3D Accelerometer

for RP6 Robot, but also useable for RP6 WIFI, ARUDINO and ASURO

Introduction:

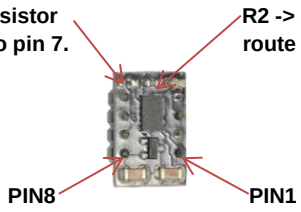
General purpose 3D accelerometer module for robots. This 3D accelerometer can determine the absolute angle of your robot, e.g. Roll and Pitch value can be easily calculated by a microcontroller. In addition you can calculate the vector of a collision impulse. This information can be used to move around an obstacle.

Hint: Often, the differences between a Gyro and an Accelerometer is not clear and got mixed up.
Here a clarification:

A gyroscope is very good in finding the rate of an angular change, but can't detect an absolute angle – but an accelerometer does!

1.) PIN OUT Accelerometer Module:

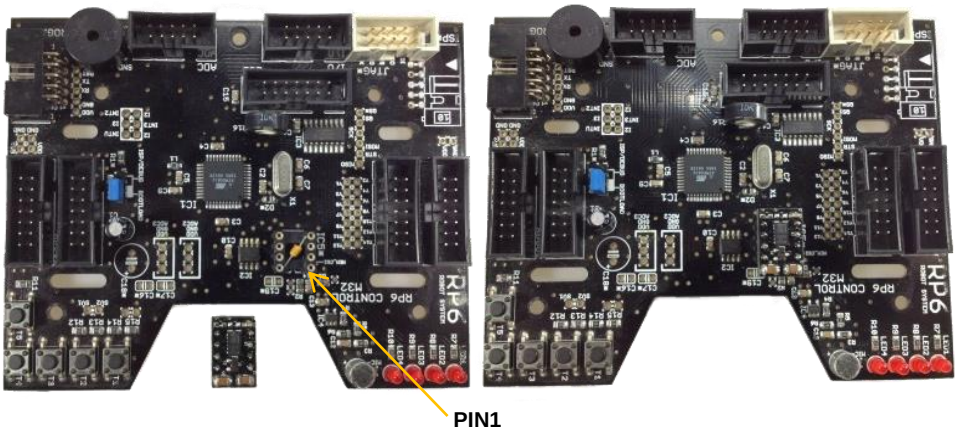
PIN1	CS/SPI enable	PIN5	SI/SDA
PIN2	SO/SDO (I2C address)	PIN6	SPC/SCL
PIN3	INT1	PIN7	INT2
PIN4	GND	PIN8	VCC



JM3 Engineering Inh. J. Maisel Dobelweg 18, D-85567 Grafing / Munich
Email: JHM@JM3-Engineering.de

2.) Connecting to RP6 Robot

The picture on the left shows the M32 board with an 8 pin DIL socket.
The Accelerometer module fits on the boards as drop in solution.



PIN3 and PIN 7 on M32 Board are connected to Vdd ! Recommendation is to remove R2 and R3!

Pin1 is on the lower right hand side on the picture below (hence DIL socket).
VDD is 3.3 or 5.0 V nominal.

3.) Connecting to RP6WIFI, ASURO or ARUDINO

Alternatively you can connect this module to the I2C Bus instead of SPI. Therefore you need to pull PIN1 to VDD. PIN5 is then SDA while PIN 6 is SCL!
With PIN2 you can select the less significant bit of the device ID.
In this case you can use it together with the RP6WIFI, ASURO or ARUDINO Robot and the specific extension board!

4.) Software changes (SPI):

The data access and initialization works over SPI Bus (for details on Registers and functions refer to the datasheet of the LIS302DLH.

Change the software on the control board as following:

1. RP6ControlLib.h:

```
/*****************************************************************/
// SPI ACC Sensor
uint8_t ACCSENS_ReadByte(uint8_t regAddr);
void ACCSENS_ReadBytes(uint8_t startAddr, uint8_t* buffer, uint8_t length);
void ACCSENS_WriteByte(uint8_t regAddr, uint8_t data);
uint8_t ACCSENS_GetStatus();
```

