University of Gothenburg

DIT168

Project: Industrial IT and Embedded Systems

Appropriate Algorithms and Adapting Available Concepts for IMU Regarding Acceleration and Yaw Calculation

Author:
Isabelle Törnqvist

Group 01

April 20th, 2018

Contents

1 Introduction

Using an Inertial Measurement Unit to measure the yaw, velocity and distance traveled of a miniature smartcar. This specific implementation is a part of the project dashFTAB[1].

2 Background

2.1 Data specific to Beaglebone Blue IMU

The component concerned in this document is the MPU9250 9-axis IMU, included in the BeagleBone Blue board by StrawsonDesign[2].

This implementation makes use of the roboticscape library. There are two modes to the roboticscape IMU: Random and DMP mode. According to documentation available by Strawson Design, the two modes can not be used simultaneously, as the DMP mode needs another configuration than the random mode.

Random mode: uses a random mode to get raw readings.

DMP mode: uses raw readings of accelerometer, magnetometer and gyroscope to calculate a filtered quaternion, useful when estimating orientation.

2.2 Components within the IMU

1. Accelerometer - measures proper acceleration in the rooms three dimensions, R^3

2. Gyroscope - measures angular velocity in the rooms three dimensions, R^3

3. Magnetometer - measures magnetism. Deemed unreliable due to the positioning of the IMU on the smartcar: magnetic interference will be high due to surrounding electronic currents.

4. Thermometer - will not be used in this project.

3 Yaw

3.1 Background

3.1.1 Specifics to project implementation

In the case of this implementation, the BeagleBone Blue board is positioned sideways on the miniature smartcar, as opposed to face-up. This will results in readings that indicate that the car is pitching, when it is in fact yawing. The interpretation of the readings in the axis must be shifted. Therefore, the z-axis becomes the x-axis, and the x-axis becomes the z-axis. For future readability within this documentation, the axises will be referred to in their shifted state, i.e. how they would be interpreted if the IMU was positioned face-up.

3.1.2 Accelerometer

Accelerometer will give readings in the x-axis, *roll*, y-axis, *pitch*, and z-axis, *yaw*. Assumption would be that x and y are in the plane, R^2 and z-axis is in the room, R^3. Readings of acceleration in the z-axis would - in a idle car - always measure around $9.8m/s^2$ due to the earth gravitational pull.

"Yaw" refers to the steering angle of a vehicle - the rotation around its z-axis. The first step is to find the yaw from the accelerometer readings. Assuming that the acceleration in the x-axis, a_x and the y-axis, a_y can be regarded to form the sides of a right angled triangle.

Firstly, the direction of the yaw must be accounted for: if the direction is negative, it can be assumed that the direction of the steering clockwise. Formula then used is a basic trigonometric function: The cosine of an angle is the adjacent side to the hypotenuse. To get yaw, arc cosine is used on the acceleration of the y-axis to the hypotenuse. The hypotenuse is the addition of acceleration in the x-axis squared, to the acceleration in the y-axis squared. This result will be in radians:

Algorithm 1 yaw in radius calculation for clockwise steering

$$yawRad = arccos\frac{a_y}{\sqrt{a_y^2 + a_x^2}}$$

If the direction of the yaw instead is counterclockwise, this algorithm may be used:

Algorithm 2 yaw in radius calculation for counterclockwise steering

$$yawRad = arctan2(a_x, a_y)$$

3.1.3 Gyroscope

The gyroscope readings given in this implementation needed to be stabilized.

3.2 Method

DMP mode will not be used. It is deemed unreliable due to the unreliability of the magnetometer as previously stated, in addition to the fact that the IMU is positioned sideways.

Instead, this project will make use of a version of a complementary filter, using sensor fusion of the accelerometer and gyroscope.

3.2.1 Sensor fusion

The idea of sensor fusion is to get cleaner data, by canceling out the errors and weaknesses of the two types of sensors. This implementation of a complementary filter high-passes readings from the fast and sensitive sensor, and low-passing data from the steady sensor, giving a result that is responsive in the short term while remaining accurate in the long term. The Low pass constant used in this project is $lpf = 0.02$ and the high pass one is defined as $hpf = 0.98$

The idea of these numbers are that they add up to a frequency of 1. The high pass is derived from

Algorithm 3 high pass constant

$$hpf = \frac{\tau}{\tau + \Delta t}$$

, where Δt is the sample rate of the gyroscope readings. Yaw is then derived from algorithm 4:

Algorithm 4 yaw in radius calculation using sensor fusion

$$\theta = hpf \times (gyro \times \Delta t) + lpf \times acceldata$$

where Δt is the sampling rate of the gyroscope readings, $accelData$ is the result of either of the two previously mentioned algorithms, and $gyro$ is the raw readings of the gyroscope in the z-axis.

4 Velocity

4.1 Background

The assumption for this project is that the acceleration in the x-axis and the acceleration in the y-axis can be interpreted as the sides of a right angled triangle. This implementation will be done in the plane R^2, as the car will be assumed to stay on a flat surface at all times, and not travel uphill nor downhill. This will simplify the implementation, as it doesn't account for the gravitational pull. The trade-off is as mentioned that the acceleration will be inaccurate if the car travels downhill or uphill. The combined acceleration will be derived from Pythagorean theorem regarding right angled triangles.

4.2 Method

As mentioned previously, the acceleration is the hypotenuse a_h in the assumed right angled triangle that is composed of the sides a_x and a_y - which are the acceleration in x-axis and y-axis. The acceleration is used to get the

Algorithm 5 acceleration using Pythagorean theorem

$$a_h = \sqrt{{a_x}^2 + {a_y}^2}$$

distance traveled by the smartcar, in Algorithm 6. Where s is the distance traveled within the sample period of Δt, and v_0 is the current velocity of vehicle.

The velocity of the smartcar is in turn given from the acceleration a and previous velocity, v_0

Algorithm 6 Distance traveled

$$s = vt + \frac{a\Delta t^2}{2}$$

Algorithm 7 Speed

$$v = a\Delta t + v_0$$

5 Result

The results of these implementations are quite unreliable due to the very nature of the sensors used. Both gyroscope and accelerometer give relative values and will be effected by irregularities in the environment, such as vibrations and rough ground.

The speed of the vehicle, as well as the distance traveled rely on the accelerometer alone. The sensor is sensitive to vibrations, and this will effect the given result of speed and distance.

References

[1] https://github.com/justasAtGU/dit168

[2] http://www.strawsondesign.com/