

Sensors Laboratory

Welcome to the sensors laboratory. Goal of this lab is to get you know with basics sensors and and their way of interacting with real world.

Prerequisites

It is expected that you have knowledge from previous laboratories (prototyping electrical circuits using breadboard, creating and uploading code).

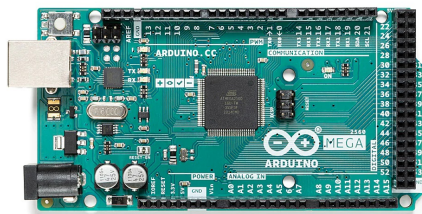
Goals of the laboratory

Goals of this laboratory are set to introduce you to basics of sensors. You will learn:

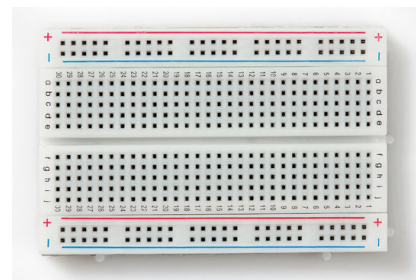
- Measuring distance using HC-SR04 ultrasonic distance sensor
- I2C protocol
- Measuring distance using SRF08 ultrasonic distance sensor
- Work with IMU (Inertial Measurement Unit) and reading its data

Hardware used in this laboratory

For this laboratory you will need following:



(a) Arduino Mega



(b) Breadboard

Figure 1: Core of the laboratory



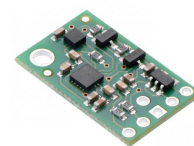
(a) Resistor



(b) HC-SR04



(c) SRF08



(d) MinIMU-9 v5

Figure 2: Electrical parts

- Arduino Mega** - Contains *ATmega2560* processor. Your code will run on it
- Breadboard** - Used for creating electronics circuits
- Resistor** - Used for limiting current for electronics parts (like LED)
- HC-SR04** - Simple ultrasonic distance sensor
- SRF08** - More sophisticated ultrasonic distance sensor
- MinIMU-9 v5** - Inertial measurement unit

Theory

Sensors

Sensors are electrical devices that are able to transform physical phenomenon to electrical output signal. Other electrical devices, such as processors, are then able to read those signals. Different sensors detect different phenomena like light, sound, speed, force, rotation, etc. Also each sensor can have different output signal.

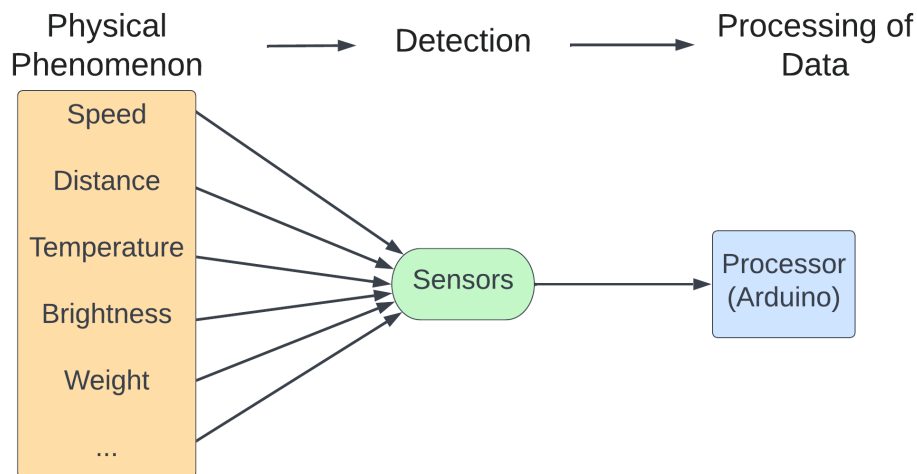


Figure 3: How sensors work

HC-SR04

HC-SR04 is an ultrasonic distance sensor that uses (40kHz) ultrasound waves to detect distance of objects from this sensor. It has 2 parts - *speaker* which creates soundwaves and *microphone* that detects reflected waves. From this basic principle it is possible to measure time it took sound to travel and from that calculate the actual distance.

