Smart Software Project

Lecture: Week 12

Gyro, Accelerometer
& Compass Sensors

Prof. HyungJune Lee
hyungjune.lee@ewha.ac.kr

이화여자대학교
EWHA WOMANS UNIVERSITY

Today

- Review from the last lecture
  - – Infrared sensors
  - – Line tracing

- SmartCAR Gyro + Accelerometer sensors
- SmartCAR Compass sensor

- Announcement

Class Schedule

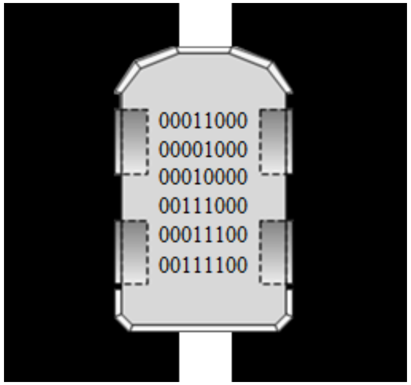
Week	Lecture Contents	Lab Contents
Week 1	Course introduction	Arduino introduction: platform & programming environment
Week 2	Embedded system overview & source management in collaborative repository (using GitHub)	Lab 1: Arduino Mega 2560 board & SmartCAR platform
Week 3	ATmega2560 Micro-controller (MCU): architecture & I/O ports, Analog vs. Digital, Pulse Width Modulation	Lab 2: SmartCAR LED control
Week 4	Analog vs. Digital & Pulse Width Modulation	Lab 3: SmartCAR motor control (Due: HW on creating project repository using GitHub)
Week 5	ATmega2560 MCU: memory, I/O ports, UART	Lab 4: SmartCAR control via Android Bluetooth
Week 6	ATmega2560 UART control & Bluetooth communication between Arduino platform and Android device	Lab 5: SmartCAR control through your own customized Android app (Due: Project proposal)
Week 7	Midterm exam	
Week 8	ATmega2560 Timer, Interrupts & Ultrasonic sensors	Lab 6: SmartCAR ultrasonic sensing
Week 9	Infrared sensors & Buzzer	Lab 7: SmartCAR infrared sensing
Week 10	Acquiring location information from Android device & line tracing	Lab 8: Implementation of line tracer
Week 11	Gyroscope, accelerometer, and compass sensors	Lab 9: Using gyroscope, accelerometer, and compass sensors
Week 12	Project	Team meeting (for progress check)
Week 13	Project	Team meeting (for progress check)
Week 14	Course wrap-up & next steps	
Week 15	Project presentation & demo I **June 13** (Due: source code, presentation slides, & poster slide)	Project presentation & demo II
Week 16	Final week (no final exam)	

Today

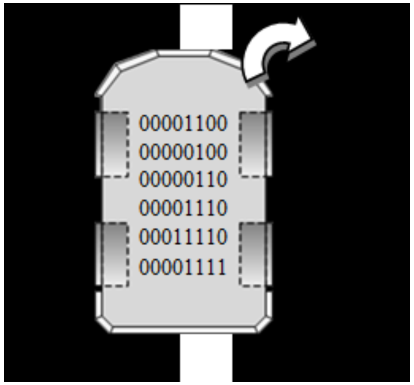
- **Review from the last lecture**
  - Infrared sensors
  - Line tracing
- SmartCAR Gyro + Accelerometer sensors
- SmartCAR Compass sensor
- Announcement

Line Tracer

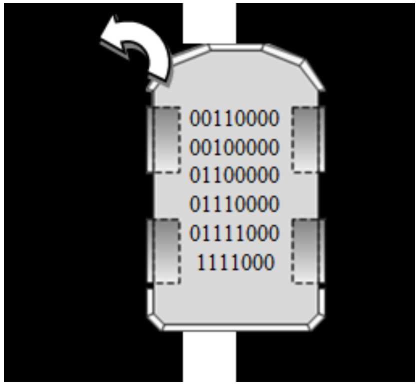
- Line tracing in SmartCAR
  - Infrared sensor data depending on SmartCAR's position



