ECSE426 - Microprocessor Systems

Winter 2013

Lab 3: 3D Tilt Angle and Tap Detection using Accelerometer

1. The General Requirements

In this lab, you will design a system that calculates tilt angles in 3 dimensions with 4^o accuracy. Angles are measured by the use of the gravity force (g). In your experiments you will use the F4-Discovery board. The input will come from the ST LIS302DL, the on-board digital accelerometer. This sensor is a package of three orthogonal one-dimensional acceleration sensors, which sense acceleration in three dimensions: X, Y, and Z. If the board is not accelerating under the influence of any external force (in this case it will be you moving it by hand), the sensor measures the projection of the gravitation force in all three dimensions.

The system should be calculating angles in real time, with the frequency of at least 100 times per second. However, this time, you are required to use the internal hardware timers of the STM32F4 processor instead of SysTick. Specifically, timer 3 is to be used. Finally, the ST LIS302DL sensor also allows for tap detection (single tap and double tap). You are required to configure the sensor to detect a single tap and turn on/off the LEDs at each tap (one tap turns LEDs on, the subsequent one turns them off and so on).

2. The Devil is in the Details

A) Accelerometer Calibration and Filtering

Even if we assume no external noise sources are present, sensor readings are never perfect. Aside from their inaccuracies introduced by design limitations and manufacturing technology, other sources of error are also present causing sensor readings to deviate from what is expected. There are methods to minimize and correct deviations (i.e. offset and sensitivity change) by recalibrating the sensors. In this experiment, you are required to apply **off-line calibration** to the accelerometer sensor. By off-line calibration, we mean that the process of obtaining the calibration parameters is done once and does not needed to be executed each time your board runs. During standard sensor's operation, you apply the corrective equations (determined during the calibration) to current sensor measurements. In contrast, on-line calibration requires the

repetition of the calibration process each time the board is powered on. There are many techniques to calibrate accelerometers; we present here one simple technique. Another more formal way is found in an application note from ST [1].

Generally, accelerometer calibration is done by putting your board in an orientation where the ideal accelerometer reading (in this case g-force) is known. Then you measure and read the actual sensed values and apply or set of equations to find parameters that map the measurement to the ideal value. We start by aligning the sensor plane to be orthogonal to the horizontal plane. This should ideally generate a value of 1g (as the unit is defined) on the sensed axis and 0g on the others (i.e. putting your Discovery board at horizontal level on the table will make Z-axis orthogonal to the horizontal plane and thus a value of 1 g is expected on Z, 0 g on X and Y). If you flip the board, a value of -1g is recorded for the Z axis. To calibrate you need to obtain a set of values for all axes. Then we need to find the following parameters:

1. Offset (OFFX, OFFY) [LSB]
2. Sensitivity (SENSX, SENSY) [LSB/mg]. "This parameter defines the value of 1 LSB with respect to mg in the digital representation" [1]

To compute each, you need to take readings in the +1g and -1g position for each axis. You will have two values which would ideally be centered about 0 but likely are not. Your offset for the axis is this error. Applying this offset, you will now have two values centered about 0 but with non-unity magnitude (i.e. -0.98, +0.98). The scalar multiplier used to correct for this will be your sensitivity for the axis (i.e. 1.0/0.98). You should store these calibration values and apply them for each sample.

Once you have calibrated the sensor values, you can then filter them using the **moving average filter**. Again, you might need to investigate the width of the window, which would produce better filtered traces.

B) Tilt detection

Now that you have calibrated filtered values, you are required to measure the tilt angles. Tilt angles are a way to describe how an object is oriented in 3D space relative to a global XYZ coordinate system. The global XY plane is horizontal to earth and the global Z is perpendicular to it. Simply as we humans perceive it in our daily lives. When an object is moved in 3D space, it might change its angle(s) respective to this global coordinate system. Those three angles are called Roll (alpha) , Pitch (beta) and Yaw (gamma). We are interested in the former two:

- Roll (Alpha): the angle between the X-axis and the horizontal plane
- Pitch (Beta): the angle between the Y-axis and the horizontal plane

Refer to Figure 1 which portrays the angles and the equations to measure them using the calibrated acceleration sensor values.