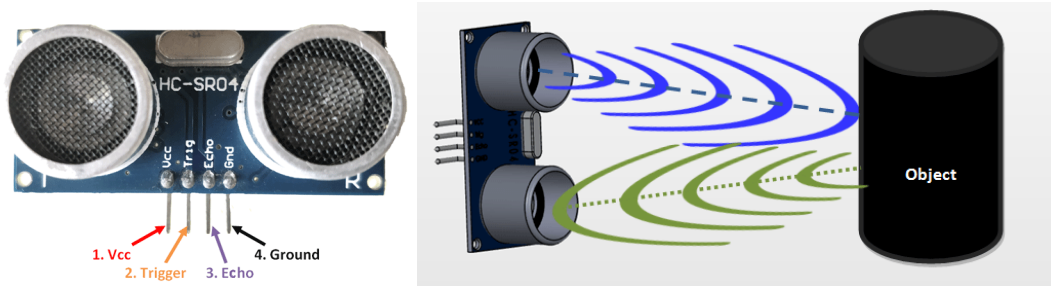


Figure 4: HC-SR04 and its pinout

Operating this sensor is relatively simple. It has 2 communication pins - *Trigger* and *Echo*. In order to take measurement there needs to be 10 microseconds pulse on *Trigger* pin. Then sensor fires its sound waves. Time it took the sound to cover distance between sensor and other object is equal to pulse generated at *Echo* pin. Figure 5 shows this as diagram.

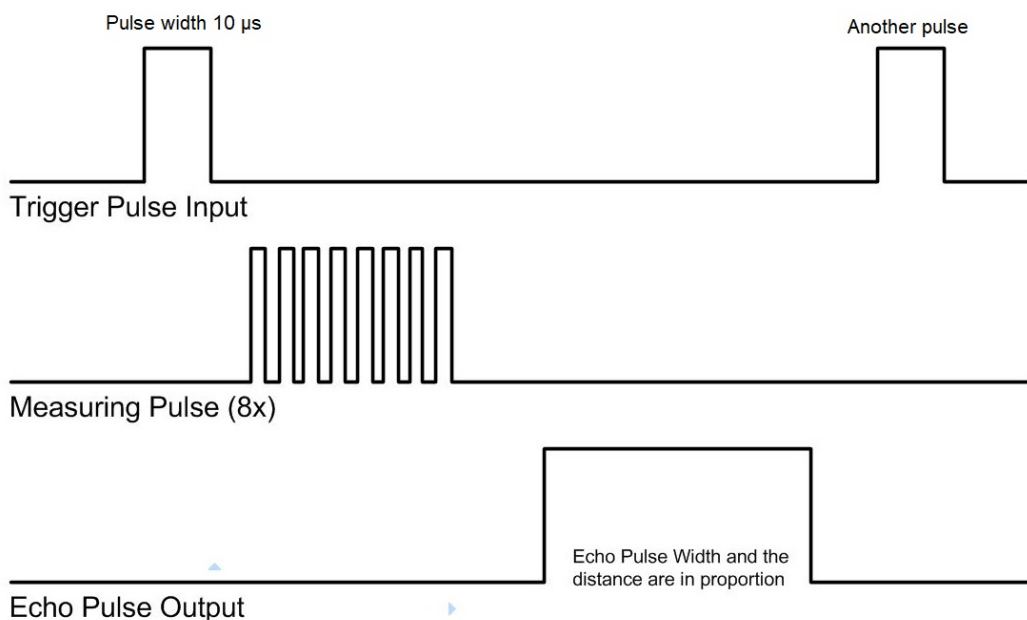


Figure 5: HC-SR04 Ranging diagram

Controlling HC-SR04 using Arduino is straightforward. For creating *Trigger* pulse functions **digitalWrite**(pin, HIGH/LOW) and **delayMicroseconds**(microseconds) can be used. For reading *Echo* pulse there is built in function **pulseIn**(pin,HIGH) which returns length of the pulse in microseconds.

In order to get the measurement of distance in centimeters you need to calculate it using speed of sound which is 343 ms^{-1} . As result is best in centimeters and *Echo* pulse length is measured in microseconds constant $0.034 \text{ cm}\mu\text{s}^{-1}$ is used. Also Echo pulse length is equal to *two way trip* of the sound. Therefore the pulse length needs to be divided by 2.

$$\text{Distance_in_cm} = \text{Echo_pulse_length} * 0.034 / 2$$

Interrupts

Active waiting is way of making requests for a event as often as possible. Although it works for simple events - like reading length of a single pulse. But this method is not viable when there is need to keep track of multiple events. Interrupts solve this problem.

Interrupts are a special way of handling events that need immediate attention. In order to be able to handle interrupts, processor needs to have hardware support for interrupts. When interrupt occurs - for example a change of a value on a pin, processor stops its main process and starts its interrupt service routine. After the routine is finished processor returns to its main process.

