Odisseus Documentation

Alexandros Giavaras

Contents

1 Modeling

This section discusses the modeling approach that Odisseus follows

1.1 Kinemtics Model

Odisseus is using the following kinematic model

$$\frac{dx}{dt} = vcos(\theta) \tag{1}$$
$$\frac{dy}{dt} = vsin(\theta) \tag{2}$$
$$\frac{d\theta}{dt} = \omega \tag{3}$$

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

$$v = \frac{v_L * R + v_R * R}{2} \tag{5}$$

where R is the wheels radius and v_R, v_L are the right and left wheels velocities respectively. Similarly the second input to the system is the angular velocity of the robot which is given by

$$\omega = \frac{\omega_L * R + \omega_R * R}{L} \tag{6}$$

where L is the axle length connecting the two motorized wheels. ω_L, ω_R are the angular velocities of the left and right wheels respectively.

This kinematic model is used as a motion model in the Extended Kalman Filter discussed in section 2.1. Concretely, the following discretized form is utilized

$$x_k = x_{k-1} + (\Delta t v_k + \mathbf{w}_{1,k}) cos(\theta_{k-1} + \Delta t \omega_k + \mathbf{w}_{2,k}) \tag{7}$$

$$y_k = y_{k-1} + (\Delta t v_k + \mathbf{w}_{1,k}) sin(\theta_{k-1} + \Delta t \omega_k + \mathbf{w}_{2,k}) \tag{8}$$

$$\theta_k = \theta_{k-1} + \Delta t \omega_k + \mathbf{w}_{2,k} \tag{9}$$

where Δt is the sampling rate and $\mathbf{w}$ is an error vector such that

$$E[\mathbf{w}] = \mathbf{0} \tag{10}$$

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

$$\bar{\mathbf{x}}_k = \mathbf{f}(\hat{\mathbf{x}}_{k-1}, \mathbf{u}_k, \mathbf{w}_k) \tag{11}$$

where $\mathbf{f}$ is described by equations 7, 8 and 9. $\hat{\mathbf{x}}_{k-1}$ 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

$$\bar{\mathbf{P}}_k = \mathbf{F}_k \mathbf{P}_{k-1} \mathbf{F}_k^T + \mathbf{L}_k \mathbf{Q}_k \mathbf{L}_k^T \tag{12}$$

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

$$\mathbf{S}_k = \mathbf{H}_k \bar{\mathbf{P}}_k \mathbf{H}_k^T + \mathbf{M}_k \mathbf{R}_k \mathbf{M}_k^T \tag{13}$$

$$\mathbf{K}_k = \bar{\mathbf{P}}_k \mathbf{H}_k^T \mathbf{S}_k^{-1} \tag{14}$$

$$\mathbf{x}_k = \bar{\mathbf{x}}_k + \mathbf{K}(\mathbf{z}_k - \mathbf{h}(\bar{\mathbf{x}}_k, \mathbf{v}_k)) \tag{15}$$

$$\mathbf{P}_k = (\mathbf{I} - \mathbf{K}_k \mathbf{H}_k)\bar{\mathbf{P}}_k \tag{16}$$

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

$$\mathbf{h} = \begin{pmatrix} h_{sonar} \\ h_{camera} \\ h_{ir} \end{pmatrix} \tag{17}$$

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

$$h_{sonar}(\mathbf{x}, \mathbf{v}_{sonar}) = \sqrt{(x - x_o)^2 + (y - y_o)^2} + \mathbf{v}_{sonar} \tag{18}$$

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`

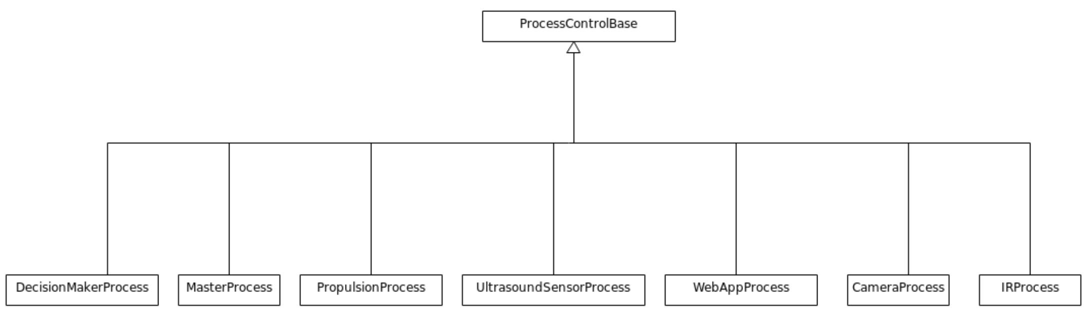

Fig. 1: Process inheritance diagram.

