Title: , Autonomous Helicopter Navigation System, State Estimation Design Document

*“A Project”*

Prepared by Date \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

,

Checked by Date \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

,

Approved by Date \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

, Student Manager 2010

Authorised for use by Date \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Dr Luis Mejias, Project Coordinator

**QUT Avionics**

Queensland University of Technology

CRCSS-EESE, GPO Box 2434

Gardens Point Campus

Brisbane, Australia, 4001.

Telephone (+61 7) 3864 1772

Facsimile (+61 7) 3864 1517

e-mail luis.mejias@qut.edu.au

web <http://code.google.com/p/ahns10/>

This document is Copyright 2010 by the QUT. The content of this document, except that information which is in the public domain, is the proprietary property of the QUT and shall not be disclosed or reproduced in part or in whole other than for the purpose for which it has been prepared without the express permission of the QUT

**Revision Record**

|  |  |  |  |
| --- | --- | --- | --- |
| Document Issue/Revision Status | **Description of Change** | **Date** | **Approved** |
|  | Initial Issue |  |  |

**Distribution List**

|  |  |  |  |
| --- | --- | --- | --- |
| **Name** | **Affiliation** | **Distribution Date** | **Approved** |
| Avionics Lab File Archive | QUT Avionics |  |  |

**Foreword**

Various parameters or states of the quadrotor platform need to be constantly measured for the platform to be controllable. This need is encapsulated in HLO-3 [RD/1] which specifies that a state estimation method needs to be designed and implemented. The purpose of this document is to present the design of the state estimation of the AHNS 2010 project. The state estimation will be documented in the following steps:

* Identifying the states which need to be measured for the quadrotor platform
* Choosing the sensors that will measure the states
* Designing a state estimation methodology for each state (which could include low pass filtering and or Kalman filtering)
* Implementing the design which is suitable for the flight computer located on the Overo Fire or on the GCS.

The state estimation design will also be divided into scenarios where the Vicon sensor isn’t and is available. Once the implementation is written and integration of the platform has proceeded then the state estimation can be tested. The outcomes of these tests will be documented in a test report which will signify that the relevant system requirements (and consequently HLO-3) have been met.

**Table of Contents**

Paragraph Page No.

[1 Introduction 8](#_Toc275631324)

[1.1 Scope 8](#_Toc275631325)

[1.2 Background 8](#_Toc275631326)

[2 Reference Documents 9](#_Toc275631327)

[2.1 QUT Avionics Documents 9](#_Toc275631328)

[2.2 Non-QUT Documents 9](#_Toc275631329)

[3 State estimation design (hover mode without Vicon) 10](#_Toc275631330)

[3.1 Euler rate estimation 10](#_Toc275631331)

[3.2 Euler angle estimation 12](#_Toc275631332)

[3.2.1 Standard Kalman filter theory 13](#_Toc275631333)

[3.2.2 Kalman filter estimation model for 14](#_Toc275631334)

[3.2.3 Kalman filter time update for 15](#_Toc275631335)

[3.2.4 Kalman filter measurement update for 16](#_Toc275631336)

[3.3 Euler angle estimation 19](#_Toc275631337)

[3.4 Alternative Euler angle estimation 19](#_Toc275631338)

[3.5 X,Y and Z accelerations 19](#_Toc275631339)

[3.6 X,Y and Z velocity 20](#_Toc275631340)

[3.7 X,Y position 20](#_Toc275631341)

[3.8 Z position 20](#_Toc275631342)

[4 State Estimation Design with Vicon 21](#_Toc275631343)

[4.1 Euler angle estimation 21](#_Toc275631344)

[4.2 Euler rate estimation 22](#_Toc275631345)

[4.3 X,Y and Z position 23](#_Toc275631346)

[4.4 X,Y and Z velocity 23](#_Toc275631347)

[4.5 X and Y target positions 23](#_Toc275631348)

[5 Conclusions 24](#_Toc275631349)

[6 Recommendations 25](#_Toc275631350)

[7 Appendices 26](#_Toc275631351)

[7.1 Appendix 1 – State estimation implementation 26](#_Toc275631352)

**List of Figures**

Figure Page No.

[Figure 1 - Kalman Filter time and measurement updates 14](#_Toc275205280)

**List of Tables**

Table Page No.

[Table 1- System requirements (relevant to state estimation) to met HLO-3 8](#_Toc275205275)

[Table 2 - Quadrotor Platform States and the Measurement Sensor (without Vicon) 10](#_Toc275205276)

[Table 3 - Gyroscope characteristics [RD/3] 11](#_Toc275205277)

[Table 4 - Accelerometer characteristics [RD/3] 20](#_Toc275205278)

[Table 5 - Quadrotor Platform States and the Measurement Sensor (with Vicon) 21](#_Toc275205279)

**Definitions**

|  |  |
| --- | --- |
| AHNS | Autonomous Helicopter Navigation System |
| QUT | Queensland University of Technology |
| HLO | High Level Objective |
| SR | System Requirement |
| GCS | Ground Control Station |
| IMU | Inertial Measurement Unit |
| Vicon | Motion capture system |
| EWMA | Exponentially Weighted Moving Average |
|  |  |
|  |  |
|  |  |

# Introduction

Design of the state estimation system is required to successfully meet HLO-3. The following system requirements need to be met to achieve this HLO [as specified in RD/2]:

Table - System requirements (relevant to state estimation) to met HLO-3

|  |  |
| --- | --- |
| **System Requirement** | **Definition** |
| SR-B-04 | The estimator shall provide Euler angle and rate estimation for the system at minimum rate of 50 Hz. |
| SR-B-05 | The estimator shall provide altitude estimation for the system at minimum rate of 50 Hz. |
| SR-B-06 | The estimator shall provide x and y estimation in an Earth fixed co-ordinate system at minimum rate of 50 Hz. |
| SR-D-05 | The airborne system shall receive and process measurement data from the state estimation and localisation sensors; supporting IMU, Camera, IR, Ultrasonic and Magnetic compass devices. |

Hence the state estimation system needs to be designed to ensure that the stated system requirements have been met. This will require a state estimation design for each individual state of the quadrotor platform. The design must also be implemented efficiently to guarantee that the estimated state will be calculated at a minimum rate of 50 Hz. Doing so will enable the quadrotor platform controller to issue high quality control commands.

