Localisation using a low-cost 6 axis accelerometer

Arthur Vie

May 3, 2021

1 Background

Through past experiments with model rockets as a hobby I have often desired an inexpensive way of tracking the position of a rocket in 3D space.

While commercial flight data recorders certainly exist they are often geared more toward simply measuring altitude and flight time. Some systems utilise GPS to track position, however the kind of commercial GPS modules available to the public only have an accuracy of 3-4m and a maximum update rate of approximately 10Hz.

I have considered that it may be possible to derive reasonably accurate position and orientation information from an onboard accelerometer, perhaps in combination with GPS using techniques of sensor fusion and real time kinematics.

2 Project Brief

The purpose of this project is to assess the feasibility of using the data from a cheap MEMS accelerometer IC like the LSM9DS1 to track the position and orientation of an object in 3D space.

2.1 LSM9DS1

The LSM9DS1 is an integrated circuit manufactured by ST Microelectronics that contains a three axis linear accelerometer (measuring acceleration in the x, y and z directions), a three axis "gyroscope" which measures angular rate / angular velocity in each of the three directions and additionally a three axis magnetometer.

explain MEMS, explain chip comms methods

2.2 Theory

In theory, with the accelerometer attached to the object we desire tracked, it should be possible, by taking a double integral of any of the linear acceleration values, to find how far the object has moved in each axial direction. Similarly, by taking the integral of the angular rate it should be possible to determine the objects orientation / attitude.

2.3 requirements

- Routines to communicate with the LSM9DS1 via SPI to gather accelerometer and gyroscope readings.
- an algorithm to perform integration of the accelerometer and gyroscope readings and accumulate the position and angle change over time.
- A routine to stream the position and angle data to a PC in real time, where it can either be displayed or logged to a file.
- perhaps a client software on the PC that will take data and plot it as a 3D trajectory.

3 Design

4 Hardware Documentation

Pin configuration:
LSM9DS1 Arduino Nano
GND -¿ GND
3V3 -¿ 3V3
CSAG -¿ GND
SCL -¿ D13
SDA -¿ D11
SDOAG -¿ D12

5 Setting up LSM9DS1

CTRL_REG1_G (0x10) bits 7, 6 and 5 set the output data rate. 110 sets an output data rate of 952 Hz.

Bits 4 and 3 select the full scale range of the gyroscope. 00: 245 dps; 01: 500 dps; 10: NA; 11: 2000 dps;

The first two bits 0 and 1 select the gyroscope bandwidth. For an output data rate of 952 Hz the maximum bandwidth, when both bits are set to 1, is 100 Hz.

CTRL_REG2_G (0x11) bits 0 and 1 control the source of the gyroscope data, either direct or through filters.

00 - direct data

CTRL_REG3_G (0x12) bit 6 HP_EN selects whether or not the high pass filter is enabled.

6 Testing

grabbing data from LSM9DS1 takes approximately 84 us using basic spi (sequential read would be faster);

total time to read and send all of the data takes worst case approximately 23 ms, best case about 16 ms, much too slow to be run inside integrator loop

Both the accelerometer and gyroscope are set to an output data rate of 952 Hz. The integration should be done at regular intervals timed using one of the internal timer counters.

given the amount of data that is generated and the fact that the serial interface is only running at 19200 baud it would not be possible to deliver data at the same rate as the integration.

There is a tradeoff. A fast integration rate would be required to track fine details of the motion. A lot of output data would be required to plot a detailed path of the motions of the object. These constraints are not possible at the same time.

A compromise would be to have a fast integration loop running on the microcontroller and to only send out certain samples of the tracked motion at a slower rate.

This could be achieved by driving the integration loop with a timed interrupt and with each interrupt have the UART shift out only one byte of the data. This would allow the sending of blocks of data to be interleaved with the integration loop such that the UART transmission never slows the loop down.

