Kalman

1.0

Generated by Doxygen 1.8.9.1

Thu Jun 25 2015 13:57:10

Contents

Chapter 1

Hierarchical Index

1.1 Class Hierarchy

This inheritance list is sorted roughly, but not completely, alphabetically:

Chapter 2

Class Index

2.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

Chapter 3

File Index

3.1 File List

Here is a list of all documented files with brief descriptions:

Chapter 4

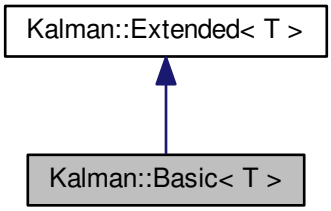
Class Documentation

4.1 Kalman::Basic< T > Class Template Reference

Generic Linear Kalman Filter template base class.

`#include <basic.h>`

Collaboration diagram for Kalman::Basic< T >:



Public Member Functions

- virtual ~Basic ()=0

  Virtual destructor.
- void init (Vector< T > &x_, Matrix< T > &P_)

  Sets initial conditions for the Kalman Filter.

Dimension Accessor Functions

- unsigned getSizeX () const

  Returns the size of the state vector.
- unsigned getSizeU () const

  Returns the size of the input vector.
- unsigned getSizeW () const

  Returns the size of the process noise vector.
- unsigned getSizeZ () const

  Returns the size of the measurement vector.

- unsigned getSizeV () const

  Returns the size of the measurement noise vector.

Resizing Functions

These functions allow to change the dimensions of all matrices and vectors, thus implementing a Variable-Dimension Extended Kalman Filter. They do nothing if the new size is the same as the old one.

Warning

`setSize*` *functions* ***must*** *be called* ***before*** *any other function, or else, matrices and vectors will not have their memory allocated.*

- void setSizeX (unsigned n_)

  Sets the size of the state vector.
- void setSizeU (unsigned nu_)

  Sets the size of the input vector.
- void setSizeW (unsigned nw_)

  Sets the size of the process noise vector.
- void setSizeZ (unsigned)

  Sets the size of the measurement vector.
- void setSizeV (unsigned)

  Sets the size of the measurement noise vector.

Kalman Filter Functions

These functions allow to get the results from the Kalman filtering algorithm. Before any of these can be called, all dimensions must have been set properly at least once and `init()` must have been called, also at least once. Each time the user want to resize some vectors, the corresponding resizing functions must be called again before being able to call one of the functions in this section. `init()` must also be called again if n or nw has changed. `init()` can also be called solely to reset the filter.

- void step (Vector< T > &u_, const Vector< T > &z_)

  Makes one prediction-correction step.
- void timeUpdateStep (Vector< T > &u_)

  Makes one prediction step.
- void measureUpdateStep (const Vector< T > &z_)

  Makes one correction step.
- const Vector< T > & predict (Vector< T > &u_)

  Returns the predicted state vector (a priori state estimate).
- const Vector< T > & simulate ()

  Returns the predicted measurement vector.
- const Vector< T > & getX () const

  Returns the corrected state (a posteriori state estimate).
- const Matrix< T > & calculateP () const

  Returns the a posteriori error covariance estimate matrix.

Protected Member Functions

- virtual void makeBaseB ()

  Virtual pre-creator of B.
- virtual void makeB ()

  Virtual creator of B.
- virtual void makeProcess ()

  Process function overridden to be linear.
- virtual void makeMeasure ()

  Measurement function overridden to be linear.
- virtual void sizeUpdate ()

Matrix and vector resizing function, overridden to take B into account.

Matrix Pre-Creators

Theses functions have been designed to be overridden by derived classes if necessary. Their role is to fill in the parts of the Kalman matrices that don't change between iterations. That is to say, these functions should only set constant values inside matrices that don't depend on x or u.

They will all be called at least once, before the calls to their corresponding matrix (not pre-) creators. In fact, they are called once per resize (not necessarily at the moment of the resize though), including while the matrices are first allocated.

Note

Matrices have already been properly resized before these functions are called, so no further resizing is or should be necessary.

Warning

Each matrix pre-creator cannot suppose that any other matrix pre-creator will be called before or after it.

- virtual void makeBaseA ()
  Virtual pre-creator of A.
- virtual void makeBaseW ()
  Virtual pre-creator of W.
- virtual void makeBaseQ ()
  Virtual pre-creator of Q.
- virtual void makeBaseH ()
  Virtual pre-creator of H.
- virtual void makeBaseV ()
  Virtual pre-creator of V.
- virtual void makeBaseR ()
  Virtual pre-creator of R.

Matrix Creators

Theses functions have been designed to be overridden by derived classes if necessary. Their role is to fill in the parts of the Kalman matrices that change between iterations. That is to say, these functions should set values inside matrices that depend on x or u.

These functions can suppose that their corresponding matrix pre-creator has been called at least once before. Also, `makeA()`, `makeW()`, `makeQ()` and `makeProcess()` can suppose that `makeCommonProcess()` is called every time just before it being called. Same thing for `makeH()`, `makeV()`, `makeR()` and `makeMeasure()` about `makeCommonMeasure()`.

Note

Matrices have already been properly resized before these functions are called, so no further resizing is or should be necessary.

Warning

Each matrix creator cannot suppose that any other matrix creator will be called before or after it. One thing is sure : `makeCommon*()` is called first, then some of `make*()` and finally, `makeProcess()` or `makeMeasure()`.
These functions can access x and u in read-only mode, except makeProcess(), which must modify x.

- virtual void makeCommonProcess ()
  Optional function used to precalculate common values for process.
- virtual void makeA ()
  Virtual creator of A.
- virtual void makeW ()
  Virtual creator of W.

- virtual void makeQ ()
  Virtual creator of Q.
- virtual void makeCommonMeasure ()
  Optional function used to precalculate common values for measurement.
- virtual void makeH ()
  Virtual creator of H.
- virtual void makeV ()
  Virtual creator of V.
- virtual void makeR ()
  Virtual creator of R.
- virtual void makeDZ ()
  Hook-up function to modify innovation vector.

Protected Attributes

- Vector< T > px
  Temporary vector.
- Matrix< T > B
  Input matrix.