## Scope

The quadrotor platform requires two sets state estimation designs: one without the usage of the Vicon system and one with the Vicon system. The state estimation design will thus be divided into two separate sections specifying the required design for each case.

## Background

State estimation is essential to keep the quadrotor platform controllable. A vast amount of research is dedicated to state estimation problems ranging from design implementations to difficulties with certain types of sensors. The estimation of quadrotor states has become increasingly popular over the years in the robotic community. This is due to the relatively simple design nature of a quadrotor compared with other complicated rotary craft e.g. helicopter.

# Reference Documents

## QUT Avionics Documents

|  |  |  |
| --- | --- | --- |
| RD/1 | AHNS-2010-SY-HL-001 | AHNS, High Level Objectives of |
| RD/2 | AHNS-2010-SY-SR-001 | AHNS, System Requirements of |

## Non-QUT Documents

|  |  |  |
| --- | --- | --- |
| RD/3 | Sensor Dynamics 6DOF IMU | Sensor Dynamics. 2009. 6 DOF INERTIAL MEASUREMENT UNIT WITH CONTINUOUS SELF DIAGNOSIS. Available: http://www.sensordynamics.cc/images/content/file/product\_linecards/1%205\_6DoF\_IMU\_v1%207.pdf (accessed October 17 2010). |
| RD/4 | Vicon DataStream SDK | Vicon. 2009. Vicon DataStream SDK 1.1.0 Developers Manual. Available: https://wiki.qut.edu.au/download/attachments/105028961/ Vicon+DataStream+SDK+Manual.pdf?version=1 (accessed October 17 2010). |

In the event of any conflict between this document and any RD referenced herein, such conflict shall be notified to Dr Luis Mejias.

In the following text, RD/x identifies referenced documents, where "x" denotes the actual document.

# State estimation design (hover mode without Vicon)

The following states of the quadrotor platform are required to be tracked while the platform is in hover mode without the Vicon sensor (refer to Table 2). The table below also includes what senor was used to measure the platform state. As this was only hover mode the key states which need to be tracked are the Euler angles (to gauge the platform’s attitude), the X,Y and Z velocities and the X,Y and Z displacement (to bound the position drift of the platform). The velocities and the displacement are to be calculated from the Blackfin Camera which will locate a target and attempt to keep the target within the middle of the frame. The estimation design for each state will need to be considered and developed to create a high quality state estimation system.

Table - Quadrotor Platform States and the Measurement Sensor (without Vicon)

|  |  |  |  |
| --- | --- | --- | --- |
| **State** | **Sensor** | **State** | **Sensor** |
| Roll rate | Rate gyro (IMU) | Z acceleration | Accelerometer (IMU) |
| Pitch rate | Rate gyro (IMU) | X velocity | Blackfin Camera |
| Yaw rate | Rate gyro (IMU) | Y velocity | Blackfin Camera |
| Roll | Rate gyro and accelerometers (IMU) | Z velocity | Altitude sensor |
| Pitch | Rate gyro and accelerometers (IMU) | X displacement | Blackfin Camera, IMU |
| Yaw | Rate gyro (IMU) and compass | Y displacement | Blackfin Camera, IMU |
| X acceleration | Accelerometer (IMU) | Z displacement | Altitude sensor, IMU |
| Y acceleration | Accelerometer (IMU) |  |  |

## Euler rate estimation

The 3 Euler rates are measured by 3 gyroscopes located in the Sensor dynamics 6DOF IMU. These gyroscopes are aligned to the body axis of the quadrotor platform and hence can directly measure the required Euler body rates. The table below lists the gyroscopes characteristics as specified in the datasheet [RD/3]. As can be seen in Table 3, the gyroscopes are not perfect sensors and contain some noise error. This measurement error can be mitigated by applying an exponentially weighted moving average (EWMA) filter. This filter is described by the following equation:

Where is the current mean value, is the previous mean value, is the current sensor reading and is the filter constant or smoothing factor. The parameter controls the degree of filtering or smoothing in the measurements and can range from 0 to 1. As approaches 1 the current mean value will rely more on the current sensor measurements. If is equal to 1 then no filtering will take place. Conversely, as approaches 0 the current mean value will rely more on the previous sensor measurements and will filter out any large changes in the current mean. Hence the EWMA filter performs exactly like a discrete low pass filter and can be tuned by the smoothing factor

Table - Gyroscope characteristics [RD/3]

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Parameter** | **MR1** | **MR2** | **Unit** | **Condition** |
| Measurement Range |  |  |  |  |
| Resolution |  |  |  | True 16 bit |
| Max RMS noise |  |  |  | Bandwidth:  MR1: 25Hz  MR2: 75Hz |
| Max zero rate bias |  | |  | Zero setting at |
| Max temperature drift of zero rate bias |  | |  | Over full temp range |
| Max sensitivity error |  | |  | Over full temp range |
| Max linearity error, versus best fit |  | |  | Over full temp range |

The EWMA filter can be shown to be equivalent to a discrete first order low pass filter. The Laplace transfer function of a first order low pass filter is equal to:

Where is equal to the time constant of the filter, is the filter signal and is the current measurement. The above equation can be converted to the time domain using inverse Laplace transform properties:

The differential in the above equation can be discretised using the following approximation:

Where is the time step between each measurement interval. Substituting this expression back into the time domain representation of the first order low pass filter yields:

Rearranging the above formula generates:

This equation can simplified by making a substitution for :

This creates the following formula when substituted into above formula

Which is equivalent to the EWMA filter. Thus the EWMA filter performs a first order low pass filter on the gyroscope data and eliminates some of the noise components generated from temperature and platform vibration. 3 separate EWMA filters will need be to implemented for each gyroscope to give a good estimation for each Euler rate . Corresponding smoothing parameters will also need to be discovered as well during integration testing.

## Euler angle estimation

The Euler angles and can be measured indirectly and directly from the IMU located on the platform. The 3 gyroscopes on the IMU measure the Euler rates which can be integrated at each time step to give an approximate or indirect measurement of the Euler angles . This approximation of the Euler angles will slowly drift over time due to temperature noise effects thus relying solely on gyroscopic data will produce poor quality estimates. The 3 accelerometers can also produce a coarse estimate of the Euler angles through trigonometric calculations. Whilst this is a direct measurement of the Euler angles it will also be of low quality due to the accelerometer noise.

