The Extended Kalman-Bucy Filter

Alexander Wittmond

University of Missouri

October 19, 2020

Extension to Non-Linear Problems

We start with the system

$$dx_t = f(x_t, t)dt + G(t)d\beta_t \tag{1}$$
$$y_k = h(x_{t_k}, t) + \nu_t \tag{2}$$

where

$$\mathbb{E}[d\beta_t d\beta_t^T] = Q(t)dt \tag{3}$$

Extension to Non-Linear Problems

We start with the system

$$dx_t = f(x_t, t)dt + G(t)d\beta_t \tag{1}$$
$$y_k = h(x_{t_k}, t) + \nu_t \tag{2}$$

where

$$\mathbb{E}[d\beta_t d\beta_t^T] = Q(t)dt \tag{3}$$

Then we pick a **reference trajectory** $\bar{x}_{t_0}(t)$ with some given $\bar{x}(t_0)$ such that

$$\frac{d\bar{x}_{t_0}}{dt}(t) = f(\bar{x}_{t_0}(t), t) \tag{4}$$

Extension to Non-Linear Problems

Then we can look at the **deviation**

$$\delta x(t) = x(t) - \bar{x}_{t_0}(t) \tag{5}$$

which statisfies

Extension to Non-Linear Problems

Then we can look at the **deviation**

$$\delta x(t) = x(t) - \bar{x}_{t_0}(t) \tag{5}$$

which statisfies

$$d(\delta x_t) = (f(x_t, t) - f(\bar{x}_{t_0}(t), t))dt + G(t)d\beta_t \tag{6}$$

Extension to Non-Linear Problems

Then we can look at the **deviation**

$$\delta x(t) = x(t) - \bar{x}_{t_0}(t) \tag{5}$$

which statisfies

$$d(\delta x_t) = (f(x_t, t) - f(\bar{x}_{t_0}(t), t))dt + G(t)d\beta_t \tag{6}$$

then do a Taylor approximation

$$f(x_t, t) - f(\bar{x}_t, t) \simeq Df(\bar{x}_{t_0}(t), t)\delta x_t \tag{7}$$

Extension to Non-Linear Problems

Then we can look at the **deviation**

$$\delta x(t) = x(t) - \bar{x}_{t_0}(t) \tag{5}$$

which statisfies

$$d(\delta x_t) = (f(x_t, t) - f(\bar{x}_{t_0}(t), t))dt + G(t)d\beta_t \tag{6}$$

then do a Taylor approximation

$$f(x_t, t) - f(\bar{x}_t, t) \simeq Df(\bar{x}_{t_0}(t), t)\delta x_t \tag{7}$$

Then we have the linear equation

$$d(\delta x_t) = Df(\bar{x}_{t_0}, t)\delta x_t dt + G(t)d\beta_t \tag{8}$$

Extension to Non-Linear Problems

We can linearize the measurement in the same way to get

$$\delta y_{t_k} = y_{t_k} - h(\bar{x}_{t_0}(t_k), t_k) + \nu_k \tag{9}$$

$$\delta y_{t_k} \simeq Dh(\bar{x}_{t_0}(t_k), t_k)\delta x_{t_k} + \nu_k \tag{10}$$

Extension to Non-Linear Problems

We can linearize the measurement in the same way to get

$$\delta y_{t_k} = y_{t_k} - h(\bar{x}_{t_0}(t_k), t_k) + \nu_k \tag{9}$$

$$\delta y_{t_k} \simeq Dh(\bar{x}_{t_0}(t_k), t_k)\delta x_{t_k} + \nu_k \tag{10}$$

Then we can process the system

$$d(\delta x_t) = Df(\bar{x}_{t_0}(t), t)\delta x_t dt + G(t) d\beta_t \tag{11}$$

$$\delta y_{t_k} = Dh(\bar{x}_{t_0}(t_k), t_k)\delta x_{t_k} + \nu_k \tag{12}$$

with linear filter.

Extension to Non-Linear Problems

We can then estimate $\hat{x}_{t_k}^{t_k}$ with

$$\hat{x}_{t_k}^{t_k} = \bar{x}_{t_0}(t_k) + \widehat{\delta x}_{t_k}^{t_k} \tag{13}$$

Our variance matrix $P_{t_k}^{t_k}$ estimates the variance of this estimation.

Extension to Non-Linear Problems

If at every observation $h(t_k)$, we relinearize around our estimate $\hat{x}_{t_k}^{t_k}$ by getting a new reference trajectory $\bar{x}_{t_k}(t)$ starting at this point, and then estimate the system using

$$d(\delta x_t) = Df(\bar{x}_{t_k}(t), t_k)\delta x_t dt + G(t) d\beta_t \tag{14}$$

$$\delta y_{t_{k+1}} \simeq Dh(\bar{x}_{t_k}(t_k), t_k)\delta x_{t_k} + \nu_k \tag{15}$$

then this is known as the **Extended Kalman Filter**

In Code

```cpp
arma::dvec ExtendedKalmanFilter::update(double t, arma::vec observation) {
  arma::dvec nominal_state = model->extrapolate(t );

  arma::dvec measurement_error = observation - measurement->measure(t , nominal_state);

  arma::dvec error = perturbationProcessFilter.update(t, measurement_error);

  arma::dvec state_estimate = nominal_state + error;

  model->set_initial_conditions(t , state_estimate);

  perturbationProcessFilter.get_model()->set_initial_conditions
    (t,arma::dvec(nominal_state.size(),arma::fill::zeros));


  return state_estimate;
}
```

Figure: The Extended Kalman Filter update in C++

Pros and Cons

Pros

Pros and Cons

Pros

- Recursiveness causes low memory requirements

Pros and Cons

Pros

- Recursiveness causes low memory requirements
- Relatively low cost to computing an update

Pros and Cons

Pros

- Recursiveness causes low memory requirements
- Relatively low cost to computing an update
- Most computationally expensive items can be computed in advance

Pros and Cons

Pros

- Recursiveness causes low memory requirements
- Relatively low cost to computing an update
- Most computationally expensive items can be computed in advance

Cons

Pros and Cons

Pros

- Recursiveness causes low memory requirements
- Relatively low cost to computing an update
- Most computationally expensive items can be computed in advance

Cons

- Is not adaptive, relies on the accuracy of the underlying model

Pros and Cons

Pros

- Recursiveness causes low memory requirements
- Relatively low cost to computing an update
- Most computationally expensive items can be computed in advance

Cons

- Is not adaptive, relies on the accuracy of the underlying model
- Suffers from filter divergence

Pros and Cons

Pros

- Recursiveness causes low memory requirements
- Relatively low cost to computing an update
- Most computationally expensive items can be computed in advance

Cons

- Is not adaptive, relies on the accuracy of the underlying model
- Suffers from filter divergence
- Can not capture multi-modalities