dt} = vsin(\theta) \tag{2}$$

$$\frac{d\theta}{dt} = \omega \tag{3}$$

where x, y are are the coordinates of the reference point, θ is the yaw angle, v is the input velocity and ω is the input angular velocity of the robotic platform.

The state vector $\mathbf{x}$ has three components; the x, y components of the reference point and the orientation or yaw angle θ. Mathematically, this is written as

$$\mathbf{x} = (x, y, \theta) \tag{4}$$

As mentioned previously, the velocity v is one of the inputs that is given to the system. Namely, it is calculated according to

$$v = \frac{v_l + v_r}{2} \tag{5}$$

where R is the wheels radius and v_r, v_l are the right and left wheels velocities respectively. Both are related to the angular wheel velocities ω_r, and ω_l respectively and the wheel radius R according to equation 6

$$v_i = \omega_i R, \quad i = r, l \tag{6}$$

Similarly the second input to the system is the angular velocity of the robot ω. This is related to v_l and v_r according to equation 7

$$\omega = \frac{v_l - v_r}{2L} \tag{7}$$

1.2 Discrete kinematic model

Equation 3 represents a continuous model. Odisseus, instead uses a discrete counterpart of the model given by the equations below.

1.2.1 Case $\omega = 0$

This case translates to the situation where the heading of the robot remains the same. In this case the model will simply update the x and y coordinates of the reference point according to the equations 8 and 9 respectively.

$$x_k = x_{k-1} + (\Delta t v_k + \mathbf{w}_{1,k}) cos(\theta_{k-1} + \mathbf{w}_{2,k}) \tag{8}$$

$$y_k = y_{k-1} + (\Delta t v_k + \mathbf{w}_{1,k}) sin(\theta_{k-1} + \mathbf{w}_{2,k}) \tag{9}$$

1.2.2 Case $\omega \neq 0$

When the ω is deemed to be non zero, then the following equations are used in order to estimate the pose of the robot.

$$\theta_k = \theta_{k-1} + \Delta t \omega_k + \mathbf{w}_{2,k} \tag{10}$$

$$x_k = x_{k-1} + (\frac{v_k}{2w_k} + \mathbf{w}_{1,k})(sin(\theta_k) - sin(\theta_{k-1})) \tag{11}$$

$$y_k = y_{k-1} - (\frac{v_k}{2w_k} + \mathbf{w}_{1,k})(cos(\theta_k) - cos(\theta_{k-1})) \tag{12}$$

Note that we first update the heading of the robot and then the x and y coordinates of the reference point.

Both scenarios incorporate the error by assuming that this is additive. The error is accounted for the linear and angular velocities. Δt is the sampling rate.

2 State Estimation

This section discusses the state estimation algorithms implemented in Odisseus.

2.1 Extended Kalman Filter

The Extended Kalman Filter is a state estimation technique for non-linear systems. It is an extension of the very popular Kalman Filter (see `https://en.wikipedia.org/wiki/Kalman_filter`). Just like the original Kalman Filter algorithm, the EKF has also two steps namely predict and update. The main difference of EKF over Kalman Filter is that it introduces a linearization of the non-linear system. Overall the algorithm is as follows

2.1.1 Predict

At this step an estimate of both the state vector $\mathbf{x}$ and the covariance matrix $\mathbf{P}$ is made. This is done according to

$$\bar{\mathbf{x}}_k = \mathbf{f}(\hat{\mathbf{x}}_{k-1}, \mathbf{u}_k, \mathbf{w}_k) \tag{13}$$

where $\mathbf{f}$ is described by equations 8, 9 and 11. $\hat{\mathbf{x}}_{k-1}$ is the state at the previous time step. $\mathbf{u}_k, \mathbf{w}$ are the input vector and error vector associated with the process. The covariance matrix is estimated via

$$\bar{\mathbf{P}}_k = \mathbf{F}_k \mathbf{P}_{k-1} \mathbf{F}_k^T + \mathbf{L}_k \mathbf{Q}_k \mathbf{L}_k^T \tag{14}$$

where $\mathbf{F}$ is the Jacobian matrix of $\mathbf{f}$ with respect to the state variables. $\mathbf{Q}_k$ is the covariance matrix of the error and $\mathbf{L}_k$ is the Jacobian matrix of the motion model, i.e. $\mathbf{f}$, with respect to $\mathbf{w}$.