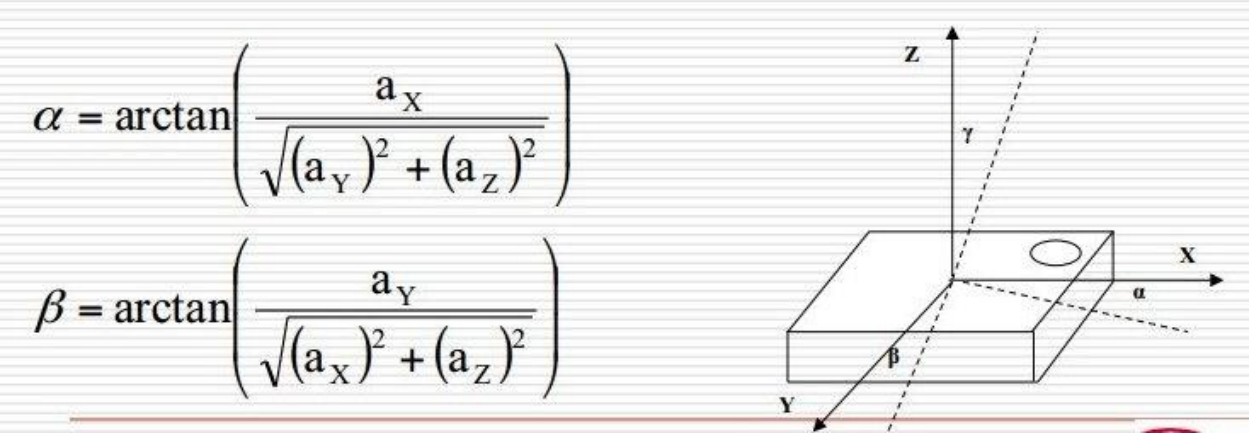

Figure 1: 3D tilt angles and their measurement equations

For detailed discussion and introduction to tilt angles, students are advised to refer to ST application notes in [1]. The document also provides the procedure of measuring tilt angles. The document also presents many of the related terminology, which you will need to know and understand when configuring the sensor. It also explains in details pitch, roll and yaw angles which you will often hear and use when using accelerometers for tilt detection and 3D orientation. Please note that among three such angles (roll, pitch and yaw/heading), the last one is not detectable, as the gravity force does not depend on the yaw. Also note that the application notes you find on the Internet might not be for exactly the same device, they are general in nature to apply for different devices.

C) Timers

In lab 2, you were introduced briefly to the SysTick timer which was used to generate a regular tick to provide a consistent sampling rate with minimal jitter. SysTick is a great tool but its uses are limited. The timers provided by ST come in many different flavors, some with rather complex functionality intended for precise control applications and featuring their own dedicated IO pins. Others, such as the general purpose TIM3 which you will be using, are much less complicated to operate for simple applications.

There are two common ways to implement an interval timer. One is to use an up-counting timer where an event (interrupt) is generated when the counter hits a certain value and resets. Alternatively, a down-counter can be used which generates an event upon hitting 0 (and reloading some initial value). At the lower level, each design has some advantages but when using TIM3, the latter makes a lot more sense. For other timers including those from other vendors, this may not always be the case. In addition to configuring the reload value, you may choose a prescaler value, which is an integer clock divider that you can use to reduce the tick rate of the timer down from the APB1 clock that drives it. ST's timers generate a number of events for different conditions and it is up to you to determine which events to enable. The chosen events will drive the TIM3 interrupt vector (IRQ).

D) Tap detection

With this accelerometer, you will trigger an interrupt when the station is tapped, and signal that by turning on/off LED lights, concurrently to the timer interrupt-driven routine which reads the current angles of the board. That is, you will have two sources of interrupts, one for the timer, which reads the values at 100Hz, and another for the tap event, which turn on/off the LED at each subsequent tap. The Application Note [3] from ST illustrates how to enable tap detection in the LIS302DL by setting up and configuring certain control registers. Then you need to configure this external interrupt and connect it to your interrupt logic and enable its detection in your software. The EXTI project in the discovery board examples is a good starting point on how to configure an external interrupt. The MEMS project is a good way to start with the accelerometer. As you can see, interrupts allow for the concurrent execution of different functions in your applications.