2. RP6ControlLib.ccp

```
/*****************************************************************/
// SPI ACC Sensor  LIS302DLH

uint8_t cSPI::ACCSENS_ReadByte(uint8_t regAddr)
{
    regAddr |= 0x80;
    uint8_t data;

    PORTB &= ~ACC_CS;
    writeSPI(regAddr);
    data = readSPI();
    PORTB |= ACC_CS;

    return data;
}

// Reads "length" Bytes into the Buffer "buffer" from startAdr2 on

void cSPI::ACCSENS_ReadBytes(uint8_t startAddr, uint8_t* buffer, uint8_t length)
{
    startAddr |= 0xC0;

    PORTB &= ~ACC_CS;
    writeSPI(startAddr);
    readBufferSPI(buffer, length);
    PORTB |= ACC_CS;
}

// Write a single data byte to the specified sensor address

void cSPI::ACCSENS_WriteByte(uint8_t regAddr, uint8_t data)
{
    PORTB &= ~ACC_CS;
    writeSPI(regAddr);
    writeSPI(data);
    PORTB |= ACC_CS;
}

// Returns Sensor Status register - for checking if sensor is busy

uint8_t cSPI::ACCSENS_GetStatus()
{
    uint8_t status;

    PORTB &= ~ACC_CS;
    writeSPI(0x27);
    status = readSPI();
    PORTB |= ACC_CS;

    return status;
}
```

This SW example demonstrates the access to the module – your SW need to handle to output to a display or the terminal via RS 232. The part is much more capable as shown here - read data sheet for more information.

3. Initialisation of Sensor:

```
/*****************************************************************
 * initialize acceleration sensor module LIS302DLH
 * reg address MSB is auto-increment
 *
 * power up sensor, enable x, y, z axis
 * +/-2g scale, 50 Hz, no update while reading,
 * internal clock, all filter bypassed, Hpc = 1
 * CTRL_REG1:           1100 0111b:    Low Power Mode 10 Hz Output rate, update 50Hz,
                                       enable x, y, z axis
 * CTRL_REG2:           0000 0000b:    default
 * CTRL_REG3:           0000 0000b:    default
 * CTRL_REG4:           1100 1000b:    Block update, big endian, +/- 2g,
 * CTRL_REG5:           0000 0000b:    default
*****************************************************************/
```

Write the following data via SPI into the registers:

```
ACCSENS_WriteByte(0x20, 0xC7);
ACCSENS_WriteByte(0x21, 0x00);
ACCSENS_WriteByte(0x22, 0x00);
ACCSENS_WriteByte(0x23, 0x88);
ACCSENS_WriteByte(0x24, 0x00);
```

Register 0x21, 0x22, 0x24 and 0x25 doesn't have to be written to, if default values are used.

4. Reading Sensor data:

Read sensor register exapmle:

1.) BYTE read:

```
buffer = ACCSENS_ReadByte(0x0F);
```

2.) Read bytes of registers for e.g. x-axis and store it into a variable (WORD):

```
buffer = ACCSENS_ReadByte(0x2A) | (ACCSENS_ReadByte(0x2B) << 8);
```

5.) Software changes (I2C):

The data access and initialization works over I2C Bus (for details on Registers and functions refer to the datasheet of the LIS302DLH.

Add to your I2C implementation the following steps:

5.1) Initialization of Sensor:

```
/*****************************************************************
 * initialize acceleration sensor module LIS302DLH
 * reg address MSB is auto-increment
 *
 * power up sensor, enable x, y, z axis
 * +/-2g scale, 50 Hz, no update while reading,
 * internal clock, all filter bypassed, Hpc = 1
 * CTRL_REG1:           1100 0111b:    Low Power Mode 10 Hz Output rate, update 50Hz,
                                       enable x, y, z axis
 * CTRL_REG2:           0000 0000b:    default
 * CTRL_REG3:           0000 0000b:    default
 * CTRL_REG4:           1100 1000b:    Block update, big endian, +/- 2g,
 * CTRL_REG5:           0000 0000b:    default
*****************************************************************/

I2CTWI_transmit4Bytes(ADR_ACC_MODULE, 0xA0, 0xC7, 0x00, 0x00);
// Slave Address; Write Register 0x20  - Data Reg1 – Data Reg 2, Data Reg3
I2CTWI_transmit3Bytes(ADR_ACC_MODULE, 0xA3, 0x88, 0x00);
// Slave Address; Write Register 0x23  - Data Reg4 – Data Reg 5

Slave Address is:  0x32 (PIN2 = High)  or 0x30 (PIN2 = Low)
```

5.2 Reading data from I2C

```
/*****************************************************************/
// I2C ACC Sensor  LIS302DLH
// Read x axis acceleration data high and low Byte and put it into buffer

I2CTWI_readRegisters(ADR_ACC_MODULE, 0x29, &buffer[0], 1);
// Slave Address; Read Register 0x29; buffer; # of Bytes to read
I2CTWI_readRegisters(ADR_ACC_MODULE, 0x28, &buffer[1], 1);
// Slave Address; Read Register 0x28; buffer; # of Bytes to read

The sensor provides much more settings for filtering, Interrupt generation etc. - please read
data sheet for more information.
```