Kalman Vectors and Matrices

- Vector< T > x
  Corrected state vector.
- Vector< T > u
  Input vector.
- Vector< T > z
  Predicted measurement vector.
- Vector< T > dz
  Innovation vector.
- Matrix< T > A
  A jacobian matrix.
- Matrix< T > W
  A jacobian matrix.
- Matrix< T > Q
  Process noise covariance matrix.
- Matrix< T > H
  A jacobian matrix.
- Matrix< T > V
  A jacobian matrix.
- Matrix< T > R
  Measurement noise covariance matrix.

Kalman Dimensions

Warning

These values, which are accessible to derived classes, are read-only. The derived classes should use the resizing functions to modify vector and matrix dimensions.

- unsigned n
  Size of the state vector.
- unsigned nu
  Size of the input vector.
- unsigned nw
  Size of the process noise vector.
- unsigned m
  Size of the measurement vector.
- unsigned nv
  Size of the measurement noise vector.

4.1.1 Detailed Description

template<typename T>class Kalman::Basic< T >

Generic Linear Kalman Filter template base class.

Usage

This class is derived from `Extended`. You should really read all the documentation of `Extended` before reading this.
This class implements a Variable-Dimension Linear Kalman Filter based on the `Extended` template class.

Notation

Assume a state vector x (to estimate) and a linear process function f (to model) that describes the evolution of this state through time, that is :

$$x_k = f(x_{k-1}, u_{k-1}, w_{k-1}) = Ax_{k-1} + Bu_{k-1} + Ww_{k-1}$$

where u is the (known) input vector fed to the process and w is the (unknown) process noise vector due to uncertainty and process modeling errors. Further suppose that the (known) process noise covariance matrix is :

$$Q = E\left(ww^T\right)$$

Now, let's assume a (known) measurement vector z, which depends on the current state x in the form of a linear function h (to model) :

$$z_k = h(x_k, v_k) = Hx_k + Vv_k$$

where v is the (unknown) measurement noise vector with a (known) covariance matrix :

$$R = E\left(vv^T\right)$$

Suppose that we have an estimate of the previous state $\hat{x}_{k-1}$, called a corrected state or an *a posteriori* state estimate. We can build a predicted state (also called an *a priori* state estimate) by using f:

$$\tilde{x}_k = f(\hat{x}_{k-1}, u_{k-1}, 0) = A\hat{x}_{k-1} + Bu_{k-1}$$

since the input is known and the process noise, unknown. With this predicted state, we can get a predicted measurement vector by using h :

$$\tilde{z}_k = h(\tilde{x}_k, 0) = H\tilde{x}_k$$

since the measurement noise is unknown.

Note

In this class, `makeProcess()` and `makeMeasure()` have already been overridden, and cannot be ovver-ridden again. However, there is a new matrix to create: *B*. This means there are two new virtual functions that can be overridden: `makeBaseB()` and `makeB()`.

See also

`Extended`

4.1.2 Member Function Documentation

4.1.2.1 template<typename T > void Kalman::Extended< T >::setSizeX (unsigned *n_*) [inherited]

Sets the size of the state vector.

Parameters

*n_*	New state vector size. Must not be 0.

Warning

`init()` must always be called after this function and before any other non-dimensioning function.

Referenced by Kalman::Extended< T >::Extended().

4.1.2.2 template<typename T > void Kalman::Extended< T >::setSizeW (unsigned *nw_*) [inherited]

Sets the size of the process noise vector.

Parameters

*nw_*	New process noise vector size.

Warning

`init()` must always be called after this function and before any other non-dimensioning function.

Referenced by Kalman::Extended< T >::Extended().

4.1.2.3 template<typename T > void Kalman::Extended< T >::init (Vector< T > & *x_*, Matrix< T > & *P_*) [inherited]

Sets initial conditions for the Kalman Filter.

This function allows to set an initial state estimate vector and an initial error covariance matrix estimate. This must be called at least once, after all dimensioning functions and before any other function. However, it can also be called anytime to reset or modify *x* or *P*.

Parameters

*x_*	State vector estimate. Will be destroyed.
*P_*	Error covariance matrix estimate. Will be destroyed.

Warning

If `setSizeX()` or `setSizeW()` is called, then init() must be called again before any other non-dimensioning function.

4.1.2.4 template<typename T > void Kalman::Extended< T >::step (Vector< T > & *u_*, const Vector< T > & *z_*) [inherited]

Makes one prediction-correction step.

This is the main `Extended` function. First, it resizes any matrix who needs it. Then, it proceeds to the time update phase, using the input vector `u_`. This means that the following virtual functions *should be* called : `makeCommonProcess()`, `makeA()`, `makeW()`, `makeQ()` and `makeProcess()`. At this stage, *x* contains a current predicted state instead of an old corrected state. If `z_` is empty, that is, if there are no measures in this step, there is no correction and the function stops there. Else, the measure update phase begins. This means that the following virtual functions *should be* called : `makeCommonMeasure()`, `makeHImpl()`, `makeVImpl()`, `makeRImpl()`, `makeMeasure()` and `makeDZ()`.After this phase, *x* contains the new corrected state.

Parameters

*u_*	Input vector. Will **not** be destroyed. Can be empty.
*z_*	Measurement vector. Will **not** be destroyed. Can be empty.

4.1.2.5 template<typename T > void Kalman::Extended< T >::timeUpdateStep (Vector< T > & *u_*) [inherited]

Makes one prediction step.

This function first resizes any matrix who needs it. Then, it proceeds to the time update phase, using the input vector `u_`. This means that the following virtual functions *should be* called : `makeCommonProcess()`, `makeA()`, `makeW()`, `makeQ()` and `makeProcess()`. At this stage, *x* contains a current predicted state instead of an old corrected state.

Parameters

*u_*	Input vector. Will **not** be destroyed. Can be empty.

4.1.2.6 template<typename T > void Kalman::Extended< T >::measureUpdateStep (const Vector< T > & *z_*) [inherited]

Makes one correction step.

First, this function resizes any matrix who needs it. If `z_` is empty, that is, if there are no measures in this step, there is no correction and the function stops there. Else, the measure update phase begins. This means that the following virtual functions *should be* called : `makeCommonMeasure()`, `makeHImpl()`, `makeVImpl()`, `makeRImpl()`, `makeMeasure()` and `makeDZ()`.After this phase, *x* contains the new corrected state.

Parameters

*z_*	Measurement vector. Will **not** be destroyed. Can be empty.