2.1.2 Update

The update step established the predicted state vector and covariance matrix. Overall this step is summarized by the equations below

$$\mathbf{S}_k = \mathbf{H}_k \bar{\mathbf{P}}_k \mathbf{H}_k^T + \mathbf{M}_k \mathbf{R}_k \mathbf{M}_k^T \tag{15}$$

$$\mathbf{K}_k = \bar{\mathbf{P}}_k \mathbf{H}_k^T \mathbf{S}_k^{-1} \tag{16}$$

$$\mathbf{x}_k = \bar{\mathbf{x}}_k + \mathbf{K}(\mathbf{z}_k - \mathbf{h}(\bar{\mathbf{x}}_k, \mathbf{v}_k)) \tag{17}$$

$$\mathbf{P}_k = (\mathbf{I} - \mathbf{K}_k \mathbf{H}_k) \bar{\mathbf{P}}_k \tag{18}$$

where $\mathbf{H}$ is the Jacobian matrix of the observation model $\mathbf{h}$. $\mathbf{M}$ is the Jacobian matrix of the observation model with respect to the error vector $\mathbf{v}$. $\mathbf{K}$ is the gain matrix and $\mathbf{R}$ is the covariance matrix related to the error vector $\mathbf{v}$.

3 Sensor Modeling

$$\mathbf{h} = \begin{pmatrix} h_{sonar} \\ h_{camera} \\ h_{ir} \end{pmatrix} \tag{19}$$

Odisseus is equipped with the following three types of sensors

- Ultrasound sensor
- Camera sensor
- infrared sensor

3.1 Ultasound Sensor Model

As mentioned previously $\mathbf{h}$ represents a vector valued function and h_{sonar} is the modeled measurement from the sonar sensor. Odisseus is using the following model

$$h_{sonar}(\mathbf{x}, \mathbf{v}_{sonar}) = \sqrt{(x - x_o)^2 + (y - y_o)^2} + \mathbf{v}_{sonar} \tag{20}$$

where $\mathbf{v}_{sonar}$ is the error vector associated with the sonar. x_o, y_o are the coordinates of the obstacle detected by the sensor.

4 Software Architecture & Design

Odisseus is a multiprocess application. All its sensors as well as its motors run on a separate process. These processes are

- `MasterProcess`
- `WebAppProcess`
- `CameraProcess`
- `IRProcess`
- `UltrasoundSensorProcess`
- `PropulsionProcess`
- `DecisionMakerProcess`

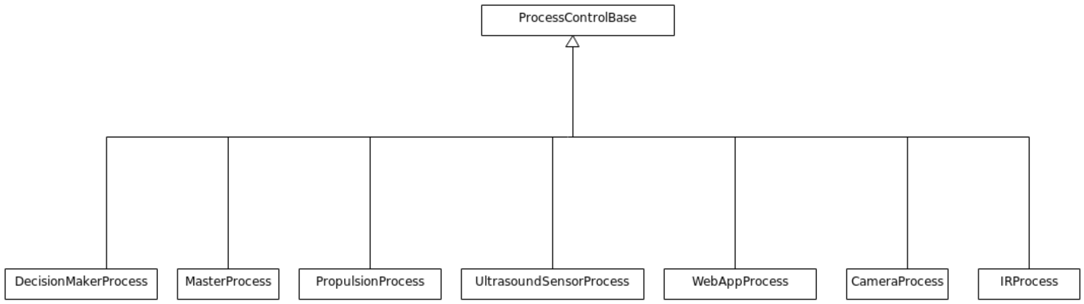

Fig. 1: Process inheritance diagram.

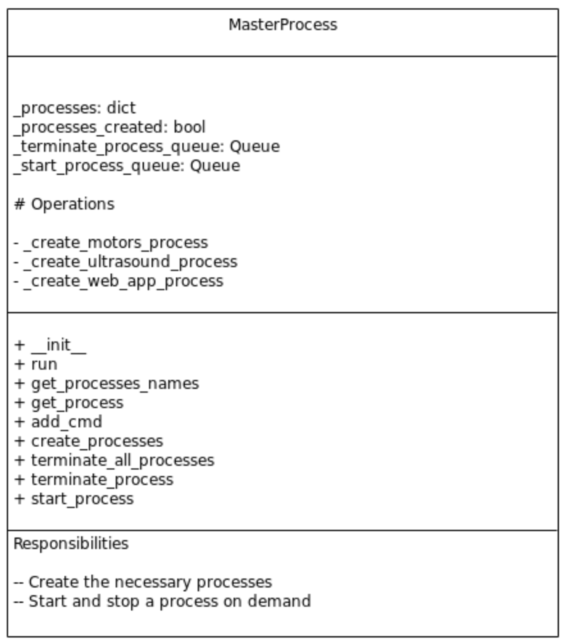

Fig. 2: `MasterProcess`.

