FoilCart Extended Kalman Filter Code Base Documentation

Hadjiloizou, Loizos
loizosh@kth.se
KTH Royal Institute of Technology

May 30, 2023

Contents

Part I

Executables

Unit Tests

1.1 main-receiver.cpp

You can upload this code on a **Teensy 4.1** to test the functionality of reading data from a CAN bus. The Teensy 4.1 is connected to the CAN bus via the CAN Protocol Controller **CJMCU-2551** The CAN protocol controller is connected to the Teensy 4.1 pins (**CRX1: pin 23**, **TX1: pin 22**)

1.2 main-transmitter.cpp

You can upload this code on a **Teensy 4.1** to test the functionality of sending data to a CAN bus. The Teensy 4.1 is connected to the CAN bus via the CAN Protocol Controller **CJMCU-2551** The CAN protocol controller is connected to the Teensy 4.1 pins (**CRX1: pin 23**, **TX1: pin 22**)

1.3 main-transceiver.cpp

You can upload this code on a **Teensy 4.1** to test the functionality of reading and sending data simultaneously from/on a CAN bus. The Teensy 4.1 is connected to the CAN bus via the CAN Protocol Controller **CJMCU-2551** The CAN protocol controller is connected to the Teensy 4.1 pins (**CRX1: pin 23**, **TX1: pin 22**)

1.4 main-imu.cpp

You can upload this code on a **Teensy 4.1** to test the functionality of reading the IMU measurements from a **STIM300 IMU** module. The IMU module is connected via UART communication through a **Pmod RS485** module. the Pmod RS485 module is connected to the first UART serial channel on the Teensy 4.1 (**RX1: pin 0, TX1: pin 1**).

1.5 main-gnss.cpp

You can upload this code on a **Teensy 4.1** to test the functionality of reading the GPS measurements from a **SparkFun GPS-RTK2 Board - ZED-F9P** board. The GPS module is connected via an I2C serial communication, using the default Teensy4.1 pins (**SDA: pin 18, SCL: pin 19**)

Main File

2.1 main.cpp

This is the main file. This file provides us with the full functionality of the Extended Kalman filter. The functionalities currently supported in this version are:

- Read Senix Sensor data from the CAN bus
- Read control input signals for the main wing, rudder, and elevator from the CAN bus
- Write on the CAN bus the state estimated by the Extended Kalman Filter
- Read the GPS data from a SparkFun GPS-RTK2 board - ZED-F9P module
- Read the IMU data from a STIM300 IMU
- Perform state estimation using an Extended Kalman Filter

The pins used in this code are the same as the ones used in the unit tests.

Part II

Header Files

2.2 KalmanFilter.h

KalmanFilter.KalmanFilter()

Input: None

Description: This is the constructor for the KalmanFilter class. On creation of an object, it initializes the system matrices A, B, C, state vector x, and control input vector u.

Output: None

KalmanFilter.llt_inverse(Eigen::MatrixXf A)

Input: Matrix A

Description: This method computes the inverse of a matrix using LLT decomposition.

Output: Inverse of input matrix A

KalmanFilter.update(Eigen::MatrixXf u, Eigen::MatrixXf z)

Input:

control input vector u

measurement vector z

Description: This method performs the update step of the Extended Kalman Filter

Output: Estimated state vector

KalmanFilter.c2d_A(Eigen::MatrixXfAc)

Input: Continuous-time system update matrix A_c

Description: This method discretizes the continuous-time system update matrix A_c

Output: Discretize-time system update matrix A_d

KalmanFilter.c2d_B(Eigen::MatrixXf Bc)

Input: Continuous-time system update matrix B_c

Description: This method discretizes the continuous-time input matrix B

Output: Discretize-time input matrix B_d

2.3 jacobian.h

This header file contains the parameterized expressions for the state and control input continuous-time matrices for the linear system $\dot{x} = Ax + Bu$.

It is advised to avoid manually editing the contents of this file. Instead opt for updating it by running the script **generate_jacobian.py** which will update it using the dynamics model defined in **dynamic_model.py**.

2.4 can_comm.h

This header file contains the code needed to enable **Teensy 4.1** to read and write from/on the CAN bus.

init_can()

Input: None

Description: This method is responsible for initializing the parameters CAN bus interface, such as the baud rate and initializing the messages.

Output: None

write_can(std::vector¡float¿ state_CAN)

Input: Vector containing the estimated state by the EKF to be written on the CAN bus.

Description: Since the only piece of information we want to write on the CAN bus (at the time of writing this document), this method is then responsible for writing on the CAN bus the state estimated by the Extended Kalman Filter.