4.1.2.7 template<typename T > const Vector< T > & Kalman::Extended< T >::predict (Vector< T > & *u_*) [inherited]

Returns the predicted state vector (*a priori* state estimate).

This function is used to predict a future state. First, it resizes any matrix who needs it. Then, it does a partial time update, in the sense that only *x* is updated, not P. This also means that only the following virtual functions *should be* called : `makeCommonProcess()` and `makeProcess()`.

Parameters

*u_*	Input vector. Will **not** be destroyed. Can be empty.

Warning

For better efficiency, the prediction is returned by reference.

4.1.2.8 template<typename T > const Vector< T > & Kalman::Extended< T >::simulate ( ) [inherited]

Returns the predicted measurement vector.

This function is used to predict a future measurement. First, it resizes any matrix who needs it. Then, it does a partial measure update, in the sense that only *z* is calculated : *x* and P are not updated. This also means that only the following virtual functions *should be* called : `makeCommonMeasure()` and `makeMeasure()`.

Warning

For better efficiency, the prediction is returned by reference.

4.1.2.9 template<typename T > const Matrix< T > & Kalman::Extended< T >::calculateP ( ) const [inherited]

Returns the *a posteriori* error covariance estimate matrix.

Warning

This is not a simple return statement. Since P is not kept and updated in the filter (an alternate and more stable representation of P is used), calculations are involved to retrieve P. So, use this function wisely.
For better efficiency, P is returned by reference.

4.1.2.10 template<typename T > void Kalman::Extended< T >::makeCommonProcess ( ) [protected], [virtual], [inherited]

Optional function used to precalculate common values for process.

If complex calculations are needed for more than one of `makeA()`, `makeW()`, `makeQ()` and `makeProcess()` functions, then this function can be used to store the results in temporary variables of the derived class.

Warning

This function must not modify any matrix of the base class.
This function must not be used to store permanent state. In other words, all calculations performed in this function should be temporary. This is because the `predict()` function will call this function but has no knowledge of how to undo it.

4.1.2.11 template<typename T > void Kalman::Extended< T >::makeCommonMeasure ( ) [protected], [virtual], [inherited]

Optional function used to precalculate common values for measurement.

If complex calculations are needed for more than one of `makeH()`, `makeV()`, `makeR()`, `makeMeasure()` and `makeDZ()` functions, then this function can be used to store the results in temporary variables of the derived class.

Warning

This function must not modify any matrix of the base class.
This function must not be used to store permanent state. In other words, all calculations performed in this function should be temporary. This is because the `simulate()` function will call this function but has no knowledge of how to undo it.

4.1.2.12 template<typename T > void Kalman::Extended< T >::makeDZ ( ) [protected], [virtual], [inherited]

Hook-up function to modify innovation vector.

This function should rarely be overridden ; this is more of a hack than anything else. In fact, this is used to perform adjustements on the result of substracting the predicted measurement vector to the real measurement vector. This is needed, for example, when measures include angles. It may be mandatory that the difference of the two angles be in a certain range, like $[-\pi, \pi]$.

4.1.3 Member Data Documentation

4.1.3.1 template<typename T > Vector<T> Kalman::Extended< T >::x [protected], [inherited]

Corrected state vector.

This is an *n-sized* vector. Derived classes should modify it only through `makeProcess()`.

4.1.3.2 template<typename T > Vector<T> Kalman::Extended< T >::u [protected], [inherited]

Input vector.

This is an *nu-sized* vector. Derived classes should never modify it.

4.1.3.3 template<typename T > Vector<T> Kalman::Extended< T >::z [protected], [inherited]

Predicted measurement vector.

This is an *m-sized* vector. Derived classes should modify it only through `makeMeasure()`.

4.1.3.4 template<typename T > Vector<T> Kalman::Extended< T >::dz [protected], [inherited]

Innovation vector.

This is an *m-sized* vector. Derived classes should modify it only through `makeDZ()`. The innovation vector is the difference between the real measurement vector and the predicted one.

4.1.3.5 template<typename T > Matrix<T> Kalman::Extended< T >::A [protected], [inherited]

A jacobian matrix.

This is an *n* by *n* jacobian matrix of partial derivatives, defined as follow :

$$A_{[i,j]} = \frac{\partial f_{[i]}}{\partial x_{[j]}}$$

Derived classes should modify it only through `makeBaseA()` for the constant part and `makeA()` for the variable part.

4.1.3.6 template<typename T > Matrix<T> Kalman::Extended< T >::W [protected], [inherited]

A jacobian matrix.

This is an *n* by *nw* jacobian matrix of partial derivatives, defined as follow :

$$W_{[i,j]} = \frac{\partial f_{[i]}}{\partial w_{[j]}}$$

Derived classes should modify it only through `makeBaseW()` for the constant part and `makeW()` for the variable part.

4.1.3.7 template<typename T > Matrix<T> Kalman::Extended< T >::Q [protected], [inherited]

Process noise covariance matrix.

This is the *nw* by *nw* covariance matrix of *w*, that is :

$$Q = E\left(ww^T\right)$$

Derived classes should modify it only through `makeBaseQ()` for the constant part and `makeQ()` for the variable part.

4.1.3.8 template<typename T > Matrix<T> Kalman::Extended< T >::H [protected], [inherited]

A jacobian matrix.

This is an *m* by *n* jacobian matrix of partial derivatives, defined as follow :

$$H_{[i,j]} = \frac{\partial h_{[i]}}{\partial x_{[j]}}$$

Derived classes should modify it only through `makeBaseH()` for the constant part and `makeH()` for the variable part.

4.1.3.9 template<typename T > Matrix<T> Kalman::Extended< T >::V [protected], [inherited]

A jacobian matrix.

This is an *m* by *nv* jacobian matrix of partial derivatives, defined as follow :

$$V_{[i,j]} = \frac{\partial h_{[i]}}{\partial v_{[j]}}$$

Derived classes should modify it only through `makeBaseV()` for the constant part and `makeV()` for the variable part.

4.1.3.10 template<typename T > Matrix<T> Kalman::Extended< T >::R [protected], [inherited]

Measurement noise covariance matrix.

This is the *nv* by *nv* covariance matrix of *v*, that is :

$$R = E\left(vv^T\right)$$

Derived classes should modify it only through `makeBaseR()` for the constant part and `makeR()` for the variable part.