Good quality estimates of the Euler angles can be achieved by fusing these two streams of data together. These two streams of data are complementary in that:

* The gyroscopic Euler angle measurement will drift over time but the coarse estimate of the Euler angles will not.
* The coarse estimate of the Euler angles produced by the accelerometer will contain far more noise than is present with the gyroscopes Euler angle estimate.

Thus fusing the two low quality estimates will bound the gyroscopic drift and generate good quality estimates of the Euler angles . The mechanism that performs this sensor fusion is the Kalman filter and can be implemented through a set of recursive mathematical equations. The theory of the Kalman filter will be presented first followed by the Kalman filter equations which will achieve Euler angle estimation of .

### Standard Kalman filter theory

A standard Kalman filter can be implemented via the following recursive equations. The equations are divided into two separate updates being the time update and the measurement update. The connection and control flow between these two updates can be seen in Figure 1. The equations required for the time update are:

Where is the state estimate at time , is the state transition matrix, is the one step ahead prediction of the state at time , is the control input matrix, is the control vector, is the state prediction covariance and is the process covariance matrix. The equations required for the measurement update are:

Where is the Kalman gain, is the observation matrix, is the measurement covariance matrix, is the error covariance matrix and is the Kalman filter estimate of the state. The Kalman filter needs to be initialised at to begin the estimation process where values of and need to be specified. is generally initialised to be a large number e.g.

The choice of and are not critical to the operation of the Kalman filter since the filter will eventually become independent of these values given a large enough time scale. However the values of and are extremely important to the operation of the Kalman filter and each matrix needs to be determined for each filter implementation.

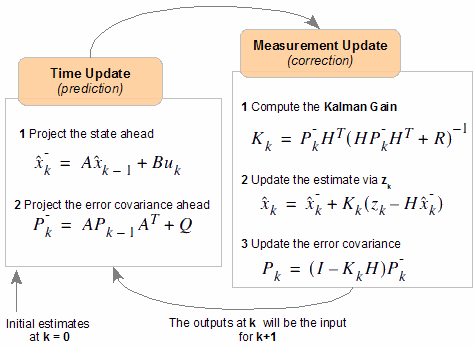


Figure - Kalman Filter time and measurement updates

### Kalman filter estimation model for

The first step in Kalman filter design is to determine how to estimate the model of the system. The initial Kalman filter design in this section will involve a separate Kalman filter for each Euler angle being estimated. The main reason for this is speed of computation since the more states a Kalman filter has the slower it will operate due to the matrix inversion required in the Kalman gain equation. The estimation model for are identical so only will be presented.

The following matrices and values need to be determined to develop the estimation model for : , , , , , and . The states that will be tracked for the Kalman filter when estimating are:

Where is equal to the Euler angle being tracked and is equal to the bias that has developed for the gyroscope measuring that Euler angle’s rate (in this case it is ). The state transition matrix is equal to:

Where is the time interval between each Kalman filter update step. The control input matrix is equal to:

The control vector is equal to the estimated Euler rate for the Euler angle being measured. In this case will be equal to which will be measured from the axis gyroscope after a EWMA filter has been applied. The process covariance matrix is equal to:

Where is the variance of the model estimate and is the variance of the model estimate. The observation matrix is equal to:

Finally the measurement covariance matrix is equal to:

Where is the variance of the measurement.

### Kalman filter time update for

The time update step in the Kalman filter for will be the same since the estimation models for are identical. Thus the one step ahead prediction of the states in the time update can be computed by substituting in the matrices and values defined in the estimation model:

Thus the equations which define the one step ahead prediction of and are equal to:

The time update is completed by computing state prediction covariance . Substituting values of the estimation model into the formula yields:

Let be equal to:

Substituting this value into the formula for creates:

Therefore the equations which define the state prediction covariance are equal to:

Note that with this implementation the term is generally ignored since will be extremely small assuming small time step increments.

### Kalman filter measurement update for

The measurement update for will be the same since the estimation model and time updates for are identical. The first equation to be executed in the measurement update is to compute the Kalman gain by the following equation:

The inversion term is typically labelled for innovation covariance. Substituting values for the estimation model into the formula for creates:

Thus the Kalman gain is equal to:

Therefore the equations which define the Kalman gain are equal to:

The second measurement update equation is to modify the error covariance described by:

Where is the identity matrix with dimension equal to the number of states or Substituting values for the estimation model into the above formula yields:

Therefore the equations which define the error covariance are equal to:

The final equation which completes the measurement update and hence the Kalman filter process is the state estimate . The state estimate is described by the following formula:

The innovation residual or error term in the above equation is denoted:

Where is the coarse Euler angle measurement calculated from the accelerometers. The coarse measurement for is equal to:

The coarse measurement for is equal to:

Thus can be represented as (demonstrating estimation of the Euler angle ):

Substituting the above expression and the estimation model values into the innovation residual equation yields:

Substituting this value and the estimation model values into the formula which describes the state estimate creates:

Thus the equations which describe the state estimate are equal to:

The value for is the Kalman filter estimate for the Euler angle at time interval . Thus both the data streams of the gyroscope and the accelerometers have been combined to produce a high quality estimate of the Euler angle state. The code that implements the state estimates for and can be seen in Appendix 1 (includes a header file kfb.h and a source file kfb.c).

## Euler angle estimation

A similar design approach is used for estimating the Euler angle as was designed for . The only difference between the designs is that the accelerometers cannot produce a coarse estimate for that is usable. Instead, another sensor needs to be employed that can directly measure the Euler angle. This sensor was chosen to be a compass thus the coarse measurement for in the Kalman filter measurement update is equal to:

All other states and equations for the time and measurement update are the identical for estimation to that which was designed for estimating The code that implements that state estimates for can be seen in Appendix 1 (includes a header file kfb.h and a source file kfb.c).

## Alternative Euler angle estimation

If time permits write about the extended Kalman filter which implements body to navigation conversion.

## X,Y and Z accelerations

The 3 accelerations in the axes are measured by 3 accelerometers located in the Sensor dynamics 6DOF IMU. These accelerometers are aligned to the body axis of the quadrotor platform and hence can directly measure the required body accelerations. The table below lists the accelerometers characteristics as specified in the datasheet [RD/3]. As can be seen in Table 4, the accelerometers are not perfect sensors and contain some noise error. This measurement error can be mitigated by applying an exponentially weighted moving average (EWMA) filter which has been previously discussed.