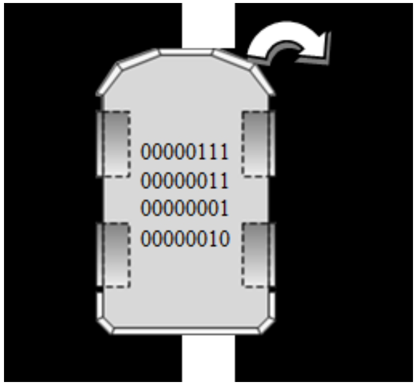
(a) Forward



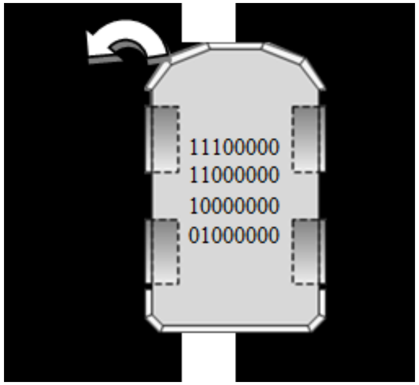
(b) Smooth Right-turn



(c) Smooth Left-turn

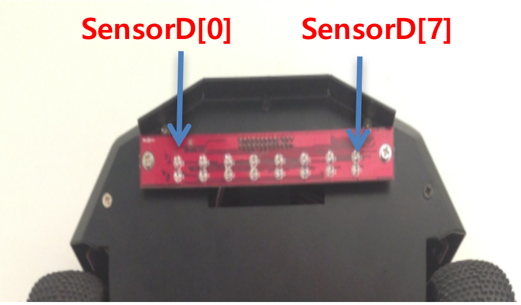


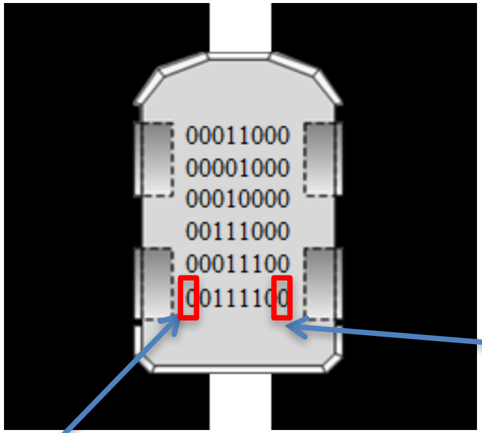
(d) Pivot Right-turn



(e) Pivot Left-turn

Sensor Data





```
...
...

unsigned char sensor_data = 0;
int z;

for(z=0;z<8;z++)
{
    unsigned int val = digitalRead(SensorD[z]);
    sensor_data |= (val << z);
}
```

SensorD[7]	SensorD[6]	SensorD[5]	SensorD[4]	SensorD[3]	SensorD[2]	SensorD[1]	SensorD[0]

To track black line in white background,

- we should complement the sensor_data ('1' to '0', '0' to '1')

```
sensor_data = ~sensor_data;
```

Control Motors w.r.t. Infrared Sensor

- How to control motors w.r.t. sensor_data

Sensor_data	Direction	Speed_data_L	Speed_data_R	Etc
0x18	FORWARD	140	140	Forward
0x10				
0x08				
0x38				
0x1C				
0x3C				
0x0C	RIGHT	200	0	Smooth Right Turn
0x04				
0x06				
0x0E				
0x1E				
0x0F				
0x30	LEFT	0	200	Smooth Left Turn
0x20				
0x60				
0x70				
0x78				
0xF0				
0x07	PIVOT_RIGHT	200	80	Pivot Right Turn
0x03				
0x02				
0x01				
0xC0	PIVOT_LEFT	80	200	Pivot Left Turn
0x40				
0x80				
0xE0				
0x00	STOP	0	0	Stop

Today

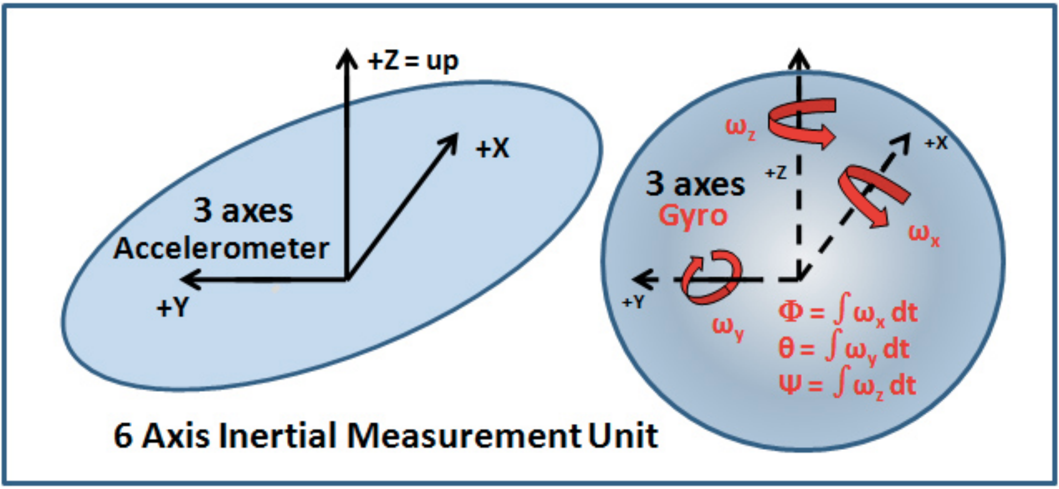
- Review from the last lecture
  - Infrared sensors
  - Line tracing

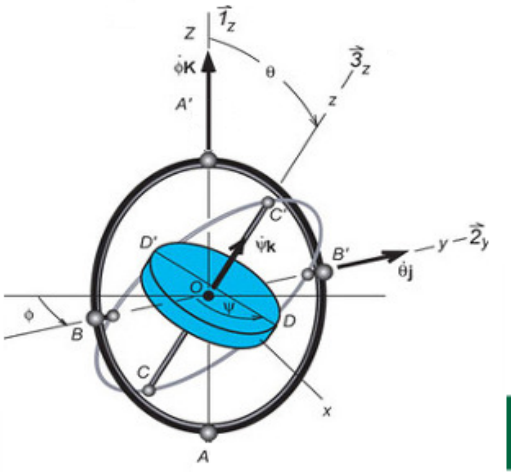
- **SmartCAR Gyro + Accelerometer**
- SmartCAR Compass sensor

