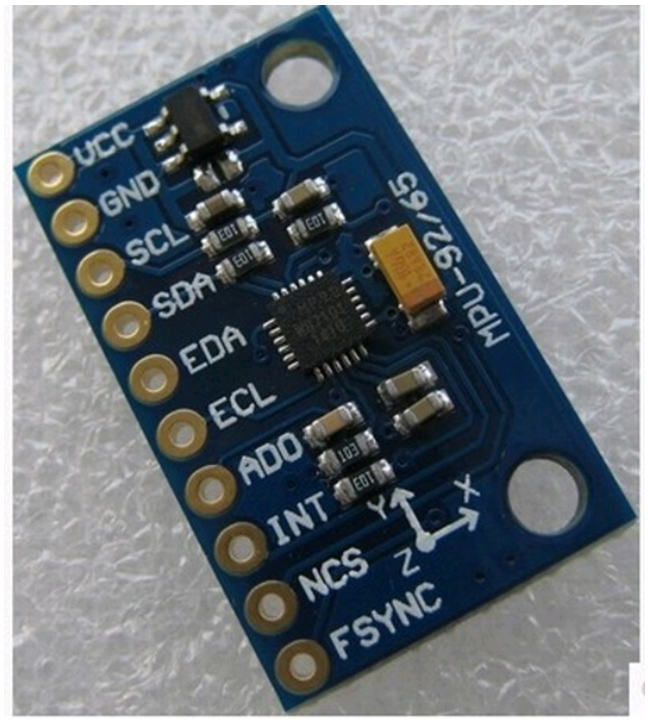
08_Angular Position with I2C Communication

Introduction:

To be completely autonomous, the robot needs to know what is its angular position. We will call these positions yaw, pitch, roll(more info on yaw, pitch,roll below). To calculate these values, we will need data from a gyroscope(angular speed, in rad/s), accelerometer (gravity force, in g) and magnetometer(magnetic field , in µT).

We used the MPU9265 from Invensense.

MPU9265:



The MPU9265 contains two distinct chips. The first one the MPU9250 is the main chip with two of the three sensors. The other chip is the magnetometer, the AK8963.

Connection :

Vcc : Connect to 5V even the chip works at 3.3V (Voltage regulator)

Gnd : Ground

SDA : Bus data

SCL : Clock

AD0 : Put 0 or 1 . If 0, changes the slave address of the MPU9250 to 0x68, 1 to 0x69. Use in case we have 2 MPU9265 into the same I2C network

I2C:

Into this I2C communication we consider the AURIX™ as the master and the MPU as the slave. As the SPI is quicker (1MHz) than I2C (400kHz) communication,SPI/I2C can have as many as there is slave, you may have the question, why did they choose the I2C than the SPI. Well for only one reason. Indeed the I2C, allow us to have more than one master with its particular protocol.

Quick sum up :