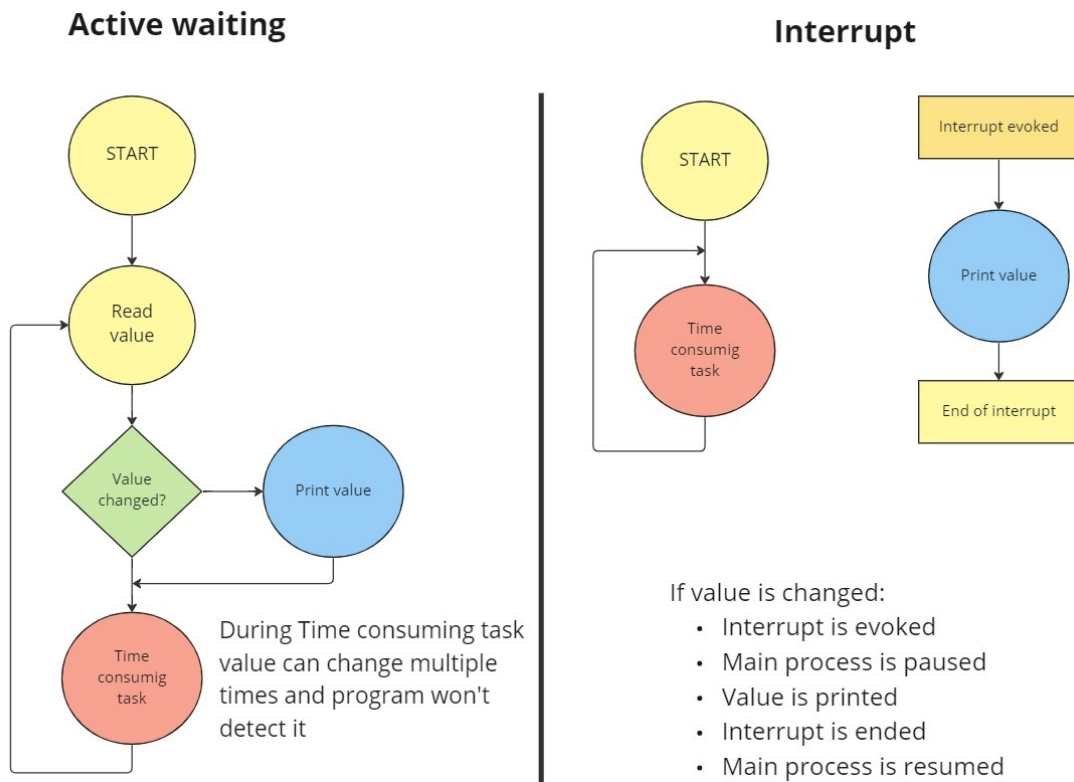


Figure 6: Active waiting vs interrupts

Arduino has hardware support for interrupts. These interrupts are used for detection of a change on a pin. For this there is built-in function `attachInterrupt(digitalPinToInterrupt(pin), ISR, mode)`, where **pin** is number of a pin to which you want to attach interrupt, **ISR** is Interrupt Service Routine - a function which is called when interrupt occurs. This function should be as short as possible. Finally **mode** specifies when interrupt should be evoked - possible values are **RISING/FALLING/CHANGE**.

I2C Protocol

I2C is a synchronous bidirectional bus with architecture master-slave. It has 2 lines **SDA** and **SCL** where **SDA** is data and **SCL** is clock signal. I2C uses open collector which means that on each wire there has to be pull up resistor $2k\Omega$ (some devices might have those resistors built-in).

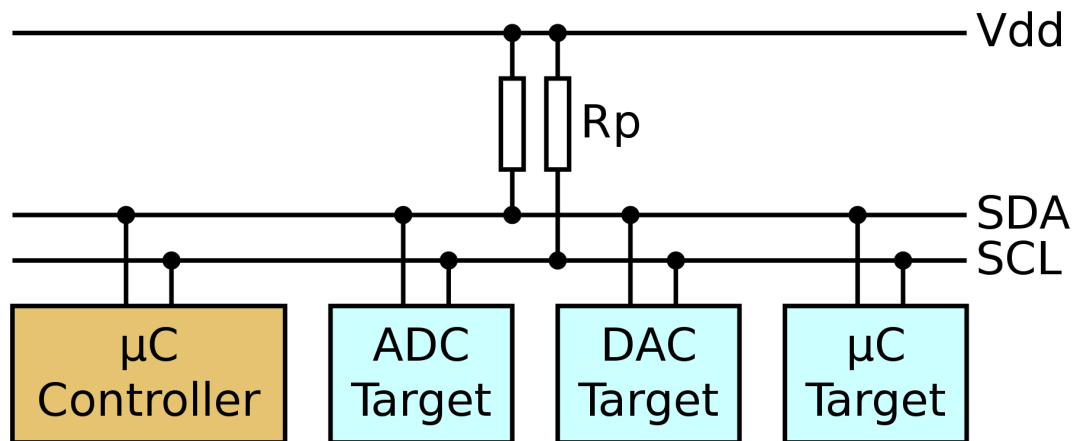


Figure 7: Block scheme of devices on I2C bus

Each device on the bus has its own 7-bit address, which means that on one I2C bus can be up to **127 devices**. Each device has its own internal registers, to which master can write data or request data from it. Each register has its own unique address, but only inside of the device (there can be 2 registers with same address in different devices).