- Announcement

3 axes Accelerometer + 3 axes Gyro

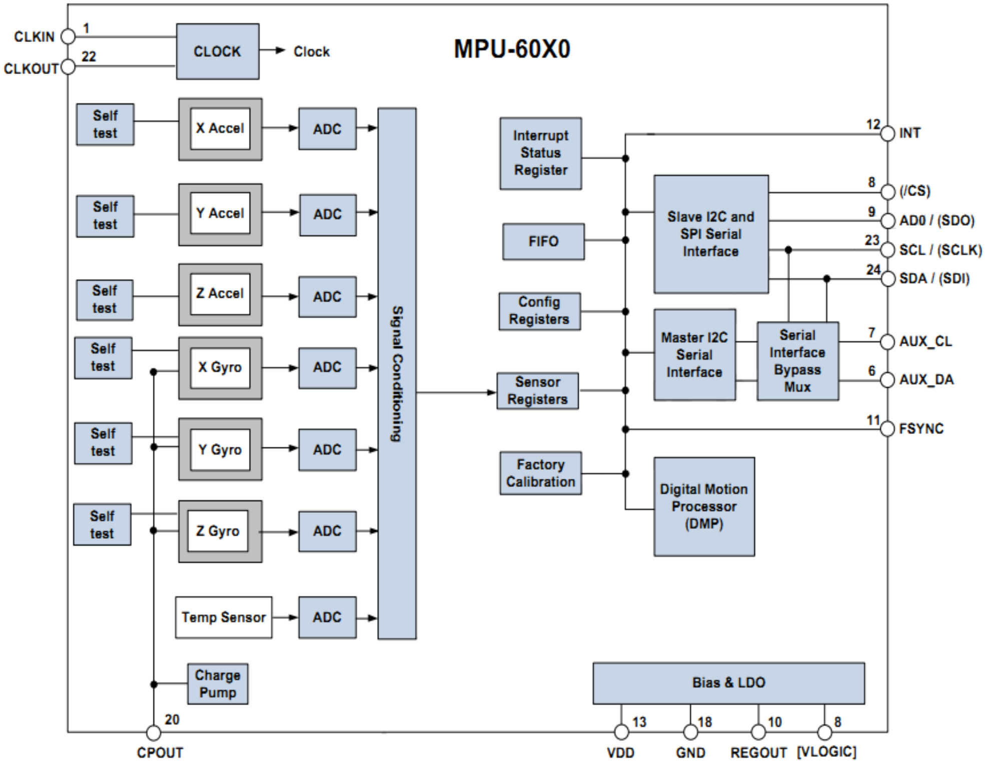
- 6 axes
  - 1) from 3 axes accelerometer
    - Measure proper acceleration in three axes
  - 2) from 3 axes gyroscope
    - Measure orientation, based on the principles of angular momentum

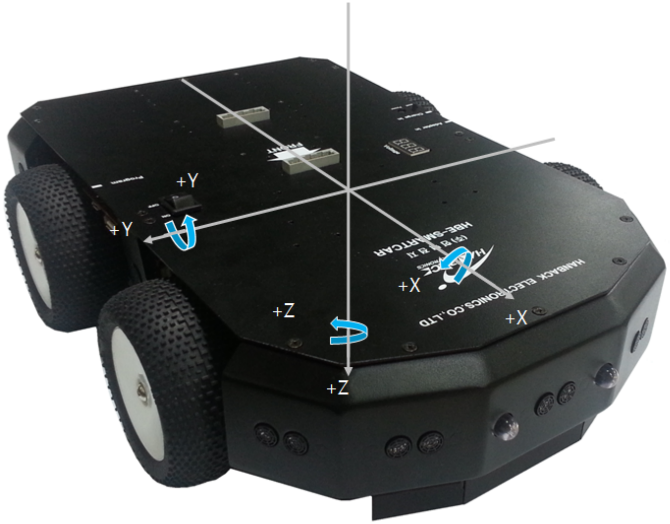

+Z = up
+X
3 axes
Accelerometer
+Y
3 axes
Gyro
ω_z
+Z
+X
ω_x
+Y
ω_y
$\Phi = \int \omega_x \, dt$
$\theta = \int \omega_y \, dt$
$\Psi = \int \omega_z \, dt$
6 Axis Inertial Measurement Unit



6 Axes Sensors in SmartCAR

- InvenSense MPU-6050: Gyro 3 axis + Accelerometer 3 axis sensors
  - Data transmission type to Atmega MCU
    - I²C(Inter-Integrated Circuit) interface





Sensor Sensitivity

- Accelerometer
  - Sensitivity

Full-Scale Range	AFS_FEL	Sensitivity Scale Factor	Etc
±2 g	0	16384 LSB/g	
±4 g	1	8192 LSB/g	
±8 g	2	4096 LSB/g	
±16 g	3	2048 LSB/g	

- Gyroscope
  - Sensitivity

Full-Scale Range	FS_FEL	Sensitivity Scale Factor	Etc
±250 °/sec	0	131 LSB/(°/sec)	
±500 °/sec	1	65.5 LSB/(°/sec)	
±1000 °/sec	2	32.8 LSB/(°/sec)	
±2000 °/sec	3	16.4 LSB/(°/sec)	

- Basic setting: ±2g for accelerometer, ±250 °/sec for gyroscope

Port Configuration for 6 axes Sensor

Chip	Name	Port / Number	Etc
MPU-6050	SDA	PD1 / 20	
	SCL	PD0 / 21	

Sample Program

- Accelgyro_sensor.h

```
#ifndef Accelgyro_sensor_H_
#define Accelgyro_sensor_H_
#include "Arduino.h"

#include "Wire.h"
#include "I2Cdev.h"
#include "MPU6050.h"

#ifdef __cplusplus
extern "C" {
#endif
void loop();
void setup();
#ifdef __cplusplus
}
#endif

#endif
```

Sample Program

- Accelgyro_sensor.cpp – (1)

```
#include "Accelgyro_sensor.h"

MPU6050 accelgyro;

int16_t ax, ay, az;
int16_t gx, gy, gz;

void setup()
{
    Wire.begin();
    Serial.begin(115200);
    Serial.println("Initializing I2C devices...\r");
    accelgyro.initialize();

    Serial.println("Testing device connections...\r");
    Serial.println(accelgyro.testConnection() ? "MPU6050 connection successful\r" : "MPU6050 connection failed\r");
}

void loop()
{

    accelgyro.getMotion6(&ax, &ay, &az, &gx, &gy, &gz);

    Serial.print("Accel : ");
```