Table - Accelerometer characteristics [RD/3]

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Parameter** | **MR1** | **MR2** | **Unit** | **Condition** |
| Measurement Range |  |  |  |  |
| Resolution |  |  |  | True 16 bit  1 |
| Max RMS noise |  |  |  | Bandwidth:  MR1: 40Hz  MR2: 100Hz |
| Max zero g at RT |  | |  | Zero setting at |
| Max temperature drift of zero g bias |  | |  | Over full temp range |
| Max sensitivity error |  | |  | Over full temp range |
| Max linearity error, versus best fit |  | |  | Over full temp range |

## X,Y and Z velocity

The state estimation design of the X,Y and Z velocities will be considered in the camera design document.

## X,Y position

The state estimation design of the X and Y position will be considered in the camera design document.

## Z position

The displacement of the quadrotor platform is measured from the ultrasonic sensor located on the platform. The ultrasonic sensor is strapped down to the quadrotor platform thus the displacement will be a function of the ultrasonic measurement and the Euler angles . This can be described by the following equation:

The ultrasonic measurement is obtained by ADC thus a filter will need to be applied to the measurements taken from the ultrasonic sensor. The filter employed will be an EWMA filter.

# State Estimation Design with Vicon

The following states of the quadrotor platform are required to be tracked while the platform is in autonomous with the Vicon sensor (refer to Table 5). The table below also includes what senor was used to measure the platform state. When using the Vicon sensor almost all required states in autonomous mode can be tracked solely by this one sensor. The only additional state estimation design is for a camera target which is denoted in displacement co-ordinates of and y

Table - Quadrotor Platform States and the Measurement Sensor (with Vicon)

|  |  |  |  |
| --- | --- | --- | --- |
| **State** | **Sensor** | **State** | **Sensor** |
| Roll rate | Vicon | Y displacement | Vicon |
| Pitch rate | Vicon | Z displacement | Vicon |
| Yaw rate | Vicon | X velocity | Vicon |
| Roll | Vicon | Y velocity | Vicon |
| Pitch | Vicon | Z velocity | Vicon |
| Yaw | Vicon | X target | Blackfin Camera |
| X displacement | Vicon | Y target | Blackfin Camera |

## Euler angle estimation

The Vicon sensor can provide the Euler angles of the quadrotor platform by querying the Vicon Tracker program. The function which is used to do this is GetSegmentGlobalRotationEulerXYZ as documented in [RD/4]. It is important to note that this function returns the Euler angles in the XYZ co-ordinate system hence the Euler angles are expressed in a world frame attached to the Vicon system. The attitude controller requires the Euler angles to be in a ZXY co-ordinate frame which is commonly used in aeronautical applications. Thus a rotation system needs to be constructed to transform the Euler angles from the XYZ co-ordinate system to the required ZXY co-ordinate frame.

The rotation system involves converting the XYZ co-ordinate system to an intermediate co-ordinate frame ZYX where the ZXY Euler angles can be calculated. The reason for this intermediate conversion to ZYX is that the Vicon system provides a rotation matrix which is equal to the ZYX co-ordinate rotation. This rotation can be built from following 3 separate axis rotations:

* Rotate about by the yaw angle
* Rotate about by the theta angle
* Rotate about by the phi angle

The overall rotation can be represented as:

The function which returns this rotation is GetSegmentGlobalRotationMatrix as is documented in [RD/4]. From this resulting rotation matrix we can then use standard Direct Cosine Matrix (DCM) conversions to extract the Euler angles in the ZXY frame. These conversions are as follows:

Where is equal to the expression contained in the th row and th column in the derived rotation matrix .

## Euler rate estimation

The Euler rates can be calculated from the Vicon sensor by taking the time derivative of the Euler angles . As an example, the calculation for the Euler rate will be equal to:

Where is the estimate of the Euler rate, is the current measurement of the Euler angle, is the previous measurement of the Euler angle and is the time interval between measurements. Additional filtering may also be required for the Euler rates depending on the quality of the time derivative calculation i.e. some calculations may produce extremely large Euler rate magnitudes which will need to be filtered or reduced. An EWMA filter will be implemented and tuned depending on the quality of the Euler rate estimation data.

## X,Y and Z position

The Vicon sensor can directly provide the displacement of the quadrotor platform. The function which is used to do this is GetSegmentGlobalTranslation as documented in [RD/4]. This function returns the difference between the centre of mass position of the platform to the origin of the Vicon system. The centre of mass position of the platform is specified by the user in the Vicon tracker program. The origin of the Vicon system is also specified by the user by performing the Vicon tracker calibration routine.

## X,Y and Z velocity

The velocities can be calculated from the Vicon sensor by taking the time derivative of the position data. As an example, the calculation for the X velocity will be equal to:

Where is the estimate of the X velocity, is the current measurement of the displacement, is the previous measurement of the displacement and is the time interval between measurements. Additional filtering may be required for the X,Y and Z velocities depending on the quality of the time derivative calculation as mentioned in section 4.2. An EWMA filter will be implemented and tuned depending on the quality of the X,Y and Z velocity estimation data.

## X and Y target positions

Refer to Blackfin camera design document

# Conclusions

This document has specified the required state estimation design to track the quadrotor platform states. These state estimators have been designed to produce high quality outputs and be as efficient as possible (in particular the state estimation involving Kalman filters). This efficiency should ensure that state estimates can be supplied to the control loops within the minimum 50 Hz requirement. This allows the quadrotor to be controllable enabling the quadrotor platform to be flown by a human pilot or by autonomous commands.

# Recommendations

It is recommended that the state estimation designs specified in this document be implemented as shown. Test reports should be conducted on the state estimation design to verify that the states being produced are of high quality i.e. they track the desired state as intended. Testing should also check that the system requirement of a minimum 50 Hz update rate has been achieved. If any problems are found in the estimation design during testing then the state estimator should be redesigned and re-implemented. This redesign should be detailed within this document resulting in a new revision of AHNS-2010-SE-DD-001.