The documentation for this class was generated from the following file:

- basic.h

4.2 Kalman::Extended< T > Class Template Reference

Generic Extended Kalman Filter template base class.

`#include <extended.h>`

Public Member Functions

- void init (Vector< T > &x_, Matrix< T > &P_)
  Sets initial conditions for the Kalman Filter.

Constructor and Destructor.

- Extended ()
  Default constructor.
- Extended (unsigned n_, unsigned nu_, unsigned nw_, unsigned m_, unsigned nv_)
  Constructor specifying all necessary matrix and vector dimensions.
- virtual ~Extended ()
  Virtual destructor.

Dimension Accessor Functions

- unsigned getSizeX () const

  Returns the size of the state vector.
- unsigned getSizeU () const

  Returns the size of the input vector.
- unsigned getSizeW () const

  Returns the size of the process noise vector.
- unsigned getSizeZ () const

  Returns the size of the measurement vector.
- unsigned getSizeV () const

  Returns the size of the measurement noise vector.

Resizing Functions

These functions allow to change the dimensions of all matrices and vectors, thus implementing a Variable-Dimension Extended Kalman Filter. They do nothing if the new size is the same as the old one.

Warning

> `setSize*` *functions* ***must*** *be called* ***before*** *any other function, or else, matrices and vectors will not have their memory allocated.*

- void setSizeX (unsigned n_)

  Sets the size of the state vector.
- void setSizeU (unsigned nu_)

  Sets the size of the input vector.
- void setSizeW (unsigned nw_)

  Sets the size of the process noise vector.
- void setSizeZ (unsigned)

  Sets the size of the measurement vector.
- void setSizeV (unsigned)

  Sets the size of the measurement noise vector.

Kalman Filter Functions

These functions allow to get the results from the Kalman filtering algorithm. Before any of these can be called, all dimensions must have been set properly at least once and `init()` must have been called, also at least once. Each time the user want to resize some vectors, the corresponding resizing functions must be called again before being able to call one of the functions in this section. `init()` must also be called again if n or nw has changed. `init()` can also be called solely to reset the filter.

- void step (Vector< T > &u_, const Vector< T > &z_)

  Makes one prediction-correction step.
- void timeUpdateStep (Vector< T > &u_)

  Makes one prediction step.
- void measureUpdateStep (const Vector< T > &z_)

  Makes one correction step.
- const Vector< T > & predict (Vector< T > &u_)

  Returns the predicted state vector (a priori state estimate).
- const Vector< T > & simulate ()

  Returns the predicted measurement vector.
- const Vector< T > & getX () const

  Returns the corrected state (a posteriori state estimate).
- const Matrix< T > & calculateP () const

  Returns the a posteriori error covariance estimate matrix.

Protected Member Functions

- virtual void sizeUpdate ()

  *Resizes all vector and matrices. **Never** call or overload this !*

Matrix Pre-Creators

Theses functions have been designed to be overridden by derived classes if necessary. Their role is to fill in the parts of the Kalman matrices that don't change between iterations. That is to say, these functions should only set constant values inside matrices that don't depend on x or u.

They will all be called at least once, before the calls to their corresponding matrix (not pre-) creators. In fact, they are called once per resize (not necessarily at the moment of the resize though), including while the matrices are first allocated.

Note

Matrices have already been properly resized before these functions are called, so no further resizing is or should be necessary.

Warning

Each matrix pre-creator cannot suppose that any other matrix pre-creator will be called before or after it.

- virtual void makeBaseA ()

  Virtual pre-creator of A.
- virtual void makeBaseW ()

  Virtual pre-creator of W.
- virtual void makeBaseQ ()

  Virtual pre-creator of Q.
- virtual void makeBaseH ()

  Virtual pre-creator of H.
- virtual void makeBaseV ()

  Virtual pre-creator of V.
- virtual void makeBaseR ()

  Virtual pre-creator of R.

Matrix Creators

Theses functions have been designed to be overridden by derived classes if necessary. Their role is to fill in the parts of the Kalman matrices that change between iterations. That is to say, these functions should set values inside matrices that depend on x or u.

These functions can suppose that their corresponding matrix pre-creator has been called at least once before. Also, `makeA()`, `makeW()`, `makeQ()` and `makeProcess()` can suppose that `makeCommonProcess()` is called every time just before it being called. Same thing for `makeH()`, `makeV()`, `makeR()` and `makeMeasure()` about `makeCommonMeasure()`.

Note

Matrices have already been properly resized before these functions are called, so no further resizing is or should be necessary.

Warning

Each matrix creator cannot suppose that any other matrix creator will be called before or after it. One thing is sure : `makeCommon*()` is called first, then some of `make*()` and finally, `makeProcess()` or `makeMeasure()`.
These functions can access x and u in read-only mode, except makeProcess(), which must modify x.

- virtual void makeCommonProcess ()

  Optional function used to precalculate common values for process.
- virtual void makeA ()

Virtual creator of A.

- virtual void makeW ()
  Virtual creator of W.
- virtual void makeQ ()
  Virtual creator of Q.
- virtual void makeProcess ()=0
  Actual process $f(x, u, 0)$. Fills in new x by using old x.
- virtual void makeCommonMeasure ()
  Optional function used to precalculate common values for measurement.
- virtual void makeH ()
  Virtual creator of H.
- virtual void makeV ()
  Virtual creator of V.
- virtual void makeR ()
  Virtual creator of R.
- virtual void makeMeasure ()=0
  Actual measurement function $h(x, 0)$. Fills in z.
- virtual void makeDZ ()
  Hook-up function to modify innovation vector.

Protected Attributes

Kalman Vectors and Matrices

- Vector< T > x
  Corrected state vector.
- Vector< T > u
  Input vector.
- Vector< T > z
  Predicted measurement vector.
- Vector< T > dz
  Innovation vector.
- Matrix< T > A
  A jacobian matrix.
- Matrix< T > W
  A jacobian matrix.
- Matrix< T > Q
  Process noise covariance matrix.
- Matrix< T > H
  A jacobian matrix.
- Matrix< T > V
  A jacobian matrix.
- Matrix< T > R
  Measurement noise covariance matrix.

Kalman Dimensions

Warning

These values, which are accessible to derived classes, are read-only. The derived classes should use the resizing functions to modify vector and matrix dimensions.