Sample Program

- Accelgyro_sensor.cpp – (2)

```
    Serial.print(ax);
    Serial.print(" ");
    Serial.print(ay);
    Serial.print(" ");
    Serial.print(az);
    Serial.print(" Gyro : ");
    Serial.print(gx);
    Serial.print(" ");
    Serial.print(gy);
    Serial.print(" ");
    Serial.println(gz);
    delay(200);
}
```

Code Explanation

- Header file
  - Include several header files
    - Include Wire.h and I2Cdev.h to use I2C hardware interface
    - Include MPU6050.h to use MPU6050

```
#include "Wire.h"
#include "I2Cdev.h"
#include "MPU6050.h"
```

- Main program
  - Global variable declaration
    - ax, ay, az: variables for accelerometer values
    - gx, gy, gz: variables for gyroscope values

```
int16_t ax, ay, az;
int16_t gx, gy, gz;
```

  - I2C interface initialization
    - Set up to 50KHz and enable it

```
    Wire.begin();
```

Code Explanation

- Sensor initialization
  - Clock: call setClockSource()
  - Gyro sensitivity: call setFullScaleGyroRange()
  - Accelerometer sensitivity: call setFullScaleAccelRange()
  - Sleep mode off: call setSleepEnabled()

```
     accelgyro.initialize();

    setClockSource(MPU6050_CLOCK_PLL_XGYRO);
    setFullScaleGyroRange(MPU6050_GYRO_FS_250);
    setFullScaleAccelRange(MPU6050_ACCEL_FS_2);
    setSleepEnabled(false);
```

Code Explanation

- Serial.println() function
  - Check whether the MPU 6050 connection is successful

```
Serial.println(accelgyro.testConnection() ? "MPU6050 connection
successful\r" : "MPU6050 connection failed\r");

 return getDeviceID() == 0x34;
```

- Accelgyro.getMotion6() function
  - Read 6 values from accelerometer and gyroscope via I2C interface
  - I2Cdev::readBytes(devAddr, MPU6050_RA_ACCEL_XOUT_H, 14, buffer)

```
accelgyro.getMotion6(&ax, &ay, &az, &gx, &gy, &gz);

void MPU6050::getMotion6(int16_t* ax, int16_t* ay, int16_t* az, int16_t* gx,
int16_t* gy, int16_t* gz) {
     I2Cdev::readBytes(devAddr, MPU6050_RA_ACCEL_XOUT_H, 14, buffer);
     *ax = (((int16_t)buffer[0]) << 8) | buffer[1];
     *ay = (((int16_t)buffer[2]) << 8) | buffer[3];
     *az = (((int16_t)buffer[4]) << 8) | buffer[5];
     *gx = (((int16_t)buffer[8]) << 8) | buffer[9];
     *gy = (((int16_t)buffer[10]) << 8) | buffer[11];
     *gz = (((int16_t)buffer[12]) << 8) | buffer[13];
}
```

Code Explanation

– Read data values and send them to bluetooth

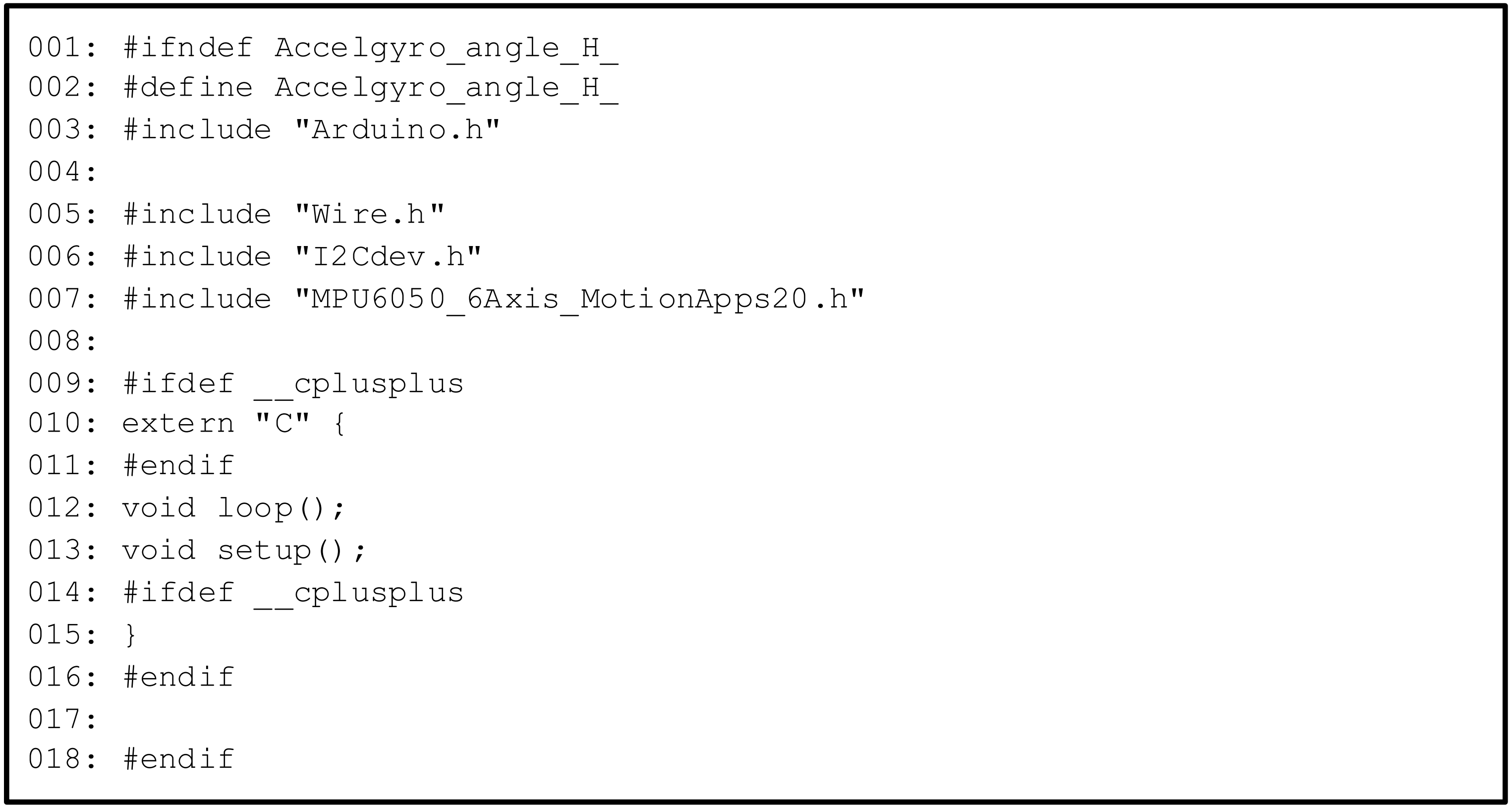
```
    Serial.print("Accel : ");
    Serial.print(ax);
    Serial.print(" ");
    Serial.print(ay);
    Serial.print(" ");
    Serial.print(az);
    Serial.print(" Gyro : ");
    Serial.print(gx);
    Serial.print(" ");
    Serial.print(gy);
    Serial.print(" ");
    Serial.println(gz);
```