Output: None

read_can(...)

Input:

pointer to fore_alt

pointer to aft_alt

pointer to wing_angle

pointer to rudder_angle

pointer to elevator_angle

pointer to throttle

Description: This method reads the CAN bus to collect the information transmitter by the sonar Senix sensors, and the control inputs (main wing, rudder wing, and elevator wing). Then it updates the values of the input variables accordingly.

Output: None

Part III

Scripts

Introduction

This code base makes use of some python scripts to define the nonlinear dynamics model interpretable by CPP code. This is achieved by first defining the nonlinear dynamics model using casadi in python. Then the jacobian is computed parametrically. For Each element of the jacobian matrix a CPP function is automatically created. These CPP functions are then grouped to form the parametric continuous-time linear matrices A, B for the continuous-time linear system $\dot{x} = Ax + Bu$.

This approach is taken in order to allow one to directly define the model in a way that can be interpreted by the microcontroller instead of defining it using MATLAB for example. In addition, by generating a parametric form of the jacobian offline, this alleviates the Extended Kalman Filter from having to generate the jacobian online, instead it just evaluates it.

To test the functionality of this code, or to establish any changes in the model run the script **generate_jacobian.py** after you are done will all your changes.

2.5 dynamic_model.py

class DynamicModel

Creating an object of this class automatically defines the nonlinear dynamics model of the FoilCart.

The dynamics model used is defined in the appendix.

DynamicModel.compute_dynamics_jacobian()

Input: None

Description: This method computes and generates a string representation interpretable by CPP code for each element of the continuous-time parameterized linear matrices A, B.

Output: CPP interpretable string representations of the continuous-time parameterized linear matrices A, B.

DynamicModel.convert_alias(s) & DynamicModel.add_decimal_point(s)

Input: String (s)

Description: These methods take a a string representation of the output of the casadi jacobian method. This string is then handled by these two methods to be interpretable by CPP code.

Output: String (s)

2.6 generate_jacobian.py

This script makes use of the dynamics model defined in **dynamic_model.py** and is reponsible for generating the CPP header file **jacobian.h**.

Part IV

Appendix

Appendix: FoilCart Dynamics Model

Most of the parameters defined here (including the offsets for the lift and drag coefficients) have been estimated either by measurements of the actual physical dimensions or derived from experience. It is important to note that certain parameters have not been thoroughly tested (e.g., I_{xx}, I_{yy}, I_{zz}

Coordinate System

The coordinate system used in this work is:

- positive x semi-axis: Towards the front of the boat
- positive y semi-axis: Towards the right hand side of the boat
- positive z semi-axis: Downwards of the boat

State vector

- x : position in x-axis $[m]$
- y : position in y-axis $[m]$
- z : position in z-axis $[m]$
- ϕ : roll angle $[rad]$
- θ : pitch angle $[rad]$
- ψ : yaw angle $[rad]$
- v_x : velocity in x-axis $[m/s]$
- v_y : velocity in y-axis $[m/s]$
- v_z : velocity in z-axis $[m/s]$
- w_ϕ : roll rate $[rad/s]$
- w_θ : pitch rate $[rad/s]$
- w_ψ : yaw rate $[rad/s]$

Control Input vector

- N_1 : Thrust force by starboard motor $[N]$
- N_2 : Thrust force by port motor $[N]$
- α_w : Angle of the main wing $[rad]$
- α_r : Angle of the rudder $[rad]$
- α_e : Angle of the elevator $[rad]$

Parameters

Parameters			
Param	Value	Units	Description
g	9.81	$[m/s^2]$	Gravitational acceleration
ρ	996	$[kg/m^2]$	Density of medium (water)
m	150.85	$[kg]$	Total mass of the boat
I_{xx}	3.9	$[kgm^2]$	Moment of inertia for x-axis
I_{yy}	20.5	$[kgm^2]$	Moment of inertia for y-axis
I_{zz}	20	$[kgm^2]$	Moment of inertia for z-axis
A^w	0.2	$[m^2]$	Area of main wing ($\times 2$ NACA0015 200×500)
A^e	0.05	$[m^2]$	Area of elevator wing (NACA0015 100×500)
A^r	0.1	$[m^2]$	Area of rudder wing (NACA0015 200×500)
L_1	0.5	$[m]$	Distance in x-axis between CoG and Keel (estimated)
L_2	1.25	$[m]$	Distance in x-axis between keel and elevator
L_3	0.25	$[m]$	Distance in y-axis between CoG and a motor
L_4	1	$[m]$	Distance in z-axis between local origo and the fuselage
L_5	0.5	$[m]$	Distance in z-axis between CoG and the local origo (estimated)
L_6	0.25	$[m]$	Distance in x-axis between keel base and main wing center