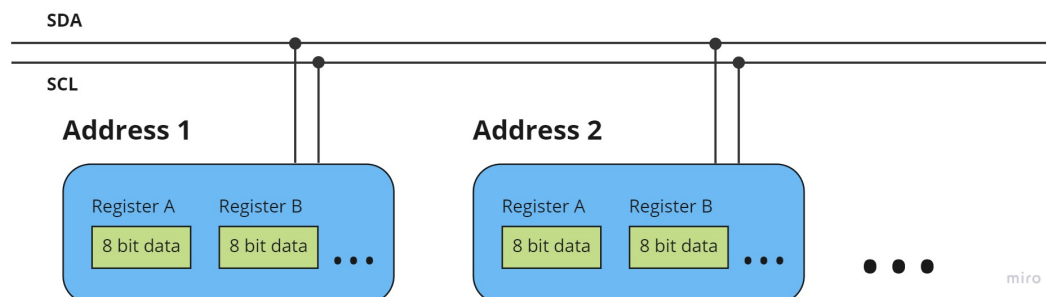


Figure 8: I2C devices

I2C uses 8 bit protocol - meaning that length of one word is 8 bit. After each message there is acknowledge bit. First word is unique device address and the rest of the communication are data.

FIGURE 1: TYPICAL I²C™ WRITE TRANSMISSION (7-BIT ADDRESS)

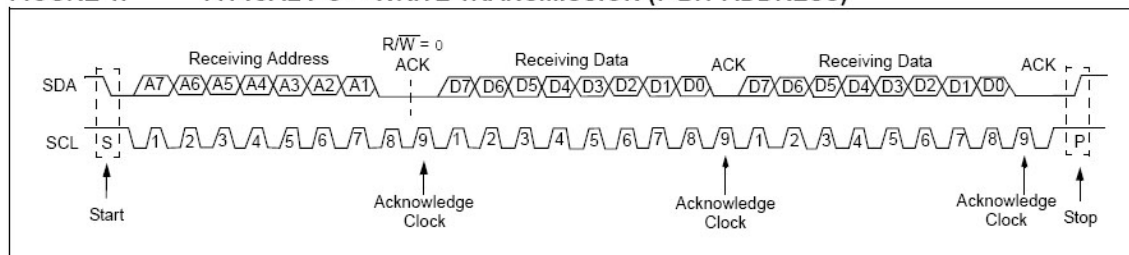


Figure 9: I2C write transmission sample

I2C and Arduino

Arduino has built-in library used for communication via I2C bus called *wire.h*. Functions used for communication are pretty self-explanatory:

beginTransmission()	- Begins I2C communication
endTransmission()	- Ends I2C communication
write()	- Writes byte to device
requestFrom()	- Requests data from device
available()	- Checks if requested data are available
read()	- Reads one byte of available data

Figure 10 shows example of *writing* data as master device to slave device. Firstly device is selected by it's unique address, then internal register is selected by it's address and after that the data itself is transmitted. If multiple bytes of data are sent, they are stored in registers which addresses follow selected register (each byte increments address by 1).

```
Wire.beginTransmission(0x50);    // Start transmission with device with address 0x50

Wire.write(0x42);                // select register 0x42
Wire.write(0x69);                // write value 0x69 to register 0x42

Wire.endTransmission();          // Stop transmission
```

Figure 10: Sample of Arduino I2C write

Figure 11 shows sample of *reading* data as master device from slave device. Firstly device is selected by it's unique address, then internal register is selected by it's address. After that data is requested from slave device. Slave device then sends contents of selected register. If multiple bytes are requested, slave device sends data from registers which addresses follow selected register (each sent byte increments address by 1).

```
// Selecting from which register to read
Wire.beginTransmission(0x50); // Start transmission with device with address 0x50
Wire.write(0x42);             // select register 0x42
Wire.endTransmission();       // stop transmitting

Wire.requestFrom(0x50, 1);    // Request 1 byte from device with address 0x50

if (1 <= Wire.available()) { // If byte was received
|   value = Wire.read();     // Read received byte
}
```

Figure 11: Sample of Arduino I2C read

SRF08

SRF08, same as HC-SR04, is ultrasonic distance sensor. Working principle is the same: sensor fires a sound wave and measures time that it take for the sound wave to reflect back into it's microphone.



Figure 12: SRF08 ultrasonic distance sensor

What is the difference between HC-SR04 and SRF08? While HC-SR04 uses pins *Trigger* and *Echo*, SRF08 is more sophisticated and uses **I2C protocol** for communication. Also SRF08 is able to return measured result in millisecond, centimeters or inches. Furthermore it has multiple advanced settings for tweaking the behaviour of the sensor - for example Maximum range and Analog gain can be set. And as a bonus, SRF08 also contains light sensor.