- unsigned n
  Size of the state vector.
- unsigned nu
  Size of the input vector.
- unsigned nw
  Size of the process noise vector.
- unsigned m
  Size of the measurement vector.
- unsigned nv
  Size of the measurement noise vector.

4.2.1 Detailed Description

template<typename T>class Kalman::Extended< T >

Generic Extended Kalman Filter template base class.

Usage

The Kalman filter is a set of mathematical equations that provides an efficient computational (recursive) solution of the least-squares method. The filter is very powerful in several aspects: it supports estimations of past, present, and even future states, and it can do so even when the precise nature of the modeled system is unknown.
This version of the Kalman filter is in fact a Variable-Dimension Extended Kalman Filter (VDEKF). It supports optimized algorithms, even in the presence of correlated process or measurement noise.
To use this template class, you must first inherit from it and implement some virtual functions. See the example page for more informations. Note that you can copy freely an `Extended-derived` class freely : this can be useful if you need to branch your filter based on some condition.

Notation

Assume a state vector x (to estimate) and a non-linear process function f (to model) that describes the evolution of this state through time, that is :

$$x_k = f\left(x_{k-1}, u_{k-1}, w_{k-1}\right)$$

where u is the (known) input vector fed to the process and w is the (unknown) process noise vector due to uncertainty and process modeling errors. Further suppose that the (known) process noise covariance matrix is :

$$Q = E\left(ww^T\right)$$

Now, let's assume a (known) measurement vector z, which depends on the current state x in the form of a non-linear function h (to model) :

$$z_k = h\left(x_k, v_k\right)$$

where v is the (unknown) measurement noise vector with a (known) covariance matrix :

$$R = E\left(vv^T\right)$$

Suppose that we have an estimate of the previous state $\hat{x}_{k-1}$, called a corrected state or an *a posteriori* state estimate. We can build a predicted state (also called an *a priori* state estimate) by using f :

$$\tilde{x}_k = f\left(\hat{x}_{k-1}, u_{k-1}, 0\right)$$

since the input is known and the process noise, unknown. With this predicted state, we can get a predicted measurement vector by using h :

$$\tilde{z}_k = h\left(\tilde{x}_k, 0\right)$$

since the measurement noise is unknown. To obtain a linear least-squares formulation, we need to linearize those two systems. Here are first-order Taylor series centered on $\tilde{x}_k$:

$$x_k \approx f\left(\hat{x}_{k-1}, u_{k-1}, 0\right) + \frac{\partial f}{\partial x}\left(\hat{x}_{k-1}, u_{k-1}, 0\right)\left(\Delta x\right) + \frac{\partial f}{\partial u}\left(\hat{x}_{k-1}, u_{k-1}, 0\right)\left(\Delta u\right) + \frac{\partial f}{\partial w}\left(\hat{x}_{k-1}, u_{k-1}, 0\right)\left(\Delta w\right)$$

$$= \tilde{x}_k + A\left(x_{k-1} - \hat{x}_{k-1}\right) + Ww_{k-1}$$

We can do the same for the other system :

$$z_k \approx h\left(\tilde{x}_k, 0\right) + \frac{\partial h}{\partial x}\left(\tilde{x}_k, 0\right)\left(\Delta x\right) + \frac{\partial h}{\partial v}\left(\tilde{x}_k, 0\right)\left(\Delta v\right)$$

$$= \tilde{z}_k + H\left(x_k - \tilde{x}_k\right) + Vv_k$$

The user of this class must derive from it, and implement all the functions corresponding to *A*, *W*, *Q*, f, *H*, *V*, *R* and h.

Template parameters

- `T` : Type of elements contained in matrices and vectors. Usually `float` or `double`.

Type requirements for T

- `T` must be **default constructible**.
- `T` must be **constructible from** `double`.
- `T` must be **assignable**.
- `T` must be **equality comparable**.
- `T` must be **serializable**.
- `T` must support **basic arithmetic operations**.

4.2.2 Member Function Documentation

4.2.2.1 template<typename T > void Kalman::Extended< T >::setSizeX (unsigned *n_*)

Sets the size of the state vector.

Parameters

*n_*	New state vector size. Must not be 0.

Warning

`init()` must always be called after this function and before any other non-dimensioning function.

Referenced by Kalman::Extended< T >::Extended().

4.2.2.2 template<typename T > void Kalman::Extended< T >::setSizeW (unsigned *nw_*)

Sets the size of the process noise vector.

Parameters

*nw_*	New process noise vector size.

Warning

`init()` must always be called after this function and before any other non-dimensioning function.

Referenced by Kalman::Extended< T >::Extended().

4.2.2.3 template<typename T > void Kalman::Extended< T >::init (Vector< T > & *x_*, Matrix< T > & *P_*)

Sets initial conditions for the Kalman Filter.

This function allows to set an initial state estimate vector and an initial error covariance matrix estimate. This must be called at least once, after all dimensioning functions and before any other function. However, it can also be called anytime to reset or modify *x* or *P*.

Parameters

*x_*	State vector estimate. Will be destroyed.
*P_*	Error covariance matrix estimate. Will be destroyed.

Warning

If `setSizeX()` or `setSizeW()` is called, then init() must be called again before any other non-dimensioning function.

4.2.2.4 template<typename T > void Kalman::Extended< T >::step (Vector< T > & *u_*, const Vector< T > & *z_*)

Makes one prediction-correction step.

This is the main `Extended` function. First, it resizes any matrix who needs it. Then, it proceeds to the time update phase, using the input vector `u_`. This means that the following virtual functions *should be* called : `makeCommonProcess()`, `makeA()`, `makeW()`, `makeQ()` and `makeProcess()`. At this stage, *x* contains a current predicted state instead of an old corrected state. If `z_` is empty, that is, if there are no measures in this step, there is no correction and the function stops there. Else, the measure update phase begins. This means that the following virtual functions *should be* called : `makeCommonMeasure()`, `makeHImpl()`, `makeVImpl()`, `makeRImpl()`, `makeMeasure()` and `makeDZ()`.After this phase, *x* contains the new corrected state.

Parameters

*u_*	Input vector. Will **not** be destroyed. Can be empty.
*z_*	Measurement vector. Will **not** be destroyed. Can be empty.