Yaw, pitch, and roll calculation

- Let’s have the followings from 6 axes measurements
  - Yaw: rotation in z-axis
  - Pitch: rotation in y-axis
  - Roll: rotation in x-axis

Sample Program

- Accelgyro_angle.h



```
#ifndef Accelgyro_angle_H_
#define Accelgyro_angle_H_
#include "Arduino.h"

#include "Wire.h"
#include "I2Cdev.h"
#include "MPU6050_6Axis_MotionApps20.h"

#ifdef __cplusplus
extern "C" {
#endif
void loop();
void setup();
#ifdef __cplusplus
}
#endif

#endif
```

Sample Program

– Accelgyro_angle.cpp – (1)

```
#include "Accelgyro_angle.h"

MPU6050 mpu;

uint8_t mpuIntStatus;   // holds actual interrupt status byte from MPU
uint8_t devStatus;      // return status after each device operation (0 = success, !0 = error)
uint16_t packetSize;    // expected DMP packet size (default is 42 bytes)
uint16_t fifoCount;     // count of all bytes currently in FIFO
uint8_t fifoBuffer[64]; // FIFO storage buffer

Quaternion q;           // [w, x, y, z]         quaternion container
VectorFloat gravity;    // [x, y, z]            gravity vector
float ypr[3];           // [yaw, pitch, roll]   yaw/pitch/roll container and gravity vector

void setup()
{
  Wire.begin();

  Serial.begin(115200);

  Serial.println(F("Initializing I2C devices..."));
  mpu.initialize();
```

Sample Program

– Accelgyro_angle.cpp – (2)

```

  Serial.println(F("Testing device connections..."));
  Serial.println(mpu.testConnection() ? F("MPU6050 connection successful") : F("MPU6050 connection failed"));

  Serial.println(F("Initializing DMP..."));
  devStatus = mpu.dmpInitialize();

  if (devStatus == 0) {

    Serial.println(F("Enabling DMP..."));
    mpu.setDMPEnabled(true);

    packetSize = mpu.dmpGetFIFOPacketSize();
  }
  else
  {
    Serial.print(F("DMP Initialization failed (code "));
    Serial.print(devStatus);
    Serial.println(F(")"));
  }
}

void loop()
{
```

Sample Program

– Accelgyro_angle.cpp – (3)

```
  while(1)
  {
    mpu.resetFIFO();
    mpuIntStatus = mpu.getIntStatus();
    fifoCount = mpu.getFIFOCount();
    if(mpuIntStatus & 0x02)
    {
      while(fifoCount < packetSize)
        fifoCount = mpu.getFIFOCount();
      mpu.getFIFOBytes(fifoBuffer, packetSize);
      fifoCount -= packetSize;
      mpu.dmpGetQuaternion(&q, fifoBuffer);
      mpu.dmpGetGravity(&gravity, &q);
      mpu.dmpGetYawPitchRoll(ypr, &q, &gravity);
      break;
    }
  }
  Serial.print(" yaw : ");
  Serial.print(180 - (ypr[0] * 180/M_PI));  //yaw
  Serial.print("  pitch : ");
  Serial.print(ypr[1] * 180/M_PI);  // pitch
  Serial.print("  roll : ");
  Serial.println(ypr[2] * 180/M_PI);  //roll
  delay(200);
}
```

Code Explanation

- Header file
  - Include a header file
    - Use DMP(Digital Motion Processor) functionality with MPU6050 sensor
    - MPU6050_6Axis_MotionApps20.h

```
#include "MPU6050_6Axis_MotionApps20.h"
```

Code Explanation

- Main program
  - Variable declaration
    - mpuIntStatus: current interrupt status
    - devStatus: current device status
    - packetSize: data packet size
    - fifoCount: # of data stored in FIFO
    - fifioBuffer[]: buffer for reading data from FIFO

```
uint8_t mpuIntStatus;
uint8_t devStatus;
uint16_t packetSize;
uint16_t fifoCount;
uint8_t fifoBuffer[64];
```

- Quaternion q: four dimensions
  - [q.w, q.x, q.y, q.z] from DMP
- VectorFloat gravity: gravity vector
  - [gravity.x, gravity.y, gravity.z] derived from Quaternion
- ypr[]: to store yaw, pitch, roll values
  - Calculated from Quaternion and gravity values

```
Quaternion q;
VectorFloat gravity;
float ypr[3];
```