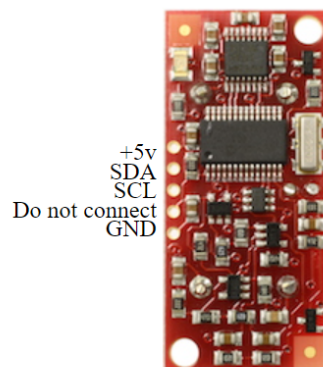


Figure 13: SRF08 Pinout from backside

I2C specifications of SRF08 are as following. Like every I2C device it has its own address. This address can be internally changed.

Address	0x70
---------	------

miro

Figure 14: SRF08 default address

Internal registers of SRF08 store all the necessary data needed for reliable functionality. Firstly there is **Command register**. This register is used for starting ranging session. Session is started by writing command value into this register.

Light sensor register is used for reading output data from light sensor that is present at SRF08.

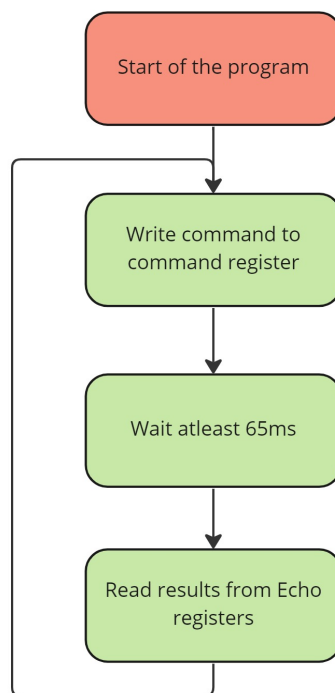
Echo Low Byte and **Echo High byte registers** are output 8-bit registers where the result of measurement is stored. Measured distance can be in *centimeters*, *inches* or *milliseconds* (as can be seen on Figure 17). In order to read values bigger than 255 (which is the maximum value stored in 8-bit register), the value is splitted into 2 registers. This is explained on Figure 18.

Register	Location	Read/Write
Command register	0x00	Write
Light Sensor	0x01	Read
Echo High Byte	0x02	Read
Echo Low Byte	0x03	Read

miro

Figure 15: SRF08 Registers and their addresses

Reading data from SRF08 goes as following: Firstly **command value** needs to be written into **command register**, which will initiate ranging sequence (values are shown on Figure 17). Then the sensor starts its ranging sequence - for the next 65ms sensor will not respond to any commands, as it is performing the ranging. After this time is passed, result is ready in **Echo registers**.



miro

Figure 16: Ranging loop of SRF08

Behaviour of sensor depends on value written into it's command register. There are 3 different values. Each specifies what will be the output unit of the result. It can be *centimeters*, *inches* or *milliseconds*.

Action	Value
Range - Result in inches	0x50
Range - Result in centimetes	0x51
Range - Result in milliseconds	0x52

Figure 17: Values for command register

Output registers are **8 bit** in order for their values to be able to be sent using I2C communication. That means that in one register can be value in range **<0, 255>**. But what happens when measured distance exceeds this range? This is why result is stored in **two 8-bit registers, which gives 16-bit range <0, 65536>**. In order to get this extended value range, values need to be **combined in master device** that requests the data. High byte needs to be **multiplied by 256** (or shifted by 8 bits to the left) and Low byte can stay as it is.

Result					
Echo High Byte			Echo Low Byte		
15	8	7	0

Figure 18: How sensors work

$$Result = HighByte * 256 + Lowbyte$$

IMU

Another sensor used in this lab is IMU - **Inertial Measurement Unit**. It's is combination of accelerometers, gyroscopes and magnetometers. Purpose of IMU is to measure objects acceleration, orientation, angular rates, and other forces.

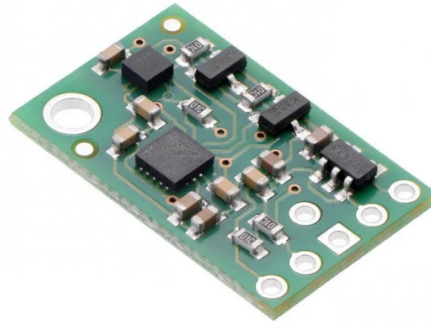


Figure 19: MinIMU-9 v5

IMU used in this lab is **MinIMU-9 v5**. Its small packaging contains 2 modules: *Gyroscope* and *Magnetometer*. MinIMU-9 v5 uses for communication **I2C** bus. Each of the modules has its own I2C address. First module is LSM6DS33 - gyroscope and accelerometer. Second is LIS3MDL - magnetometer. Each of the modules also offers temperature sensor.