E) Interrupts

Interrupts are designed to do exactly as the name suggests -- that is interrupt the normal program flow in the processor to handle some event. When an interrupt is signaled, the Nested Vector Interrupt Controller (NVIC) will save program state to the stack and begin execution of a handler which is indicated in the vector table located at the beginning of flash memory. There are a total of 255 vectors (interrupt sources) available, though fewer than 100 are implemented according to Table 30 of the STM32F4 Reference Manual. Each vector corresponds to a position in the vector table which contains the address of the handler to be executed. The 'nested' part of the NVIC refers to the fact that multiple interrupts can be serviced based on their priority. That is, a higher priority interrupt can preempt a lower priority interrupt. When an interrupt returns, any outstanding lower priority interrupts will run.

Interrupt priority is an important point to consider when configuring interrupts. Priority indicates how important it is that the interrupt be serviced immediately. Events that drive a more jitter-tolerant process may be set at a lower priority than something which may require immediate response. Because interrupts operate at higher priority than regular code, it is extremely

important to minimize the amount of time spent in your interrupt routines. There should **under no circumstances** be any external bus transactions (ie. accelerometer read/write) in your interrupt handlers. Further, you should be careful with your use of shared resources (memory, peripherals) as it is often difficult to predict what operations you might be interrupting. It is very easy to end up with undefined behavior this way when you attempt to use a resource which may already be in use.

To configure an interrupt, you must configure both the interrupt source (peripheral) and the NVIC to accept the interrupt. Most vectors have configurable priority and are by default disabled. Check the NVIC-related functions in the ST Peripheral Library's *misc.h*. There is, as usual, an initialization structure which defines the settings for a given interrupt vector. Keep in mind when assigning a numerical priority value that **on ARM systems, lower number means higher priority**. Check examples from the peripheral library.

A special and particularly simple group of interrupts is the external interrupt (EXTI) which is a configurable edge-triggered interrupt on GPIO pins. For your purposes, it can be used to detect button presses or accelerometer-generated interrupts for tap detection. There are 16 EXTI channels (EXTI0-EXTI15) which each correspond to a position in any given GPIO port. For example, PA2 and PB2 both map to EXTI2 and thus only one of them may be used as an interrupt source. The higher pins are mapped in groups so that EXTI-EXTI9 share a single interrupt vector and similarly for EXTI10-EXTI15.

3. Software library support

There is a set of functions acting as drivers for SPI (Serial Peripheral Interface, the synchronous serial protocol used in this lab), LIS302DL, Timers and all other peripherals that are available with your board. Those driver files are usually provided by ST in two files per peripheral, a C file and an associated header file. The C file has the implementation of the function whereas the header file has lists all the functions available to you as well as any configuration parameters you might need to pass by to those drivers as you have already seen in lab 2 while working with the ADC. The driver file for the accelerometer is called "stm32f4_discovery_lis302dl.c", "stm32f4xx_tim.c" is the driver file for the timers and "stm32f4xx_exti.c" is the drivers file for configuring external interrupts.

A good starting point is to refer to the Discovery Board Peripheral examples and start the MEMS project. It will show you how these drivers are to be used as well as the steps needed to adjust those drivers for your needs. The EXTI project is a good starting point on how to configure an external interrupt. The TIM_Base is a starting point for timers.

You can identify the functions and the data structure for initiation of the accelerometer. Based on the specification of the problem, you need to provide the values for the fields of that data structure to select values such as: data rate, enable axes, define the scale of the accelerometer and the update mode of the device, as well as the potential filtering of the data. Here we list a short

example of how to use the accelerometer driver. Some of the driver functions which you will find useful are:

1. `void LIS302DL_Init(LIS302DL_InitTypeDef *LIS302DL_InitStruct);`
2. `void LIS302DL_ReadACC(int32_t* out);`
3. `void LIS302DL_Write(uint8_t* pBuffer, uint8_t WriteAddr, uint16_t NumByteToWrite);`
4. `void LIS302DL_Read(uint8_t* pBuffer, uint8_t ReadAddr, uint16_t NumByteToRead);`