Code Explanation

- dmpInitialize() function
  - Initialize DMP

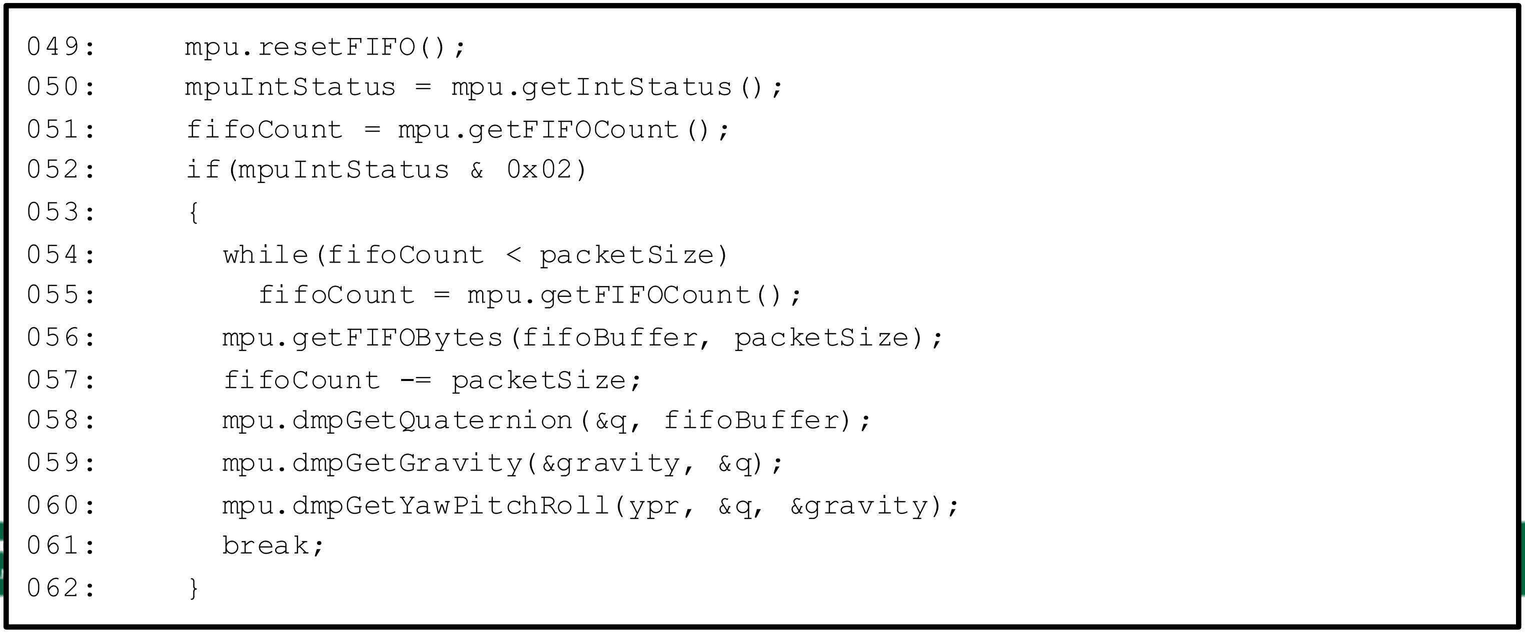
```
devStatus = mpu.dmpInitialize();
```

- After a successful DMP initialization,
  - Enable DMP: setDMPEnabled(true);
  - Store FIFO packet size (=42)

```
if (devStatus == 0) {

  Serial.println(F("Enabling DMP..."));
  mpu.setDMPEnabled(true);

  packetSize = mpu.dmpGetFIFOPacketSize();
}
else
{
  Serial.print(F("DMP Initialization failed (code "));
  Serial.print(devStatus);
  Serial.println(F(")"));
}
```

Code Explanation

- loop() function
  - resetFIFO(): reset FIFO buffer
  - getIntStatus(): read the current interrupt status (store it at **mpuIntStatus**)
  - getFIFOCount(): read the current # of data in FIFO buffer (store it at **fifoCount**)
  - If mpuIntStatus & 0x02 is equal to 0x02, then do the following procedures:
    - Until fifoCount >= packetSize, fifoCount stores getFIFOCount() value
    - fifoBuffer[]: to store data from DMP FIFO
    - fifoCount is updated to fifoCount – packetSize
    - Obtain the quaternion value by calling dmpGetQuaternion() from fifoBuffer
    - Calculate the gravity value from the quaternion, and then calculate ypr[] values



```
mpu.resetFIFO();
mpuIntStatus = mpu.getIntStatus();
fifoCount = mpu.getFIFOCount();
if(mpuIntStatus & 0x02)
{
  while(fifoCount < packetSize)
    fifoCount = mpu.getFIFOCount();
  mpu.getFIFOBytes(fifoBuffer, packetSize);
  fifoCount -= packetSize;
  mpu.dmpGetQuaternion(&q, fifoBuffer);
  mpu.dmpGetGravity(&gravity, &q);
  mpu.dmpGetYawPitchRoll(ypr, &q, &gravity);
  break;
}
```

Code Explanation

- fifobuffer data format

0	1	2	3	4	5
QUAT W		-		QUAT X	
6	7	8	9	10	11
-		QUAT Y		-	
12	13	14	15	16	17
QUAT Z		-		GYRO X	
18	19	20	21	22	23
-		GYRO Y		-	
24	25	26	27	28	29
GYRO Z		-		ACC X	
30	31	32	33	34	35
-		ACC Y		-	
36	37	38	39	40	41
ACC Z		-		-	