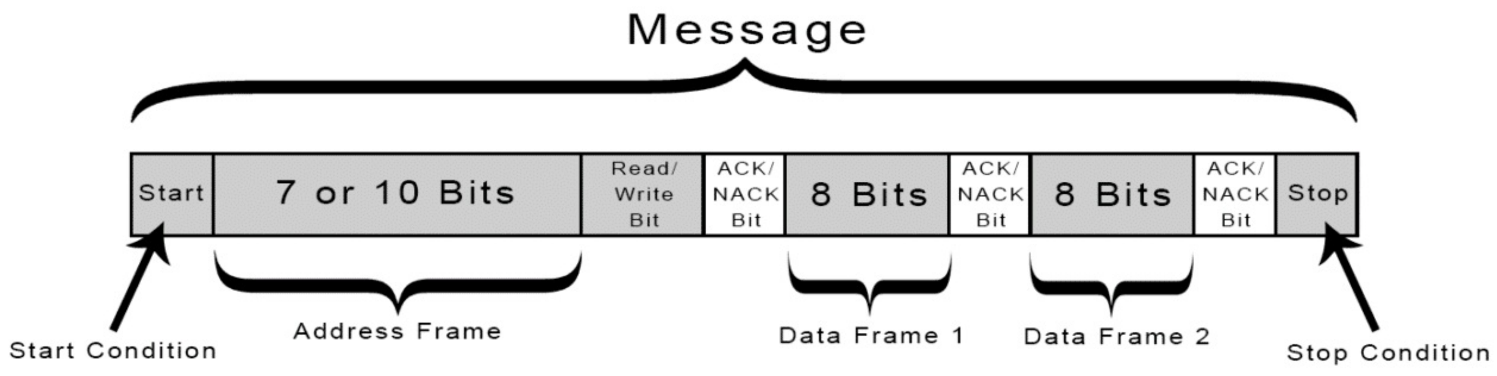
1. I2C protocol
2. Code Explanation

I2C Protocol:

The I2C is a communication protocol where each device into the network has a unique address. The communication is done with only two data bus. The SCL bus (Clock) and the SDA bus (Data).

Below you see a message (SDA bus) between a master and one of its slave. At the beginning of each message we have a start bit and a the end a stop bit.

We can cut the message in two parts. The first one is for the addressing and the second one for the data.



Addressing:

For this communication protocol, all the devices into the network have a unique address of 7 or 10 bits. In our case we will use a 7 bits address. The last bit of the addressing frame is the "Read/Write" bit. Indeed, when we are addressing the device we also set/or not this bit for a reading or a writing action to be done. See this bit from the master point of you, so write means, you'll write into a register of the slave device.

After the addressing part, finished by the R/W bit, the slave send to the master an Acknowledge bit (level 0)

Code Explanation :

In this part we will explain how we managed to initialize the I2C communication on an AURIX™ (TC297). What registers into the MPU9265 we initialized, and how work a write and read with I2C.

Initialisation of the I2C communication and the MPU9265 :

I2C Communication :

For the I2C communication's initialisation we will initialise 3 Module :

```
- i2cHandle

- i2cDev_Gyro_Accel

- i2cDev_Mag
```

"i2cHandle" will have as I2C parameters the pins (here SDA = P2.4, SCL = P2.5), and the frequency's communication. The frequency is forced by the MPU at 400kHz.

You can see that there is two device modules for only one mentioned slave adrress (see above). Well, the MPU9265 is composed of 2 chips MPU9250 and the AK8963, and each chip has a different slave address, the MPU9250 (0x68 or 0x69) and the AK8963 (0x0C). So the modules are the same except for the device slave address. We shift the slave addres by one on the left for R/W bit which will be add when sending message.

MPU registers :

Meanwhile initialising the MPU's registers, we also initialise some variables for the data processing such as the scale factor(bits->dps).

For the MPU9250, we intiailise the registers below :

- The gyroscope scale value
- The accelerometer scale value
- Filter at 5Hz for the gyroscope values
- Filter at 5Hz for the accelerometer values
- Bypass Mode. This mode is enable in order to communicate with the magnetometer (AK8963)

For the AK8963, we initialise the register below:

- The magnetometer ADC 16 bits mode

Read/Write with I2C.

When you look at the ifx function read and write you see that you need to pass as parameters the device config, a table and the number of bytes you want to read/write.

Write : In the table, write in the first byte the sensor register's address, and next the value you want to write (see register map in bibliograhy)

Ex:

```
I2C_Masters_Slaves.i2cBuffer.i2cTxBuffer[0] = 0x1C; //Register address

I2C_Masters_Slaves.i2cBuffer.i2cTxBuffer[1] = scale_acc;//value

while(IfxI2c_I2c_write(&I2C_Masters_Slaves.drivers.i2cDev_Gyro_Accel,&I2C_Masters_Slaves.i2cBuffer.i2cTxBuffer[0],2) == IfxI2c_I2c_Status_nak); //write opertation
```

Read : To make a read you need first to make a write. Indeed in this case the sensor will be able to know which register he has to read and transfer it to the master. So in this case you just need to write the register address into the parameter table.