5 Simulation Verification

5.1 EKF Verification

This section presents some simulation results that verify the EKF implementation on Odisseus

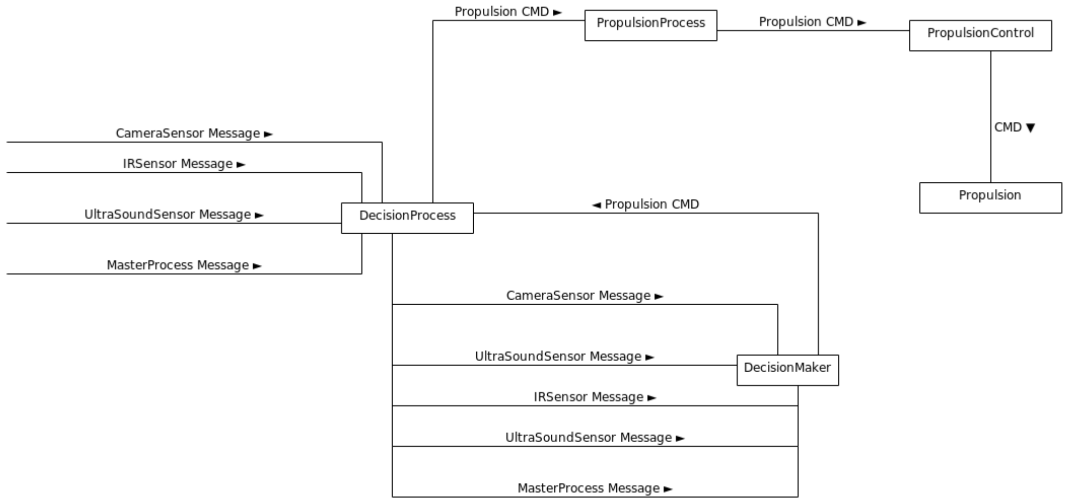

Fig. 3: Process messaging.

5.1.1 Test 1

In this test the following input data was used

$$R = \begin{pmatrix} 1.0 & 0.0 \\ 0.0 & 1.0 \end{pmatrix} \tag{21}$$

$$Q = \begin{pmatrix} 0.001 & 0.0 \\ 0.0 & 0.001 \end{pmatrix} \tag{22}$$

The motion model $\mathbf{f}$ is according to equation **??** where the error vector $\mathbf{w}$ was set to zero.

The observation model function $\mathbf{h}$ was simply the identity function meaning returning the passed state vector

$$\mathbf{h}(\mathbf{x}, \mathbf{v}) = \mathbf{x} \tag{23}$$

The error vector $\mathbf{v}$ was set to

$$\mathbf{v} = (0.0, 0.0) \tag{24}$$

Finally, the following data was used

1. $\Delta t = 0.5$
2. $R = 2.5cm$
3. $v_L = v_R = 50RPM$
4. $L = 15cm$

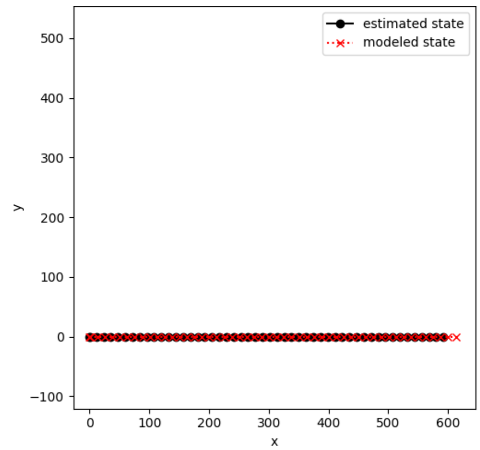

Fig. 4: Straight motion test 1.

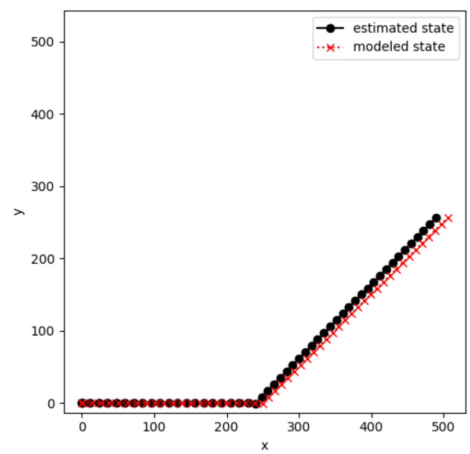

Fig. 5: Change direction test 1.

5.1.2 Test 2

The second simulation test uses equation 20 to model the sonar measurement

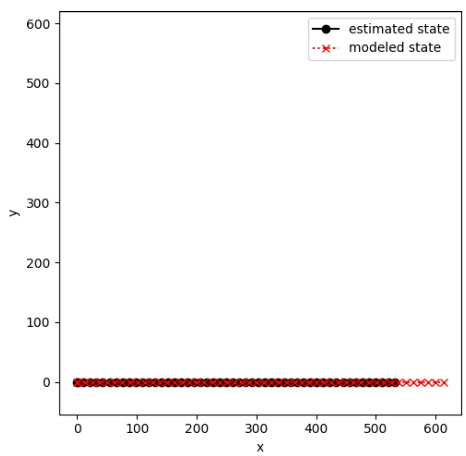

Fig. 6: Straight motion test 2.