- dmpGetGravity() function

```
uint8_t MPU6050::dmpGetGravity(VectorFloat *v, Quaternion *q) {
     v -> x = 2 * (q -> x*q -> z - q -> w*q -> y);
     v -> y = 2 * (q -> w*q -> x + q -> y*q -> z);
     v -> z = q -> w*q -> w - q -> x*q -> x - q -> y*q -> y + q -> z*q -> z;
}
```

Code Explanation

- dmpGetYawPitchRoll() function

```
uint8_t MPU6050::dmpGetYawPitchRoll(float *data, Quaternion *q, VectorFloat
*gravity) {
     data[0] = atan2(2*q -> x*q -> y - 2*q -> w*q -> z, 2*q -> w*q -> w +
2*q -> x*q -> x - 1);
     data[1] = atan(gravity -> x / sqrt(gravity -> y*gravity -> y + gravity
-> z*gravity -> z));
     data[2] = atan(gravity -> y / sqrt(gravity -> x*gravity -> x + gravity
-> z*gravity -> z));
 }
```

```
Serial.print(" yaw : ");
Serial.print(180 - (ypr[0] * 180/M_PI));
Serial.print("  pitch : ");
Serial.print(ypr[1] * 180/M_PI);
Serial.print("  roll : ");
Serial.println(ypr[2] * 180/M_PI);
```

- We convert the measured values in radian to degree
- Then, send them to bluetooth

Today

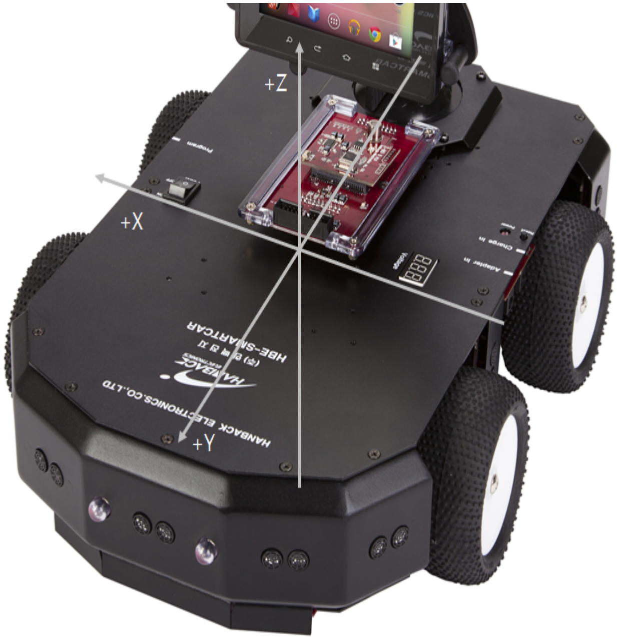
- Review from the last lecture
  - Infrared sensors
  - Line tracing

- SmartCAR Gyro + Accelerometer sensors
- **SmartCAR Compass sensor**

- Announcement

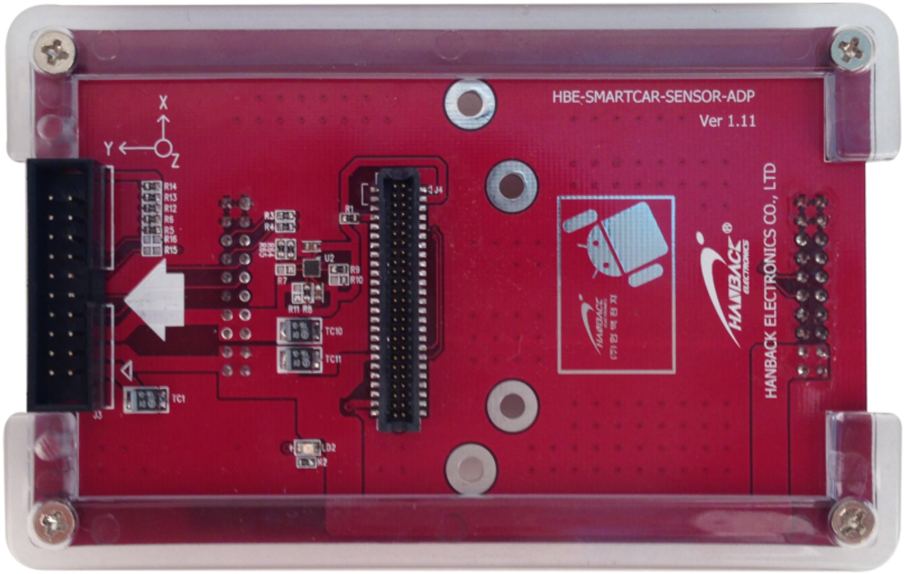
Compass Sensor Direction

- Compass Sensor Direction
  - Value will be the maximum when it points to magnetic north