# Appendices

## Appendix 1 – State estimation implementation

Header file: kfb.h

/\*\*

\* **@file** kfb.h

\* **@author** Liam O'Sullivan

\*

\* $Author: liamosullivan $

\* $Date: 2010-08-28 11:26:11 +1000 (Sat, 28 Aug 2010) $

\* $Rev: 324 $

\* $Id: kf.h 324 2010-08-28 01:26:11Z liamosullivan $

\*

\* Queensland University of Technology

\*

\* **@section** DESCRIPTION

\* Kalman filter library (revision b)

\*

\*/

#ifndef KFB\_H

#define KFB\_H

#include <stdio.h>

#include <stdlib.h>

#include <unistd.h>

#include <math.h>

// Kalman filter constants for phi (Q1=0.001,Q2=0.003)

#define PHI\_ANGLE\_Q 0.057296

#define PHI\_GYRO\_Q 0.171887

#define PHI\_R 1.7

#define PHI\_DIRECTION 1.0

// Kalman filter constants for theta

#define THETA\_ANGLE\_Q 0.057296

#define THETA\_GYRO\_Q 0.171887

#define THETA\_R 1.7

#define THETA\_DIRECTION -1.0

// Kalman filter constants for psi

#define PSI\_ANGLE\_Q 0.057296

#define PSI\_GYRO\_Q 0.171887

#define PSI\_R 5.0

#define PSI\_DIRECTION -1.0

// Euler angle calibration cycles

#define CYCLES 1000

// LPF alpha constants for accelerometers

#define ACCX\_ALPHA 1.0

#define ACCY\_ALPHA 1.0

#define ACCZ\_ALPHA 1.0

// LPF alpha constants for gyro rates

#define RATEX\_ALPHA 1.0

#define RATEY\_ALPHA 1.0

#define RATEZ\_ALPHA 1.0

// LPF alpha constant for compass reading

#define COMPASS\_ALPHA 0.1

// LPF alpha constant for ultrasonic sensor

#define ALT\_ALPHA 0.07

// Data logger flag

#define DATA\_LOGGER 0

// struct definition for each axis

**typedef** struct \_axis **{**

// x state (tracking angle and gyro bias, 1x2)

double X**[**2**];**

// p covariance matrix (2x2)

double P**[**2**][**2**];**

// s term

double S**;**

// y measurement term

double Y**;**

// error term

double err**;**

// L gain matrix (1x2)

double L**[**2**];**

// Q process noise term (variance for angle and gyro bias, 1x2)

double Q**[**2**];**

// R measurement noise term

double R**;**

// direction of axis

double direction**;**

// angle offset

double offset**;**

**}** axis**;**

// function defintions

int attitudeFilterInitialiseB**(**double **\***accXr**,** double **\***accYr**,** double **\***accZr**);**

int attitudeFilterB**(**double **\***rateXr**,** double **\***rateYr**,** double **\***rateZr**,** double **\***accXr**,** double **\***accYr**,** double **\***accZr**,** double **\***rateXf**,** double **\***rateYf**,** double **\***rateZf**,** double **\***phif**,** double **\***thetaf**,** double **\***psif**,** double **\***compassZr**,** double dT**);**

int kFilterTimeUpdate**(**axis **\***axis\_t**,** double **\***gyroRate**,** double dT**);**

int kFilterMeasureUpdate **(**axis **\***axis\_t**);**

double coarsePitchAngle**(**double **\***accXr**,** double **\***accYr**,** double **\***accZr**);**

double coarseRollAngle**(**double **\***accXr**,** double **\***accYr**,** double **\***accZr**);**

int accLPF **(**double **\***accXr**,** double **\***accYr**,** double **\***accZr**,** double dT**);**

int rateLPF**(**double **\***rateXf**,** double **\***rateYf**,** double **\***rateZr**);**

int compassLPF**(**double **\***compass\_heading**);**

int altLPF**(**double **\***altitude**);**

int calibrateEulerAngles**(**double **\***phif**,** double **\***thetaf**,** double **\***psif**);**

int printkFilterData**(**double **\***rateXr**,** double **\***rateYr**,** double **\***rateZr**,** double **\***accXr**,** double **\***accYr**,** double **\***accZr**,** double dT**);**

#endif

Source file: kfb.c

/\*\*

\* **@file** kfb.c

\* **@author** Liam O'Sullivan

\*

\* $Author: liamosullivan $

\* $Date: 2010-08-28 11:26:11 +1000 (Sat, 28 Aug 2010) $

\* $Rev: 324 $

\* $Id: kfb.c 324 2010-08-28 01:26:11Z liamosullivan $

\*

\* Queensland University of Technology

\*

\* **@section** DESCRIPTION

\* Kalman filter library (revision b) which implements:

\* - Initialising the Kalman filter

\* - Performs the attitude filtering

\* - Low pass filtering for read IMU data

\*/

#include "kfb.h"

// phi axis struct

axis phi\_axis**;**

// theta axis struct

axis theta\_axis**;**

// psi axis struct

axis psi\_axis**;**

// accelerometer value storage (used for filtering)

double acc\_previous**[**3**];**

// filtered accelerometer readings (from the raw sensor data)

double accXf **=** 0.0**;**

double accYf **=** 0.0**;**

double accZf **=** 0.0**;**

// gyro value storage (used for filtering)

double rate\_previous**[**3**];**

double rate\_current**[**3**];**

// compass heading value storage (used for filtering)

double compass\_heading\_previous **=** 0.0**;**

double compass\_heading\_current **=** 0.0**;**

// altitude value storage (used for filtering)

double altitude\_previous **=** 0.0**;**

double altitude\_current **=** 0.0**;**

// euler angle storage (calculation of filtered rate)

double angle\_previous**[**3**];**

// calibration flag

int calib **=** 0**;**

// calibration cycle counter

int cyc\_count **=** 0**;**

// phi and theta angle storage for calibration

double phi\_sum **=** 0.0**;**

double theta\_sum **=** 0.0**;**

double psi\_angle **=** 0.0**;**

// function to initialise the values in the axis structures

int attitudeFilterInitialiseB**(**double **\***accXr**,** double **\***accYr**,** double **\***accZr**)**

**{**

/\* phi axis \*/

// initialise phi angle to coarse roll angle

phi\_axis**.**X**[**0**]** **=** coarseRollAngle**(**accXr**,**accYr**,**accZr**);**