Lift and drag coefficients

$$C_D^w = \frac{(\alpha_w+\theta)^2}{555.56} - \frac{(\alpha_w+\theta)}{333.33} + 0.0314 \qquad C_L^w = 0.076(\alpha_w + \theta) + 0.35$$

$$C_D^r = 0.0018\alpha_r^2 \qquad C_L^r = 0.076\alpha_r$$

$$C_D^e = \frac{(\alpha_e+\theta)^2}{555.56} - \frac{(\alpha_e+\theta)}{333.33} + 0.0314 \qquad C_L^e = 0.076(\alpha_e + \theta)$$

Rotation matrices

$$T_x = \begin{bmatrix} 1 & 0 & 0 \\ 0 & cos(\phi) & -sin(\phi) \\ 0 & sin(\phi) & cos(\phi) \end{bmatrix} \qquad T_y = \begin{bmatrix} cos(\theta) & 0 & sin(\theta) \\ 0 & 1 & 0 \\ -sin(\theta) & 0 & cos(\theta) \end{bmatrix} \qquad T_z = \begin{bmatrix} cos(\psi) & -sin(\psi) & 0 \\ sin(\psi) & cos(\psi) & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

$$T_{rot} = T_z T_y T_x$$

Forces and Torques

$$f_{strb} = N_1 \qquad f_D^w = \tfrac{1}{2}\rho C_D^w A^w v_x^2 \qquad f_D^r = \tfrac{1}{2}\rho C_D^r A^r v_x^2 \qquad f_D^e = \tfrac{1}{2}\rho C_D^e A^e v_x^2$$

$$f_{port} = N_2 \qquad f_L^w = \tfrac{1}{2}\rho C_L^w A^w v_x^2 \qquad f_L^r = \tfrac{1}{2}\rho C_L^r A^r v_x^2 \qquad f_L^e = \tfrac{1}{2}\rho C_L^e A^e v_x^2$$

$$\tau_{strb} = \begin{bmatrix} -(L_1 - L_6) \\ -L_3 \\ L_4 \end{bmatrix} \times \begin{bmatrix} f_{strb} cos(\alpha_w) \\ 0.0 \\ -f_{strb} sin(\alpha_2) \end{bmatrix} \qquad \tau_{port} = \begin{bmatrix} -(L_1 - L_6) \\ L_3 \\ L_4 \end{bmatrix} \times \begin{bmatrix} f_{port} cos(\alpha_w) \\ 0.0 \\ -f_{port} sin(\alpha_2) \end{bmatrix}$$

$$\tau_{wing} = \begin{bmatrix} -(L_1 - L_6) \\ 0.0 \\ L_4 \end{bmatrix} \times \begin{bmatrix} -f_D^w \\ 0.0 \\ -f_L^w \end{bmatrix} \qquad \tau_{elev} = \begin{bmatrix} -(L_1 L_2) \\ 0.0 \\ L_4 \end{bmatrix} \times \begin{bmatrix} -f_D^e \\ 0.0 \\ -f_L^e \end{bmatrix}$$

$$\tau_{keel} = \begin{bmatrix} -L_1 \\ 0.0 \\ L_4 \end{bmatrix} \times \begin{bmatrix} -f_D^r \\ f_L^r \\ 0.0 \end{bmatrix} \qquad \tau_{grav} = \text{Check code}$$

$$F_{tot,} = \begin{bmatrix} F_x \\ F_y \\ F_z \end{bmatrix} = \begin{bmatrix} (f_{strb} + f_{port}) cos(\alpha_w) - f_D^w - f_D^r - f_D^e) \\ f_L^r \\ mg - f_L^W - f_L^e - (f_{strb} + f_{port}) sin(\alpha_w) \end{bmatrix}$$

$$M_{tot} = \begin{bmatrix} M_x \\ M_y \\ M_z \end{bmatrix} = \begin{bmatrix} \tau_{strb} + tau_{port} + \tau_{wing} + \tau_{keel} + \tau_{elev} + \tau_{grav} \end{bmatrix}$$

Nonlinear state space dynamics model

$$h_1 = v_x \qquad h_4 = w_x \qquad h_7 = F_x/m \qquad h_{10} = M_x/m$$

$$h_2 = v_y \qquad h_5 = w_y \qquad h_8 = F_y/m \qquad h_{11} = M_y/m$$

$$h_3 = v_z \qquad h_6 = w_z \qquad h_9 = F_z/m \qquad h_{12} = M_z/m$$