Gyroscope address	0x6B
Magnetometer address	0x1E

miro

Figure 20: 2 Modules of MinIMU-9 v5 and their addresses

IMU - Gyroscope

Gyroscopes goal is to measure forces objects specific forces. In this case, gyroscope consists of 2 sensors - one for **linear acceleration** and other one for **angular rate**. This gives the gyroscope to measure objects specific forces along **6 degrees freedom**.

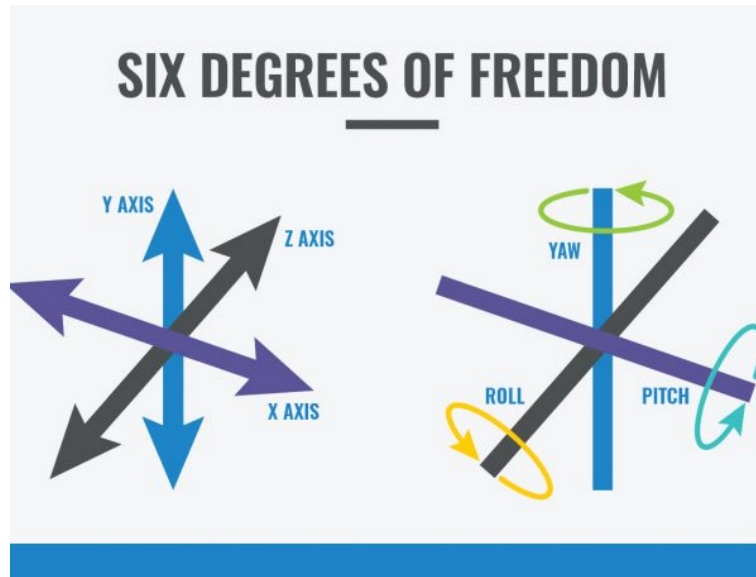


Figure 21: Six degrees of freedom

Gyroscope can measure linear acceleration of object along 3 axes **X, Y, Z** and its angular rate along this axes, called **Pitch, Yaw, Roll**.

Communication with each part of the gyroscope goes as following: Firstly there needs to be written setup value into **control register**. This value says what should be the frequency of sampling and its given range of operation. After value is set, gyroscope will perform measurement as per given frequency and other devices can read its data from it's output registers.

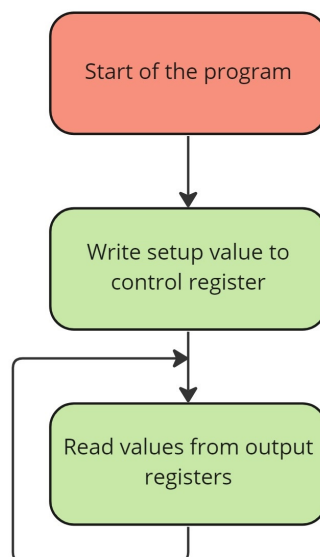


Figure 22: Control loop of IMU

Use values shown at Figure 23. It sets up the frequency of sampling to 833Hz and defines ranges. Range for linear acceleration is $\pm 2g$ and range for angular rate is **2000dps** (degrees per second).

Control register	Location	Read/Write	Desired value
Linear acceleration	0x10	Write	0x74
Angular rate	0x11	Write	0x7C

Figure 23: IMU - gyroscope's control registers and desired values

Gyroscope - Linear acceleration

Linear acceleration sensor measures change in speed of a object along it's axes. Output is along 3 axes **X, Y, Z** and each output is **16 bit signed integer**. As I2C supports only sending 8-bit numbers, the output value is split among two 8-bit registers. Way of combining the numbers is explained at Figure 18. Output range is in this case **<32768, 32767>** which corresponds to range that was selected in setup register, in case of this laboratory it is **<-16g, +16g>**. Unit of **g** is way to measure acceleration an **1g** is equal to **9.81ms⁻¹**.

Measuring loop of linear acceleration is explained at Figure 22. Setup register and it's value for accelerometer is described at Figure 23 and output registers are at Figure 24.

Linear Acceleration register	Location	Read/Write
X - axis low byte	0x28	Read
X - axis high byte	0x29	Read
Y - axis low byte	0x2A	Read
Y - axis high byte	0x2B	Read
Z - axis low byte	0x2C	Read
Z - axis high byte	0x2D	Read

Figure 24: Gyroscope - Linear acceleration output registers

Gyroscope - Angular rate

Angular rate sensor measure changes in rotation of the object along its axes. Output is along 3 axes **X, Y, Z** and rotation along this axes is called **Pitch, Yaw, Roll**.

As I2C supports only sending 8-bit numbers, the output value is split among two 8-bit registers. Way of combining the numbers is explained at Figure 18. Output range is in this case **<32768, 32767>** which corresponds to range that was selected in setup register, in case of this laboratory it is **<-2000dps, +2000dps>**. Unit of **dps** means **degrees per seconds**.