Compass Sensor in SmartCAR

- Compass sensor is connected to extension board



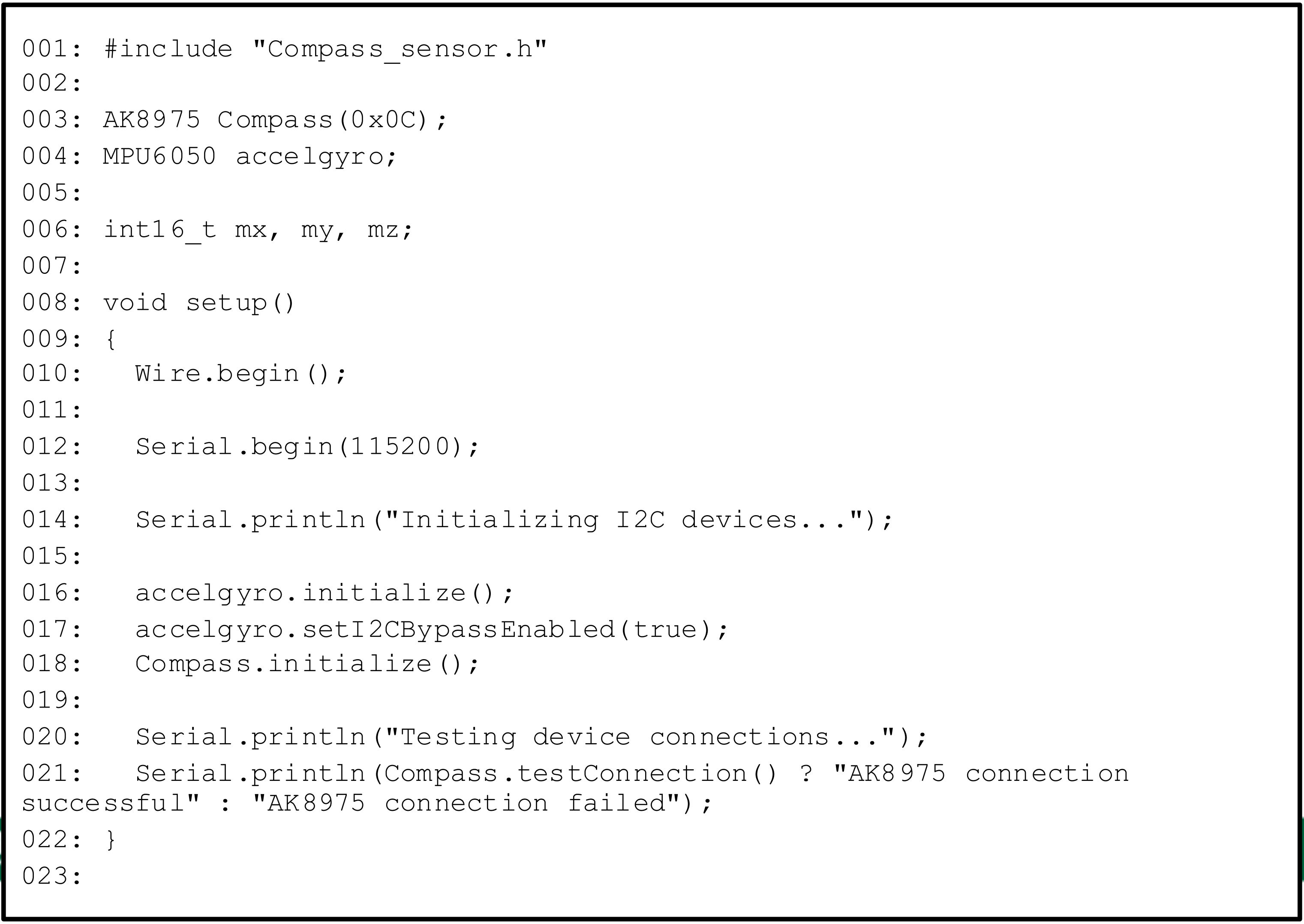
Sample Program

– Compass_sensor.h

```
#ifndef Compass_sensor_H_
#define Compass_sensor_H_
#include "Arduino.h"

#include "Wire.h"
#include "I2Cdev.h"
#include "MPU6050.h"
#include "AK8975.h"

#ifdef __cplusplus
extern "C" {
#endif
void loop();
void setup();
#ifdef __cplusplus
}
#endif

#endif
```

Sample Program

– Compass_sensor.cpp – (1)



```
#include "Compass_sensor.h"

AK8975 Compass(0x0C);
MPU6050 accelgyro;

int16_t mx, my, mz;

void setup()
{
  Wire.begin();

  Serial.begin(115200);

  Serial.println("Initializing I2C devices...");

  accelgyro.initialize();
  accelgyro.setI2CBypassEnabled(true);
  Compass.initialize();

  Serial.println("Testing device connections...");
  Serial.println(Compass.testConnection() ? "AK8975 connection successful" : "AK8975 connection failed");
}

```

Sample Program

– Compass_sensor.cpp – (2)

```
void loop()
{
  Compass.getHeading(&mx, &my, &mz);
  Serial.print(" Compass : ");
  Serial.print(mx);
  Serial.print("  ");
  Serial.print(my);
  Serial.print("  ");
  Serial.println(mz);
  delay(200);
}
```

Code Explanation

- Header file

```
#include "AK8975.h"
```

  - Include AK8975.h to use compass sensor in AK8975

- Main program
  - Global variable declaration
    - Create an object named “Compass” in AK8975 class
    - Initialize a member variable to 0x0C (as compass sensor address)
    - mx,my,mz : store data from Compass sensor

```
AK8975 Compass(0x0C);
MPU6050 accelgyro;

int16_t mx, my, mz;
```

Code Explanation

- Setup() function
  - Initialize 6 axes sensors, and enable I2CBypass mode
    - To directly control Compass sensor

```
accelgyro.initialize();
accelgyro.setI2CBypassEnabled(true);
Compass.initialize();

Serial.println("Testing device connections...");
Serial.println(Compass.testConnection() ? "AK8975 connection successful" : "AK8975 connection failed");
```

- loop() function
  - Read data by calling getHeading() and store them at mx, my, mz
  - Send them to bluetooth

```
Compass.getHeading(&mx, &my, &mz);
Serial.print(" Compass : ");
Serial.print(mx);
Serial.print("  ");
Serial.print(my);
Serial.print("  ");
Serial.println(mz);
```

Course Announcement

- For lab session, we will cover
  - Using gyro, accelerometer, and compass sensors

- Next Week
  - Project discussion meetings: 10 – 15 minutes per team
  - Team 1 ~ 5