5 Simulation Verification

5.1 EKF Verification

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

5.1.1 Test 1

In this test the following input data was used

$$R = \begin{pmatrix} 1.0 & 0.0 \\ 0.0 & 1.0 \end{pmatrix} \tag{19}$$

$$Q = \begin{pmatrix} 0.001 & 0.0 \\ 0.0 & 0.001 \end{pmatrix} \tag{20}$$

Fig. 2: `MasterProcess`.

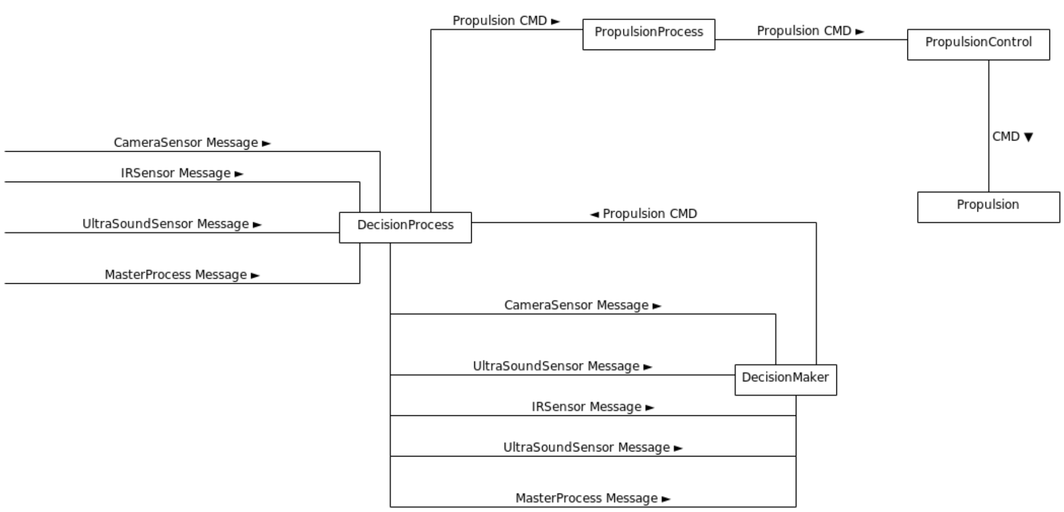

Fig. 3: Process messaging.

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

$$\mathbf{h}(\mathbf{x}, \mathbf{v}) = \mathbf{x} \tag{21}$$

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

$$\mathbf{v} = (0.0, 0.0) \tag{22}$$

Finally, the following data was used

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

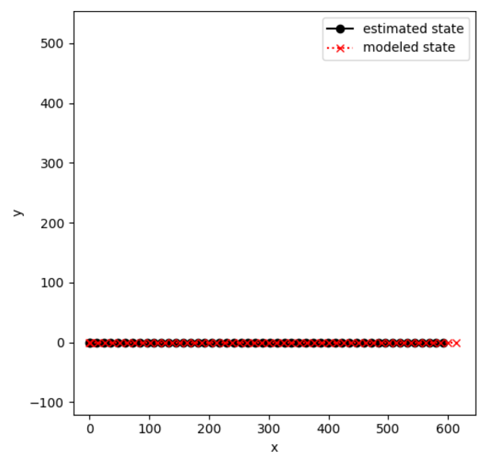

Fig. 4: Straight motion test 1.

5.1.2 Test 2

The second simulation test uses equation 18 to model the sonar measurement

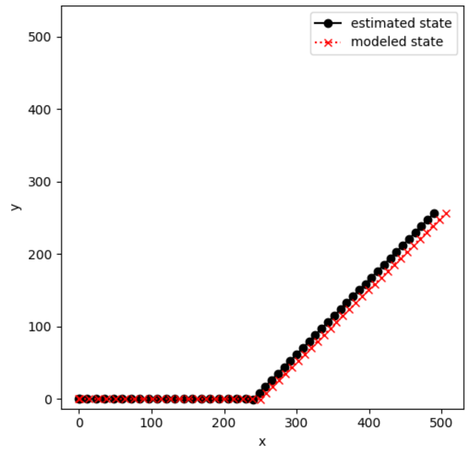

Fig. 5: Change direction test 1.

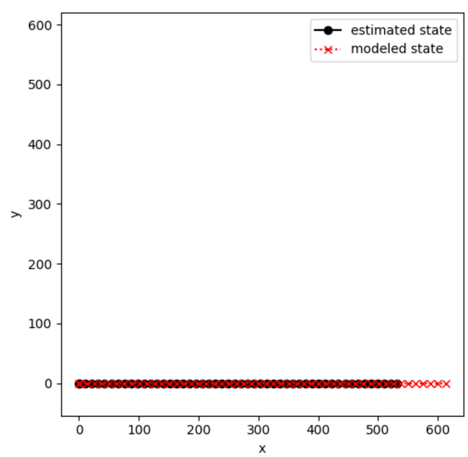

Fig. 6: Straight motion test 2.