Measuring loop of angular rate is explained at Figure 22. Setup register and it's value for angular rate is described at Figure 23 and output registers are at Figure 25.

Angular Rate register	Location	Read/Write
X - axis low byte	0x22	Read
X - axis high byte	0x23	Read
Y - axis low byte	0x24	Read
Y - axis high byte	0x25	Read
Z - axis low byte	0x26	Read
Z - axis high byte	0x27	Read

Figure 25: Gyroscope - Angular rate output registers

Gyroscope - Temperature sensor

Another extra sensor that MinIMU-9 v5 offers is temperature sensor. There is not much info about this sensor even in documentation. It is mainly used for internal self-correction of other sensors. But it is available for user to read it's measurement. It's output is one *16-bit signed* integer and most importantly, it is not calibrated - meaning that it has **no output unit** and **it's up to user** to find out the ranges and **calibrate the sensor**.

Measuring loop is the same as with accelerometer and linear acceleration. Firstly there needs to be command written into linear acceleration register (to start up the component). After that result can be read from output registers.

Temperature register	Location	Read/Write
Output low byte	0x20	Read
Output high byte	0x21	Read

Figure 26: Output registers of gyroscopes temperature sensor

Magnetometer

Another module that MinIMU-9 v5 offers is magnetometer. Magnetometer is device that measures magnetic field, in this case in 3 axes - **X, Y, Z** . Similarly as gyroscope, it uses I2C for communication. It has it's own address **0x1E** (as it is another device on I2C bus) as could be seen at Figure 20. It has its own command register that needs to be set up before reading from output registers.

Control register	Location	Read/Write	Desired value
Magnetometer	0x22	Write	0x00

Figure 27: Magnetometer's control register with desired value

Output number is one 16-bit number, but as I2C supports only sending 8-bit numbers, the output value is split among two 8-bit registers. Way of combining the numbers is explained at Figure 18. Output range is in that way **<32768, 32767>** which corresponds to range that was selected in setup register. In case of this laboratory it is **<-2G, 2G>**. Unit of **G** is meant for **Gauss**, which is unit for measuring magnetic flux. **10 000 Gauss = 1 Tesla**.


Magnetometer register	Location	Read/Write
X - axis low byte	0x28	Read
X - axis high byte	0x29	Read
Y - axis low byte	0x2A	Read
Y - axis high byte	0x2B	Read
Z - axis low byte	0x2C	Read
Z - axis high byte	0x2D	Read

Figure 28: Output register of magnetometer

Exercises

HC-SR04 - Ranging

Measure distance using HC-SR04 sensor.

1. Open file **HC-SR04_ranging.ino** with Arduino IDE using **File -> Open...**
2. Take a look at **HC-SR04_scheme.png** and create circuit as it is shown on image
3. Follow the instructions inside and write your own code that reads length of pulse and converts milliseconds into centimeters
4. Upload your code by clicking on Upload button  or pressing Ctrl+U
5. Test the sensor by pointing it to different sides and measuring different distances

HC-SR04 - Testing the sensor

Test the abilities and range of the sensor.