The data values that need to be transmitted are the angle in X, Y and Z and the position in X, Y and Z. The angles would be in degrees with maximum values of 360, therefore using 3 bytes each. The native word length on the ATMega328 is 16-bit. The position could be tracked in mm as a 16-bit signed integer, having maximum and minimum values of +32767 and -32768 respectively. Using up 6 bytes each. with comma delimiters and some sort of line ending delimiter like a semicolon this would yield a total of 33 bytes to transmit each sample of data. the data transmission rate would be 1 / 33 of the integrator loop rate.

A 500Hz integration loop should leave adequate time to send one byte of data with each loop.

It is clear form the size of the numbers generated, particularly by the double integration of the linear acceleration data, that there is a risk of the numbers overflowing in a very short time. In the case of value for angle, it will naturally wrap around every 360 degrees. Perhaps another variable could be used to track the number of complete rotations that have been made. In a similar way the position accumulator could perhaps be capped at a value of 1000 mm, at which point it would increment another variable counting the number of metres traveled, and then reset back to zero. This method could be used to prevent any single accumulated value from becoming vastly too large. Care would have to be taken to ensure the metres counter would be decremented were the accumulated millimeters to fall back bellow zero.

Tracking of the angle is working, however there is some amount of slow drift due to minor random offsets in the accumulated angular rate. The method is to take the difference of the angular rate reading over each time period. This cancels out any static offset error that there might be on the readings. These delta values are then summed to get an offset free value for the angular rate. There is still always some random noise on the readings though hopefully most of this should average out to zero over time. To get rid of the random static offset in the angular rate it may be possible to add a forcing term that would

pull the angular rate value back in toward zero when the module is not moving. The strength of this forcing term would have to be proportional to the offset it's self, i.e. if the angular rate settles to a constant value above or bellow zero, due to the module rotating at constant speed, this offset should not be nulled out. However if the angular rate settles very close to zero, i.e. a very very slow rotation speed (due to drift rather than actual motion) the forcing term should get strong enough to null out this offset. The forcing term should also be proportional to how fast the angular rate is changing. If the rate is changing fast then this is more likely to be due to actual motion, whereas if the rate is more or less static this is more likely to be an error offset. Therefore the forcing term should get stronger the smaller the average difference between successive rate readings.

There are several problems with the calculation of position from the acceleration values. The first problem is that the acceleration values measured are relative to the orientation of the accelerometer chip. This means that if the module never rotates the position measurements should be accurate, however once the orientation of the module changes, the position measurements will only be accurate in its reference frame rather than in world space.

An additional problem is how to account for the constant offset caused by the acceleration due to gravity. Initially I considered that taking a simple difference of the acceleration values would cancel out this offset. This works if the orientation / attitude of the module stays fixed, however if it rotates in any axis the acceleration due to gravity is spread into components in the other axes. As a result, when the difference is taken it appears that the module has accelerated even if it has never moved.

The solution to these problems is to use the angle information that we have already calculated to determine the orientation of the module in world space relative to its starting orientation. This would allow the direction of the gravitational vector to be calculated and corrected for (assuming the module starts from a known orientation), and would allow the acceleration values in the modules reference frame to be converted into world space.

7 Programming techniques used

- Conditional statements.
- Loops.
- Subroutines.
- Pointers.
- Modularity using separate program files and header files.
- interrupt service routines.
- Classes.
- Operator overloading.

8 Reflection

Before starting my degree at Leeds I worked for several years at the National Physical Laboratory where I was responsible for designing and implementing several embedded systems (here's an example of one of mine: https://www.npl.co.uk/instruments/at-k-radiation-thermometer), as a result the content of this module was mostly a revision of topics I had covered before. My experience in programming is mostly with pure C, so for this project I have attempted to push my knowledge and learn some more C++ concepts. I have been able to use classes and operator overloading for my project, both of which are powerful object-oriented techniques that I was only tangentially aware of before. I have also practiced and challenged my skills by attempting to run my project successfully on the actual hardware.

DISCLAIMER: I am well aware that for some of the routines and libraries I have written there already exist implementations that I could have used. In any case that I have written my own it is for fun, practice and for my own education.