// no initial bias

phi\_axis**.**X**[**1**]** **=** 0.0**;**

// initialise covariance matrix to 1

phi\_axis**.**P**[**0**][**0**]** **=** 1.0**;**

phi\_axis**.**P**[**0**][**1**]** **=** 1.0**;**

phi\_axis**.**P**[**1**][**0**]** **=** 1.0**;**

phi\_axis**.**P**[**1**][**1**]** **=** 1.0**;**

// initialise S term

phi\_axis**.**S **=** 0.0**;**

// initialise measurement term

phi\_axis**.**Y **=** phi\_axis**.**X**[**0**];**

// initialise correction term

phi\_axis**.**err **=** 0.0**;**

// initialise the kalman gain

phi\_axis**.**L**[**0**]** **=** 0.0**;**

phi\_axis**.**L**[**1**]** **=** 0.0**;**

// initialise the Q terms

phi\_axis**.**Q**[**0**]** **=** PHI\_ANGLE\_Q**;**

phi\_axis**.**Q**[**1**]** **=** PHI\_GYRO\_Q**;**

// initialise the R term

phi\_axis**.**R **=** PHI\_R**;**

// initialise the direction term

phi\_axis**.**direction **=** PHI\_DIRECTION**;**

// initialise the offset term

phi\_axis**.**offset **=** 0.0**;**

/\* theta axis \*/

// initialise theta angle to coarse pitch angle

theta\_axis**.**X**[**0**]** **=** coarsePitchAngle**(**accXr**,**accYr**,**accZr**);**

// no initial bias

theta\_axis**.**X**[**1**]** **=** 0.0**;**

// initialise covariance matrix to 1

theta\_axis**.**P**[**0**][**0**]** **=** 1.0**;**

theta\_axis**.**P**[**0**][**1**]** **=** 1.0**;**

theta\_axis**.**P**[**1**][**0**]** **=** 1.0**;**

theta\_axis**.**P**[**1**][**1**]** **=** 1.0**;**

// initialise S term

theta\_axis**.**S **=** 0.0**;**

// initialise measurement term

theta\_axis**.**Y **=** theta\_axis**.**X**[**0**];**

// initialise correction term

theta\_axis**.**err **=** 0.0**;**

// initialise the kalman gain

theta\_axis**.**L**[**0**]** **=** 0.0**;**

theta\_axis**.**L**[**1**]** **=** 0.0**;**

// initialise the Q terms

theta\_axis**.**Q**[**0**]** **=** THETA\_ANGLE\_Q**;**

theta\_axis**.**Q**[**1**]** **=** THETA\_GYRO\_Q**;**

// initialise the R term

theta\_axis**.**R **=** THETA\_R**;**

// initialise the direction term

theta\_axis**.**direction **=** THETA\_DIRECTION**;**

// initialise the offset term

theta\_axis**.**offset **=** 0.0**;**

/\* psi axis \*/

// initialise psi angle to compass

psi\_axis**.**X**[**0**]** **=** 0.0**;**

// no initial bias

psi\_axis**.**X**[**1**]** **=** 0.0**;**

// initialise covariance matrix to 1

psi\_axis**.**P**[**0**][**0**]** **=** 1.0**;**

psi\_axis**.**P**[**0**][**1**]** **=** 1.0**;**

psi\_axis**.**P**[**1**][**0**]** **=** 1.0**;**

psi\_axis**.**P**[**1**][**1**]** **=** 1.0**;**

// initialise S term

psi\_axis**.**S **=** 0.0**;**

// initialise measurement term

psi\_axis**.**Y **=** 0.0**;**

// initialise correction term

psi\_axis**.**err **=** 0.0**;**

// initialise the kalman gain

psi\_axis**.**L**[**0**]** **=** 0.0**;**

psi\_axis**.**L**[**1**]** **=** 0.0**;**

// initialise the Q terms

psi\_axis**.**Q**[**0**]** **=** PSI\_ANGLE\_Q**;**

psi\_axis**.**Q**[**1**]** **=** PSI\_GYRO\_Q**;**

// initialise the R term

psi\_axis**.**R **=** PSI\_R**;**

// initialise the direction term

psi\_axis**.**direction **=** PSI\_DIRECTION**;**

// initialise the offset term

psi\_axis**.**offset **=** 0.0**;**

/\* accelerometer previous values \*/

acc\_previous**[**0**]** **=** **\***accXr**;**

acc\_previous**[**1**]** **=** **\***accYr**;**

acc\_previous**[**2**]** **=** **\***accZr**;**

/\* gyro previous values \*/

rate\_previous**[**0**]** **=** 0.0**;**

rate\_previous**[**1**]** **=** 0.0**;**

rate\_previous**[**2**]** **=** 0.0**;**

/\* euler angle previous values \*/

angle\_previous**[**0**]** **=** phi\_axis**.**X**[**0**]\***M\_PI**/**180**;**

angle\_previous**[**1**]** **=** theta\_axis**.**X**[**0**]\***M\_PI**/**180**;**

// initialise the text file for data logging

**if(!**calib**)**

**{**

FILE **\***kfilterfd **=** fopen**(**"kfilter.ahnskfilter"**,**"a"**);**

**if(**kfilterfd**)**

**{**

// print header information

fprintf**(**kfilterfd**,**"### Kalman Filter data ###\n"**);**

fclose**(**kfilterfd**);**

**}**

**}**

**return** 1**;**

**}**

int attitudeFilterB**(**double **\***rateXr**,** double **\***rateYr**,** double **\***rateZr**,** double **\***accXr**,** double **\***accYr**,** double **\***accZr**,** double **\***rateXf**,** double **\***rateYf**,** double **\***rateZf**,** double **\***phif**,** double **\***thetaf**,** double **\***psif**,** double **\***compassZr**,** double dT**)**

**{**

// LPF the accelerometer values

accLPF**(**accXr**,**accYr**,**accZr**,**dT**);**

//\*rateZf = (\*rateZr)\*M\_PI/180;

//rateLPF(rateXf,rateYf,rateZf);