1. Take a look at Figure 29.

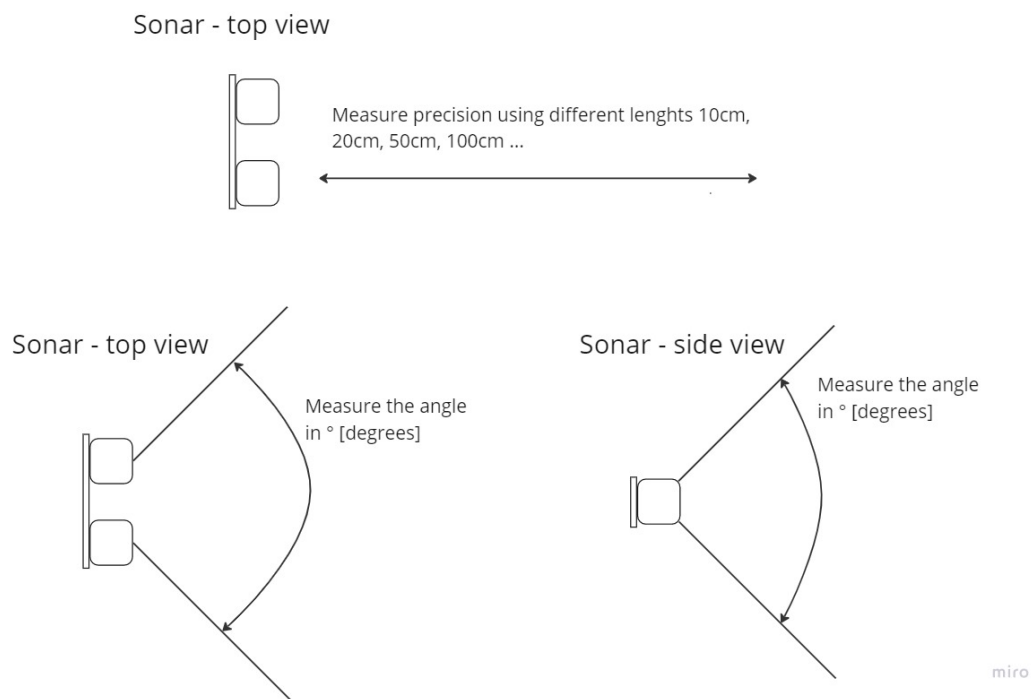



Figure 29: Sonar exercise

2. Test precision of the sensor by placing object in known distance from sensor and observing output

3. Find out what is the sensor's angle of detection.
4. Try detection with different materials. Do all materials behave the same?
5. Discuss your results


HC-SR04 - Interrupt method

Previous implementation of ranging uses active waiting - meaning that processor can't be used for anything else while it performs measurement. Take a look at this implementation that uses interrupts.

1. Open file **HC-SR04_interrupt.ino** with Arduino IDE using **File -> Open...**
2. Take a look at the implementation.
3. Upload your code by clicking on Upload button  or pressing Ctrl+U
4. Test the sensor and see if you can spot difference in behaviour compared to previous method.

HC-SR04 - Measuring speed

Take derivation of measured distance in time to calculate speed of a moving object.

1. Open file **HC-SR04_speedmeter.ino** with Arduino IDE using **File -> Open...**
2. Follow the instructions inside and write your own code that calculates the speed of a moving object in front of the sensor
3. Upload your code by clicking on Upload button  or pressing Ctrl+U
4. Test your code by moving object in front of the sensor

SRF08 - ranging

sem dat aj centimetre aj miliseconds - skladanie registrov Measure distance using SRF08 sensor.

1. Open file **SRF08_ranging.ino** with Arduino IDE using **File -> Open...**
2. Take a look at **SRF08_scheme.png** and create circuit as it is shown on image
3. Implement the control loop by sending command to sensor, waiting at least 65 ms and then reading the result from output register
4. Combine two 8-bit output values into one 16-bit value
5. Look at the results using Serial monitor and test your implementation by pointing sensor to different sides
6. Test the sensor by pointing it to different sides and measuring different distances

SRF08 - Light sensor

SRF08 also offers lights sensor. Write a code that reads data from light sensor.

1. Open file **SRF08_light_sensor.ino**
2. Follow the instructions inside and write your own code that reads data from the light sensor and sends it to PC via Serial communication
3. Upload and test your implementation by pointing the light sensor towards light source and away from it.

MinIMU - measuring linear acceleration

Use MinIMU for measuring linear acceleration.

1. Open file **IMU_lin_acc.ino**
2. Follow the instructions inside and write your own code that sets up the IMU by writing setup value into control register
3. Read data from from linear acceleration output registers and send them to PC via Serial communication
4. Upload and test your implementation moving with the sensor in different direction. (X, Y, Z axes are marked at IMU)
5. By experimenting guess the output value which corresponds into acceleration of 1g (9.81ms⁻¹). Convert output value into unit of g.

MinIMU - Angular rate

Use MinIMU for measuring angular rate.

1. Open file **IMU_ang_rate.ino**
2. Follow the instructions inside and write your own code that sets up the IMU by writing setup value into control register
3. Read data from from angular rate output registers and send them to PC via Serial communication
4. Upload and test your implementation by rotating IMU along different axes. (X, Y, Z axes are marked at IMU)

MinIMU - Angular rate integral

Integrate the output from angular rate to get absolute angle.

1. Open file **IMU_ang_rate_integral.ino**
2. Follow the instructions inside and write your own code that sets up the IMU by writing setup value into control register
3. Read data from from angular rate output registers and send them to PC via Serial communication
4. Upload and test your implementation by rotating IMU along different axes. (X, Y, Z axes are marked at IMU)

MinIMU - Temperature sensor

Use MinIMU to measure relative temperature.

1. Open file **IMU_temperature.ino**
2. Follow the instructions inside and write your own code that reads data from temperature sensor and sends it to PC via Serial communication
3. Upload and test your implementation by touching the sensor (warming it up) and blowing air on it (cooling it down)
4. Bonus assignment: Calibrate the sensor.

MinIMU - Magnetometer

Use MinIMUs magnetometer for measuring magnetic fields around the sensor.

1. Open file **IMU_magnetometer.ino**
2. Follow the instructions inside and write your own code that reads data from the magnetometer and sends it to PC via Serial communication
3. Upload and test your implementation by rotating the sensor. Sensor can even detect earth's magnetic field. Alternatively you can put to magnetometer's close proximity item made out of iron, or speaker from a phone
4. Your assignment: Find NORTH