Ex :

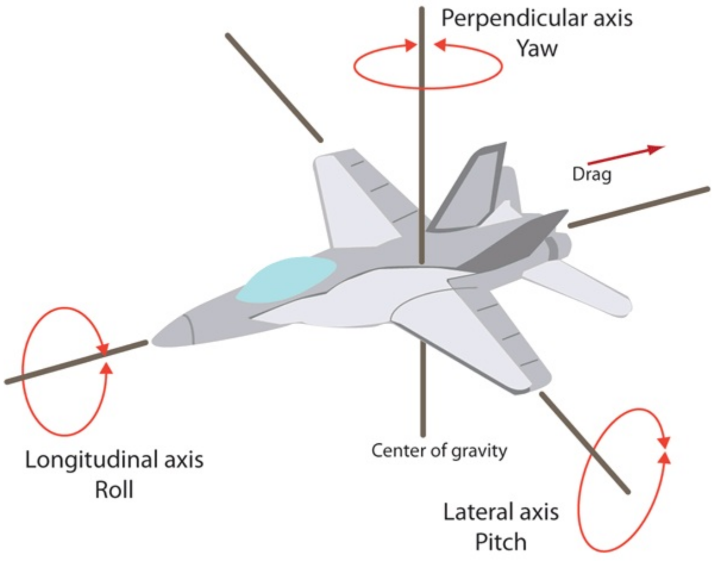
```
I2C_Masters_Slaves.i2cBuffer.i2cTxBuffer[0] = Register_Address;
```

```
while(IfxI2c_I2c_write(Device_I2C, pointertx, 1) == IfxI2c_I2c_Status_nak);

while(IfxI2c_I2c_read(Device_I2C, pointerrx, Number_Received) == IfxI2c_I2c_Status_nak);
```

Data Processing :

The aim is to get the angular position of the robot. To do so we need informations from the gyroscope(dps), the accelerometer(g) and the magnetometer (µT). The Madgwick filter, with all these informations,will calculate the angular position (Yaw, Pitch, Roll).



However, we can't give the data to the Madgwick filter without any data processing.

Sum-up :

```
1. Calibration
2. Madgwick Filter
```

Calibration :

For the three sensors, the output from the I2C communication is coded by the ADC. So to retrieve the measured value, we need to convert into its original unit. For example the data from the gyroscope are divided by a factor in LSB/dps.

Also, the sensors need to be calibrate.

Gyroscope's calibration :

When you don't move the sensor, the values have to reach 0 on each axis.

Accelerometer's calibration :

When you don't move the sensor, on x and y axis, the values should reach 0. However on z you have to reach the value -1 or 1 it depends on the orientation of the sensor (gravity = 1g).

Magnetometer's calibration :

For the magnetometer's calibration you need to see the value's device as a projection of z on the axis x,y. So it means in this position the values on z are not reliable for the calibration.

In order to calibrate, you have to check when the value is the highest and the smallest. For example, on x I have a -25 and -2 as maximum and minimum. The magnetometer's values need to be centered to 0. So i need to add 13.5 to the x values and i will have a variation between -11.5 and 11.5 centered on 0.

Now for z you just have to change the orientation of the device, z mag reference moves to the x earth reference and vice versa.

After the Madgwick filter works it is easier to check if the magnetometer works well. Indeed like on the triginometric ring, cos(α) = mag x position, and sin(α)=mag y position. So it means when you are at π/4 mod π/2, mag x= ± mag y. In this case α corresponds to the yaw value.

Madgwick Filter :

In this section we will not explain the Madgwick filter, (info : bibliography) but how we managed to make it runs well.

We implemented the last Madgwick filter schema without "Group 2" as you can see on the bibliography's link.

The Madgwick filter is based on an interuption every 10ms (100Hz). To do so we used a GTM interruption on the CPU1.

Also the Madgwick filter is intialised meanwhile the I2C communication initialisation.

Thanks to Mayeul Jeannin for his help.

Written by Guillaume Nicolle on 14th March 2019

Bibliography:

MPU9265:

Register Map:

https://www.invensense.com/wp-content/uploads/2015/02/RM-MPU-9250A-00-v1.6.pdf

MPU9265 description:

https://www.invensense.com/wp-content/uploads/2015/02/PS-MPU-9250A-01-v1.1.pdf

Madgwick Filter:

https://www.samba.org/tridge/UAV/madgwick_internal_report.pdf

https://en.wikipedia.org/wiki/Fast_inverse_square_root

Arduino's test program, in the folder "MPU9265 Arduino test program"