The first function is used to initialize and configure the accelerometer, similar to what you did with the ADC, you have to pass an initialization structure which takes the form of:

```
typedef struct
{
  uint8_t Power_Mode;                     /* Power-down/Active Mode */
  uint8_t Output_DataRate;                /* OUT data rate 100 Hz / 400 Hz */
  uint8_t Axes_Enable;                    /* Axes enable */
  uint8_t Full_Scale;                     /* Full scale */
  uint8_t Self_Test;                      /* Self test */
}LIS302DL_InitTypeDef;
```

To configure the accelerometer with low power mode, output data rate of 100Hz and enable all sensing axes and the range of measurements you might write something like:

```
LIS302DL_InitTypeDef  LIS302DL_InitStruct; // Declare structure

/* Set configuration of LIS302DL */
LIS302DL_InitStruct.Power_Mode = LIS302DL_LOWPOWERMODE_ACTIVE;
LIS302DL_InitStruct.Output_DataRate = LIS302DL_DATARATE_100;
LIS302DL_InitStruct.Axes_Enable = LIS302DL_X_ENABLE | LIS302DL_Y_ENABLE |
                                  LIS302DL_Z_ENABLE;
LIS302DL_InitStruct.Full_Scale = LIS302DL_FULLSCALE_2_3;
LIS302DL_InitStruct.Self_Test = LIS302DL_SELFTEST_NORMAL;

LIS302DL_Init(&LIS302DL_InitStruct); // Actually initialize the accelerometer
```

The “void LIS302DL_ReadACC” function is used to read all measured accelerometer values from all enables sensors at once. The “void LIS302DL_Read” and “void LIS302DL_Write” functions are used to read and write individual internal accelerometers registers by specifying a buffer (to read from/ store into), an address to that register and the number of values to be read. For example, to enable interrupt generation on click/double click on Z axis you might write:

```
ctrl = 0x70;
LIS302DL_Write(&ctrl, LIS302DL_CLICK_CFG_REG_ADDR, 1);
```

All register addresses and configuration options can be found in the header file of the driver. You should always consult the datasheet for more details.

As communication to the accelerometer needs to be done via SPI, proper initialization of SPI and sending/receiving of the packets needs to happen first, followed by the actual setting of the registers in LIS302DL via SPI and reading of the sampled data. Luckily for you, all SPI functionality is being abstracted by the accelerometer driver functions. Later on, we will delve into the details of SPI and you will eventually write your own SPI functions.

4. Calculating and observing calculated data in real time

Since the sensor measurements change over time, you are expected to demonstrate the measurement results in real time, as per above specification, using the SWD debug interface. In addition to using the SWD debug interface, you should be ready to provide the evidence that your solution meets the specification.

5. Testing

During demo time, the T.A.s will be bringing some objects with known oblique angles, like a triangular wooden piece with 30^o, 60^o and 90^o degrees. Your application should measure these values accurately to within 4^o degrees tolerance. That is, for the 30^o angle, your values should be within 26^o to 34^o degrees.

6. Demonstration

You will need to demonstrate that your program is correct, and explain in detail how you guarantee the desired precision, and how you tested your solution. You will be expected to demonstrate a functional program and its operation performs in all situations. The final demonstration will be on **February. 18th.**

7. Required Reading

[1] ST Application note AN3182, Tilt measurement using a low-g 3-axis accelerometer http://www.st.com/internet/com/TECHNICAL_RESOURCES/TECHNICAL_LITERATURE/A

PPLICATION_NOTE/CD00268887.pdf

[2] LIS302DL Datasheet

http://www.st.com/internet/com/TECHNICAL_RESOURCES/TECHNICAL_LITERATURE/DATASHEET/CD00135460.pdf

[3] ST Application note AN2579, LIS302DL 3-axis digital MEMS accelerometer translates finger taps into actions

http://www.st.com/internet/com/TECHNICAL_RESOURCES/TECHNICAL_LITERATURE/APPLICATION_NOTE/CD00163557.pdf