// Bound and LPF the compass value

compassLPF**(**compassZr**);**

// time update for phi axis

kFilterTimeUpdate**(&**phi\_axis**,**rateXf**,**dT**);**

// time update for theta axis

kFilterTimeUpdate**(&**theta\_axis**,**rateYf**,**dT**);**

// time update for psi axis

kFilterTimeUpdate**(&**psi\_axis**,**rateZf**,**dT**);**

// calculate the coarse angle for phi from the sensors

phi\_axis**.**Y **=** coarseRollAngle**(&**accXf**,&**accYf**,&**accZf**);**

// measurement update for phi axis

kFilterMeasureUpdate**(&**phi\_axis**);**

// calculate the coarse angle for theta from the sensors

theta\_axis**.**Y **=** coarsePitchAngle**(&**accXf**,&**accYf**,&**accZf**);**

// measurement update for phi axis

kFilterMeasureUpdate**(&**theta\_axis**);**

// LPF the compass values

compassLPF**(**compassZr**);**

// allocate measurement angle for the psi axis

psi\_axis**.**Y **=** **\***compassZr**;**

// measurement update for psi axis

kFilterMeasureUpdate**(&**psi\_axis**);**

// assign new phi angle (radians)

**\***phif **=** **(**phi\_axis**.**X**[**0**]-**phi\_axis**.**offset**)\***M\_PI**/**180**;**

// assign new theta angle (radians)

**\***thetaf **=** **(**theta\_axis**.**X**[**0**]-**theta\_axis**.**offset**)\***M\_PI**/**180**;**

// assign new psi angle (radians)

**\***psif **=** fmod**(((**psi\_axis**.**X**[**0**]-**psi\_axis**.**offset**)\***M\_PI**/**180**),**2**\***M\_PI**);**

// assign new phi rate

//\*rateXf = (\*phif - angle\_previous[0])/dT;

// assign new theta rate

//\*rateYf = (\*thetaf - angle\_previous[1])/dT;

// LPF the rate values (raw rate for the psi rate)

//\*rateZf = (\*rateZr)\*M\_PI/180;

// LPF the rate values (all raw)

**\***rateXf **=** **(\***rateXr**)\***M\_PI**/**180**;**

**\***rateYf **=** **(\***rateYr**)\***M\_PI**/**180**;**

**\***rateZf **=** **(\***rateZr**)\***M\_PI**/**180**;**

rateLPF**(**rateXf**,**rateYf**,**rateZf**);**

// update the previous angles

angle\_previous**[**0**]** **=** **\***phif**;**

angle\_previous**[**1**]** **=** **\***thetaf**;**

// check if calibration is taking place

**if(!**calib**)**

**{**

calibrateEulerAngles**(&**phi\_axis**.**X**[**0**],** **&**theta\_axis**.**X**[**0**],** psif**);**

**}**

// write all filtered data out to a file

**if(**DATA\_LOGGER**)**

**{**

printkFilterData**(**rateXr**,**rateYr**,**rateZr**,**accXr**,**accYr**,**accZr**,**dT**);**

**}**

**return** 1**;**

**}**

// Kalman filter time update for axis

int kFilterTimeUpdate**(**axis **\***axis\_t**,** double **\***gyroRate**,** double dT**)**

**{**

// compute the axis angle by unbiasing the gyro reading and integrating

axis\_t**->**X**[**0**]** **+=** **((\***gyroRate**\***axis\_t**->**direction**)** **-** axis\_t**->**X**[**1**])\***dT**;**

// calculate the covariance matrix and integrate (discrete)

axis\_t**->**P**[**0**][**0**]** **+=** **(**axis\_t**->**Q**[**0**]** **-** axis\_t**->**P**[**0**][**1**]** **-** axis\_t**->**P**[**1**][**0**])\***dT**;**

axis\_t**->**P**[**0**][**1**]** **+=** **-**axis\_t**->**P**[**1**][**1**]\***dT**;**

axis\_t**->**P**[**1**][**0**]** **+=** **-**axis\_t**->**P**[**1**][**1**]\***dT**;**

axis\_t**->**P**[**1**][**1**]** **+=** axis\_t**->**Q**[**1**]\***dT**;**

// end time update

**return** 1**;**

**}**

// Kalman filter measurement update for axis (assume coarse angle is calculated)

int kFilterMeasureUpdate **(**axis **\***axis\_t**)**

**{**

// compute the error term

axis\_t**->**S **=** axis\_t**->**P**[**0**][**0**]** **+** axis\_t**->**R**;**

// calculate the Kalman gain

axis\_t**->**L**[**0**]** **=** axis\_t**->**P**[**0**][**0**]** **/** axis\_t**->**S**;**

axis\_t**->**L**[**1**]** **=** axis\_t**->**P**[**1**][**0**]** **/** axis\_t**->**S**;**

// update the covariance matrix (careful of computation order)

axis\_t**->**P**[**1**][**0**]** **-=** axis\_t**->**L**[**1**]\***axis\_t**->**P**[**0**][**0**];**

axis\_t**->**P**[**1**][**1**]** **-=** axis\_t**->**L**[**1**]\***axis\_t**->**P**[**0**][**1**];**

axis\_t**->**P**[**0**][**0**]** **-=** axis\_t**->**L**[**0**]\***axis\_t**->**P**[**0**][**0**];**

axis\_t**->**P**[**0**][**1**]** **-=** axis\_t**->**L**[**0**]\***axis\_t**->**P**[**0**][**1**];**

// calculate the difference between prediction and measurement

axis\_t**->**err **=** axis\_t**->**Y **-** axis\_t**->**X**[**0**];**

// update the x states (angle and gyro bias)

axis\_t**->**X**[**0**]+=** axis\_t**->**err**\***axis\_t**->**L**[**0**];**

axis\_t**->**X**[**1**]+=** axis\_t**->**err**\***axis\_t**->**L**[**1**];**

// end measurement update

**return** 1**;**

**}**

// function to return the coarse pitch angle based on the accelerometer values (degrees)

double coarsePitchAngle**(**double **\***accXr**,** double **\***accYr**,** double **\***accZr**)**

**{**

**return** atan2**(\***accXr**,**sqrt**(\***accYr **\*** **\***accYr **+** **\***accZr **\*** **\***accZr**))\***180**/**M\_PI**;**

**}**

// function to return the coarse roll angle based on the accelerometer values (degrees)

double coarseRollAngle**(**double **\***accXr**,** double **\***accYr**,** double **\***accZr**)**