4.2.2.5 template<typename T > void Kalman::Extended< T >::timeUpdateStep (Vector< T > & *u_*)

Makes one prediction step.

This function first resizes any matrix who needs it. Then, it proceeds to the time update phase, using the input vector `u_`. This means that the following virtual functions *should be* called : `makeCommonProcess()`, `makeA()`, `makeW()`, `makeQ()` and `makeProcess()`. At this stage, *x* contains a current predicted state instead of an old corrected state.

Parameters

*u_*	Input vector. Will **not** be destroyed. Can be empty.

4.2.2.6 template<typename T > void Kalman::Extended< T >::measureUpdateStep (const Vector< T > & *z_*)

Makes one correction step.

First, this function resizes any matrix who needs it. If `z_` is empty, that is, if there are no measures in this step, there is no correction and the function stops there. Else, the measure update phase begins. This means that the following virtual functions *should be* called : `makeCommonMeasure()`, `makeHImpl()`, `makeVImpl()`, `makeRImpl()`, `makeMeasure()` and `makeDZ()`.After this phase, *x* contains the new corrected state.

Parameters

*z_*	Measurement vector. Will **not** be destroyed. Can be empty.

4.2.2.7 template<typename T > const Vector< T > & Kalman::Extended< T >::predict (Vector< T > & *u_*)

Returns the predicted state vector (*a priori* state estimate).

This function is used to predict a future state. First, it resizes any matrix who needs it. Then, it does a partial time update, in the sense that only *x* is updated, not P. This also means that only the following virtual functions *should be* called : `makeCommonProcess()` and `makeProcess()`.

Parameters

*u_*	Input vector. Will **not** be destroyed. Can be empty.

Warning

For better efficiency, the prediction is returned by reference.

4.2.2.8 template<typename T > const Vector< T > & Kalman::Extended< T >::simulate ( )

Returns the predicted measurement vector.

This function is used to predict a future measurement. First, it resizes any matrix who needs it. Then, it does a partial measure update, in the sense that only *z* is calculated : *x* and P are not updated. This also means that only the following virtual functions *should be* called : `makeCommonMeasure()` and `makeMeasure()`.

Warning

For better efficiency, the prediction is returned by reference.

4.2.2.9 template<typename T > const Matrix< T > & Kalman::Extended< T >::calculateP ( ) const

Returns the *a posteriori* error covariance estimate matrix.

Warning

This is not a simple return statement. Since P is not kept and updated in the filter (an alternate and more stable representation of P is used), calculations are involved to retrieve P. So, use this function wisely.
For better efficiency, P is returned by reference.

4.2.2.10 template<typename T > void Kalman::Extended< T >::makeCommonProcess ( ) [protected], [virtual]

Optional function used to precalculate common values for process.

If complex calculations are needed for more than one of `makeA()`, `makeW()`, `makeQ()` and `makeProcess()` functions, then this function can be used to store the results in temporary variables of the derived class.

Warning

This function must not modify any matrix of the base class.
This function must not be used to store permanent state. In other words, all calculations performed in this function should be temporary. This is because the `predict()` function will call this function but has no knowledge of how to undo it.

4.2.2.11 template<typename T > virtual void Kalman::Extended< T >::makeProcess ( ) [protected], [pure virtual]

Actual process $f(x, u, 0)$. Fills in new *x* by using old *x*.

This function **must** be overridden, since it is the core of the system process.

Warning

This function should have no side effects to class members (even members of derived classes) other than *x*. This is because this function is used by `predict()`, which does a calculation and then undoes it before returning the result.

Implemented in Kalman::Basic< T >.

4.2.2.12 template<typename T > void Kalman::Extended< T >::makeCommonMeasure ( ) [protected], [virtual]

Optional function used to precalculate common values for measurement.

If complex calculations are needed for more than one of `makeH()`, `makeV()`, `makeR()`, `makeMeasure()` and `makeDZ()` functions, then this function can be used to store the results in temporary variables of the derived class.

Warning

This function must not modify any matrix of the base class.
This function must not be used to store permanent state. In other words, all calculations performed in this function should be temporary. This is because the `simulate()` function will call this function but has no knowledge of how to undo it.

4.2.2.13 template<typename T > virtual void Kalman::Extended< T >::makeMeasure ( ) [protected], [pure virtual]

Actual measurement function $h(x, 0)$. Fills in *z*.

This function **must** be overridden, since it is the core of the measurement system. At the time this will be called, *x* contains the predicted state (*a priori* state estimate), which is the one that must be used with the measurement function.

Warning

This function should have no side effects to class members (even members of derived classes) other than *z*. This is because this function is used by `simulate()`, which does a calculation and then undoes it before returning the result.

Implemented in Kalman::Basic< T >.

4.2.2.14 template<typename T > void Kalman::Extended< T >::makeDZ ( ) [protected], [virtual]

Hook-up function to modify innovation vector.

This function should rarely be overridden ; this is more of a hack than anything else. In fact, this is used to perform adjustements on the result of substracting the predicted measurement vector to the real measurement vector. This is needed, for example, when measures include angles. It may be mandatory that the difference of the two angles be in a certain range, like $[-\pi, \pi]$.

4.2.3 Member Data Documentation

4.2.3.1 template<typename T > Vector<T> Kalman::Extended< T >::x [protected]

Corrected state vector.

This is an *n-sized* vector. Derived classes should modify it only through `makeProcess()`.

4.2.3.2 template<typename T > Vector<T> Kalman::Extended< T >::u [protected]

Input vector.

This is an *nu-sized* vector. Derived classes should never modify it.

4.2.3.3 template<typename T > Vector<T> Kalman::Extended< T >::z [protected]

Predicted measurement vector.

This is an *m-sized* vector. Derived classes should modify it only through `makeMeasure()`.

4.2.3.4 template<typename T > Vector<T> Kalman::Extended< T >::dz [protected]

Innovation vector.

This is an *m-sized* vector. Derived classes should modify it only through `makeDZ()`. The innovation vector is the difference between the real measurement vector and the predicted one.

4.2.3.5 template<typename T > Matrix<T> Kalman::Extended< T >::A [protected]

A jacobian matrix.

This is an *n* by *n* jacobian matrix of partial derivatives, defined as follow :

$$A_{[i,j]} = \frac{\partial f_{[i]}}{\partial x_{[j]}}$$

Derived classes should modify it only through `makeBaseA()` for the constant part and `makeA()` for the variable part.