**{**

**return** atan2**(\***accYr**,**sqrt**(\***accXr **\*** **\***accXr **+** **\***accZr **\*** **\***accZr**))\***180**/**M\_PI**;**

**}**

// function to low pass filter the accelerometer values

int accLPF **(**double **\***accXr**,** double **\***accYr**,** double **\***accZr**,** double dT**)**

**{**

// LPF the X accelerometer value

accXf **=** acc\_previous**[**0**]\*(**1**-**ACCX\_ALPHA**)** **+** **(\***accXr **\*** ACCX\_ALPHA**);**

// LPF the Y accelerometer value

accYf **=** acc\_previous**[**1**]\*(**1**-**ACCY\_ALPHA**)** **+** **(\***accYr **\*** ACCY\_ALPHA**);**

// LPF the Z accelerometer value

accZf **=** acc\_previous**[**2**]\*(**1**-**ACCZ\_ALPHA**)** **+** **(\***accZr **\*** ACCZ\_ALPHA**);**

// store the filtered values

acc\_previous**[**0**]** **=** accXf**;**

acc\_previous**[**1**]** **=** accYf**;**

acc\_previous**[**2**]** **=** accZf**;**

**return** 1**;**

**}**

// function to low pass filter the rates values

int rateLPF**(**double **\***rateXf**,** double **\***rateYf**,** double **\***rateZf**)**

**{**

rate\_current**[**0**]** **=** **\***rateXf**;**

rate\_current**[**1**]** **=** **\***rateYf**;**

rate\_current**[**2**]** **=** **\***rateZf**;**

// LPF the phi rate

**\***rateXf **=** rate\_previous**[**0**]\*(**1**-**RATEX\_ALPHA**)** **+** **(**rate\_current**[**0**]** **\*** RATEX\_ALPHA**);**

// LPF the theta rate

**\***rateYf **=** rate\_previous**[**1**]\*(**1**-**RATEY\_ALPHA**)** **+** **(**rate\_current**[**1**]** **\*** RATEY\_ALPHA**);**

// LPF the psi rate

**\***rateZf **=** rate\_previous**[**2**]\*(**1**-**RATEZ\_ALPHA**)** **+** **(**rate\_current**[**2**]** **\*** RATEZ\_ALPHA**);**

// save the new rates

rate\_previous**[**0**]** **=** **\***rateXf**;**

rate\_previous**[**1**]** **=** **\***rateYf**;**

rate\_previous**[**2**]** **=** **\***rateZf**;**

**return** 1**;**

**}**

// function to calibrate the euler angles obtained from the IMU by computing an offset

int calibrateEulerAngles**(**double **\***phif**,** double **\***thetaf**,** double **\***psif**)**

**{**

// calculate offset sum

phi\_sum **+=** **\***phif**;**

theta\_sum **+=** **\***thetaf**;**

cyc\_count**++;**

// check if calibration has been completed

**if** **(**cyc\_count **>=** CYCLES**)**

**{**

// calibration complete, compute the mean for each axis

phi\_axis**.**offset **=** phi\_sum**/**CYCLES**;**

theta\_axis**.**offset **=** theta\_sum**/**CYCLES**;**

// change calibration flag

calib **=** 1**;**

printf**(**"Phi: %lf\n"**,**phi\_axis**.**offset**);**

printf**(**"Theta: %lf\n"**,**theta\_axis**.**offset**);**

**}**

**return** 1**;**

**}**

// function to print the kalman filter data

int printkFilterData**(**double **\***rateXr**,** double **\***rateYr**,** double **\***rateZr**,** double **\***accXr**,** double **\***accYr**,** double **\***accZr**,** double dT**)**

**{**

FILE **\***kfilterfd **=** fopen**(**"kfilter.ahnskfilter"**,**"a"**);**

**if(**kfilterfd**)**

**{**

// print the time stamp

fprintf**(**kfilterfd**,**"%lf,"**,**dT**);**

// print the raw gyroscope data

fprintf**(**kfilterfd**,**"%lf,%lf,%lf,"**,\***rateXr**,\***rateYr**,\***rateZr**);**

// print the raw accelerometer data

fprintf**(**kfilterfd**,**"%lf,%lf,%lf,"**,\***accXr**,\***accYr**,\***accZr**);**

// print phi axis data (angle,bias,measurement,offset)

fprintf**(**kfilterfd**,**"%lf,%lf,%lf,%lf,"**,**phi\_axis**.**X**[**0**],**phi\_axis**.**X**[**1**],**phi\_axis**.**Y**,**phi\_axis**.**offset**);**

// print theta axis data (angle,bias,measurement,offset)

fprintf**(**kfilterfd**,**"%lf,%lf,%lf,%lf,"**,**theta\_axis**.**X**[**0**],**theta\_axis**.**X**[**1**],**theta\_axis**.**Y**,**theta\_axis**.**offset**);**

// print psi axis data (angle,bias,measurement,offset)

fprintf**(**kfilterfd**,**"%lf,%lf,%lf,%lf\n"**,**psi\_axis**.**X**[**0**],**psi\_axis**.**X**[**1**],**psi\_axis**.**Y**,**psi\_axis**.**offset**);**

// all data saved, close the file

fclose**(**kfilterfd**);**

**}**

**return** 1**;**

**}**

int compassLPF**(**double **\***compass\_heading**)**

**{**

compass\_heading\_current **=** **\***compass\_heading**;**

**if(\***compass\_heading **>** 360.0**)**

**{**

// bound the compass heading by applying previous reading

**\***compass\_heading **=** compass\_heading\_previous**;**

**}** **else**

**{**

// reading ok, LPF the value

**\***compass\_heading **=** compass\_heading\_previous**\*(**1**-**COMPASS\_ALPHA**)** **+** **(\***compass\_heading **\*** COMPASS\_ALPHA**);**

compass\_heading\_previous **=** **\***compass\_heading**;**

**}**

**return** 1**;**

**}**

int altLPF**(**double **\***altitude**)**

**{**

altitude\_current **=** **\***altitude**;**

**\***altitude **=** altitude\_previous**\*(**1**-**ALT\_ALPHA**)** **+** **(\***altitude **\*** ALT\_ALPHA**);**

altitude\_previous **=** **\***altitude**;**

**return** 1**;**

**}**