4.2.3.6 template<typename T > Matrix<T> Kalman::Extended< T >::W [protected]

A jacobian matrix.

This is an *n* by *nw* jacobian matrix of partial derivatives, defined as follow :

$$W_{[i,j]} = \frac{\partial f_{[i]}}{\partial w_{[j]}}$$

Derived classes should modify it only through `makeBaseW()` for the constant part and `makeW()` for the variable part.

4.2.3.7 template<typename T > Matrix<T> Kalman::Extended< T >::Q [protected]

Process noise covariance matrix.

This is the *nw* by *nw* covariance matrix of *w*, that is :

$$Q = E\left(ww^T\right)$$

Derived classes should modify it only through `makeBaseQ()` for the constant part and `makeQ()` for the variable part.

4.2.3.8 template<typename T > Matrix<T> Kalman::Extended< T >::H [protected]

A jacobian matrix.

This is an *m* by *n* jacobian matrix of partial derivatives, defined as follow :

$$H_{[i,j]} = \frac{\partial h_{[i]}}{\partial x_{[j]}}$$

Derived classes should modify it only through `makeBaseH()` for the constant part and `makeH()` for the variable part.

4.2.3.9 template<typename T > Matrix<T> Kalman::Extended< T >::V [protected]

A jacobian matrix.

This is an *m* by *nv* jacobian matrix of partial derivatives, defined as follow :

$$V_{[i,j]} = \frac{\partial h_{[i]}}{\partial v_{[j]}}$$

Derived classes should modify it only through `makeBaseV()` for the constant part and `makeV()` for the variable part.

4.2.3.10 template<typename T > Matrix<T> Kalman::Extended< T >::R [protected]

Measurement noise covariance matrix.

This is the *nv* by *nv* covariance matrix of *v*, that is :

$$R = E\left(vv^T\right)$$

Derived classes should modify it only through `makeBaseR()` for the constant part and `makeR()` for the variable part.

The documentation for this class was generated from the following file:

- extended.h

4.3 Kalman::Matrix< T > Class Template Reference

Minimalist matrix template class.

`#include <matrix.h>`

Public Member Functions

Constructors.

- Matrix ()
  Default constructor. Creates an empty matrix.
- Matrix (unsigned m, unsigned n)
  Creates an `m` by `n` matrix of default instances of `T`.
- Matrix (unsigned m, unsigned n, const T &a)
  Creates an `m` by `n` matrix of copies of `a`.

Member access functions.

- T & operator() (unsigned i, unsigned j)
  Returns the element `(i,j)`.
- const T & operator() (unsigned i, unsigned j) const
  Returns the element `(i,j)`, `const` version.
- Matrix & operator= (const Matrix &m)
  Copy assignment operator. Performs a deep copy.
- unsigned Rows () const
  Returns m_, the number of rows of the matrix.
- unsigned Columns () const
  Returns n_, the number of columns of the matrix.
- void Resize (unsigned m, unsigned n)
  Resizes the matrix. Resulting matrix contents are undefined.

4.3.1 Detailed Description

template<typename T>class Kalman::Matrix< T >

Minimalist matrix template class.

This matrix class does not define any fancy linear algebra functions, nor even any operator between matrices. Its sole purpose is to well encapsulate an extensible array representing a matrix.

Template parameters

- `T` : Type of elements contained in the matrix. Usually `float` or `double`.

Type requirements for T

- `T` must be **default constructible**.
- `T` must be **assignable**.
- `T` must be **serializable**.

4.3.2 Constructor & Destructor Documentation

4.3.2.1 template<typename T > Kalman::Matrix< T >::Matrix (unsigned *m*, unsigned *n*) [inline]

Creates an `m` by `n` matrix of default instances of `T`.

Parameters

m	Number of rows in matrix. Can be 0.
n	Number of columns in matrix. Can be 0.

Note

If either `m` or `n` is 0, then both the number of rows and the number of columns will be set to 0.

4.3.2.2 template<typename T > Kalman::Matrix< T >::Matrix (unsigned *m*, unsigned *n*, const T & *a*) [inline]

Creates an `m` by `n` matrix of copies of `a`.

Parameters

m	Number of rows in matrix. Can be 0.
n	Number of columns in matrix. Can be 0.
a	Value to copy multiple times in the matrix.

Note

If either `m` or `n` is 0, then both the number of rows and the number of columns will be set to 0.

4.3.3 Member Function Documentation

4.3.3.1 template<typename T > T & Kalman::Matrix< T >::operator() (unsigned *i*, unsigned *j*) [inline]

Returns the element `(i,j)`.

Parameters

i	Row index of matrix element.
j	Column index of matrix element.

Returns

A reference to the element `(i,j)`.

4.3.3.2 template<typename T > const T & Kalman::Matrix< T >::operator() (unsigned *i*, unsigned *j*) const [inline]

Returns the element `(i,j)`, `const` version.

Parameters

i	Row index of matrix element.
j	Column index of matrix element.

Returns

A `const` reference to the element `(i,j)`.

4.3.3.3 template<typename T > Matrix< T > & Kalman::Matrix< T >::operator= (const Matrix< T > & *m*) [inline]

Copy assignment operator. Performs a deep copy.

Parameters

m	Matrix to copy.

Returns

A reference to the assigned matrix.

4.3.3.4 template<typename T > unsigned Kalman::Matrix< T >::Rows ( ) const [inline]

Returns *m_*, the number of rows of the matrix.

Returns

The number of rows in the matrix.

4.3.3.5 template<typename T > unsigned Kalman::Matrix< T >::Columns ( ) const [inline]

Returns *n_*, the number of columns of the matrix.

Returns

The number of columns in the matrix.

4.3.3.6 template<typename T > void Kalman::Matrix< T >::Resize (unsigned *m*, unsigned *n*) [inline]

Resizes the matrix. Resulting matrix contents are undefined.

Parameters

m	Number of rows in matrix. Can be 0.
n	Number of columns in matrix. Can be 0.

Note

If either `m` or `n` is 0, then both the number of rows and the number of columns will be set to 0.

The documentation for this class was generated from the following file:

- matrix.h

4.4 Kalman::Vector< T > Class Template Reference

Minimalist vector template class.

`#include <vector.h>`

Public Member Functions

- void Resize (unsigned)
  Resizes the vector. Resulting vector contents are undefined.
- void Swap (Vector &v)
  Constant-time swap function between two vectors.

Constructors.

- Vector ()
  Default constructor. Creates an empty vector.
- Vector (unsigned n)
  Creates a vector of `n` default instances of `T`.
- Vector (unsigned n, const T &a)
  Creates a vector of `n` copies of `a`.

Member access functions.

- T & operator() (unsigned i)
  Returns the `i'th` element.
- const T & operator() (unsigned i) const
  Returns the `i'th` element, `const` version.
- unsigned Size () const
  Returns the vector size.

4.4.1 Detailed Description

template<typename T>class Kalman::Vector< T >

Minimalist vector template class.

This vector class does not define any fancy linear algebra functions, nor even any operator between vectors. Its sole purpose is to well encapsulate an extensible array representing a vector.

Template parameters

- `T` : Type of elements contained in the vector. Usually `float` or `double`.

Type requirements for T

- `T` must be **default constructible**.
- `T` must be **assignable**.
- `T` must be **serializable**.

4.4.2 Constructor & Destructor Documentation

4.4.2.1 template<typename T > Kalman::Vector< T >::Vector (unsigned *n*) [inline]

Creates a vector of `n` default instances of `T`.

Parameters

n	Number of elements in vector. Can be 0.

4.4.2.2 template<typename T > Kalman::Vector< T >::Vector (unsigned *n*, const T & *a*) [inline]

Creates a vector of `n` copies of `a`.

Parameters

n	Number of elements in vector. Can be 0.
a	Value to copy multiple times in the vector.

4.4.3 Member Function Documentation

4.4.3.1 template<typename T > T & Kalman::Vector< T >::operator() (unsigned *i*) [inline]

Returns the `i'th` element.

Parameters

i	Index of element to retrieve from vector.

Returns

A reference to the `i'th` element.

4.4.3.2 template<typename T > const T & Kalman::Vector< T >::operator() (unsigned *i*) const [inline]

Returns the `i'th` element, `const` version.

Parameters

i	Index of element to retrieve from vector.

Returns

A `const` reference to the `i'th` element.

4.4.3.3 template<typename T > unsigned Kalman::Vector< T >::Size ( ) const [inline]

Returns the vector size.

Returns

The number of elements in the vector.

4.4.3.4 template<typename T > void Kalman::Vector< T >::Resize (unsigned *n*) [inline]

Resizes the vector. Resulting vector contents are undefined.

Parameters

n	The new size of the vector. Can be 0.

Note

Resizing to a smaller size does not free any memory. To do so, one can swap the vector to shrink with a temporary copy of itself : `Vector<T>(v).Swap(v)`.

4.4.3.5 template<typename T > void Kalman::Vector< T >::Swap (Vector< T > & *v*) [inline]

Constant-time swap function between two vectors.

This function is fast, since it exchanges pointers to underlying implementation without copying any element.

Parameters

v	Vector to swap.

The documentation for this class was generated from the following file:

- vector.h

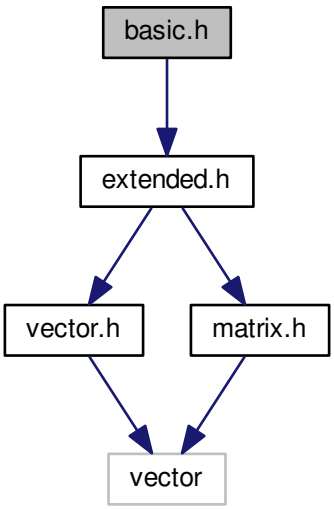
Chapter 5

File Documentation

5.1 basic.h File Reference

Contains the interface of the `Basic` template class.

`#include "extended.h"`
Include dependency graph for basic.h:



Classes

- class Kalman::Basic< T >

  Generic Linear Kalman Filter template base class.

5.1.1 Detailed Description

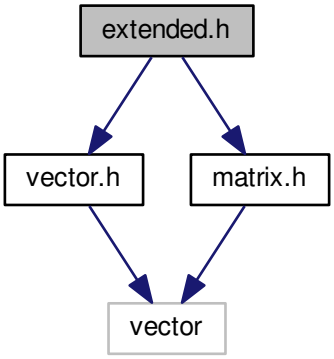
Contains the interface of the `Basic` template class.

5.2 extended.h File Reference

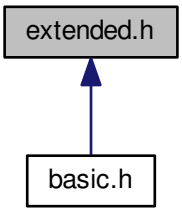
Contains the interface of the `Extended` base template class.

```
#include "vector.h"
#include "matrix.h"
```

Include dependency graph for extended.h:



This graph shows which files directly or indirectly include this file:



Classes

- class Kalman::Extended< T >

  Generic Extended Kalman Filter template base class.

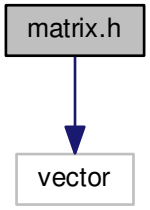
5.2.1 Detailed Description

Contains the interface of the `Extended` base template class.

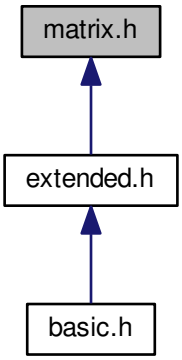
5.3 matrix.h File Reference

Contains the interface of the `Matrix` template class.

`#include <vector>`
Include dependency graph for matrix.h:



This graph shows which files directly or indirectly include this file:



Classes

- class Kalman::Matrix< T >

  Minimalist matrix template class.

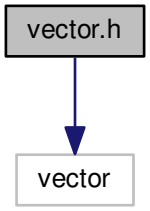
5.3.1 Detailed Description

Contains the interface of the `Matrix` template class.

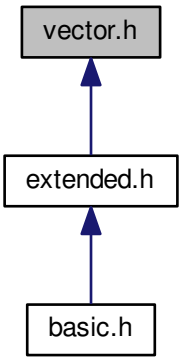
5.4 vector.h File Reference

Contains the interface of the `Vector` template class.

`#include <vector>`
Include dependency graph for vector.h:



This graph shows which files directly or indirectly include this file:



Classes

- class Kalman::Vector< T >

  Minimalist vector template class.

5.4.1 Detailed Description

Contains the interface of the `Vector` template class.

Index