Educational tutorial for Arduino and Trilobot

Jan Beran

Faculty of information technology Brno University of Technology
Božetěchova 1/2. 612 66 Brno – Královo Pole

xberan43@stud.fit.vutbr.cz

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ FAKULTA INFORMAČNÍCH TECHNOLOGIÍ

May 25, 2020

Overview

Introduction

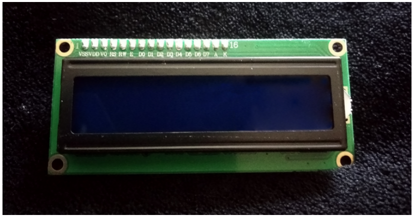
Trilobot

- Originally: robot for academical purposes
- Rebuilt several–times
- Original look in the picture
- Current version based on Arduino is in front of you
- It is fully–assembled and there is no need to connect anything physically
- Fun fact: it used to ride in the opposite direction :)

Trilobot components

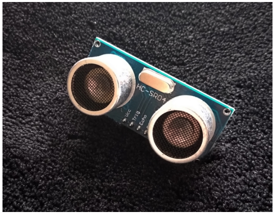
Displej

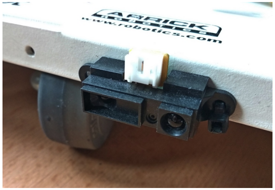
- Standard 16x2 LCD display connected via I^{2}C
- Can be controlled via `display` object or via `LiquidCrystalI2C.h` library



Distance sensors

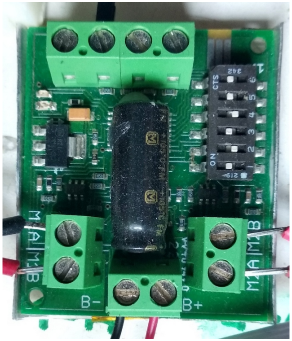
- Ultrasound HC–SR04 a SRF–08, infrared Sharp
- Every sensor fits on something else
- They can be controlled via `hcsr04`, `srf08` and `sharp`

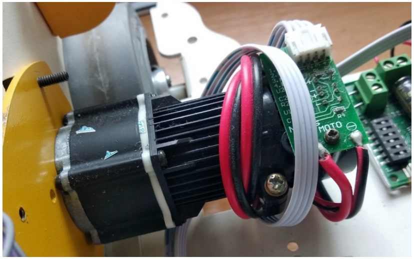




Motors

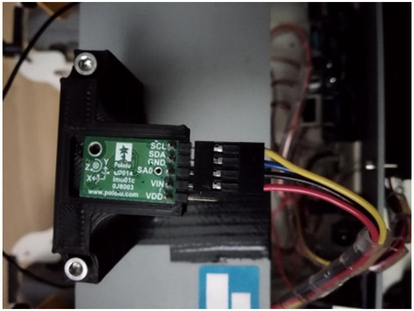
- Two Faulhaber motors controlled by the Sabertooth board are used to drive
- Communication between Arduino and Sabertooth takes place via Serial line `Serial1`
- Feedback is realized via two rotary encoders
- Motors are controlled via `motors` object in the code





IMU – gyroscope, accelerometer, magnetometer

- 3–in–1 sensor pack:
  - Gyroscope for angular velocity
  - Accelerometer for acceleration
  - Magnetometer for measuring of magnetic field intensity
- The data from IMU serves for positioning via AHRS (Attitude and heading reference system)
- IMU has to be above Trilobot to prevent magnetometer from interfering with Trilobot metal body
- There are three object to control the sensors: `gyro`, `accel` and `mag`



Before start

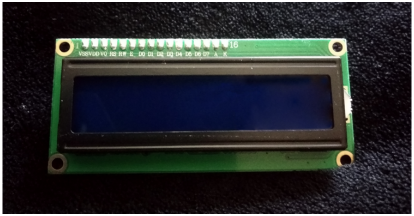
Notes

- Number in `0x00` format:
    - Number in hexadecimal system
    - It is used widely in IT, but it is not mandatory
    - If you want to use decimal numbers, you surely can! But don’t forget to convert them (converters on Google)!
- Functions to complete has **czech** names to help czech speaking people to recognize them.

Display

Display 16x2

- 16 chars, 2 rows
- Connected via I^{2}C
- LCD is relatively slow – it cannot be redrawn 1000x in a second
- Controls:
  - `display.h` library and `Display` class



display.h

- Very simple wrapper* to use the display and print data output from sensors etc.
- Instance of `Display` class (default name is `display` in the code) has three methods:

```
 #include <display.h>
 ...
 /** Methods clear display and print data*/
 display->print_first_line(to_print);
 display->print_second_line(to_print);
 display->clear(); /* Clears the whole display */
}
```

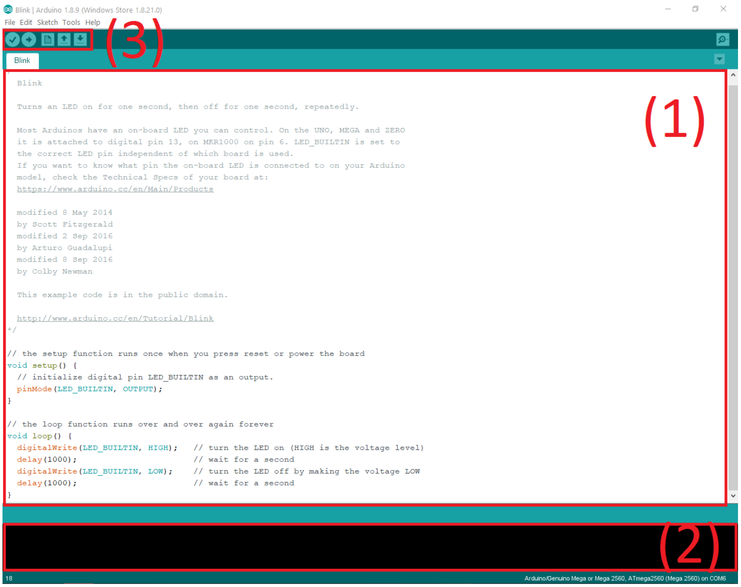
*wrapper = class, which wraps another class. In this case, it make it easier to use display.

Arduino IDE and Trilobot connection

Arduino IDE

- Development environment for writing and loading code into Arduino
- Simple – no code completition etc.
- For programming Arduino, it is needed to select a proper port:
    - Go to *Tools/Port* and select right COM port
    - If we don't know which one, we open COM port list, disconnect and reconnect the Arduino. The port which disappears and appears again is the right one.
- Arduino IDE description (picture in the next slide):
    - (1): Main part, code goes here
    - (2): Console, where useful information is shown (during board programming etc.)
    - (3): Buttons for (from left): Check, programming into board, New File, Open and Save

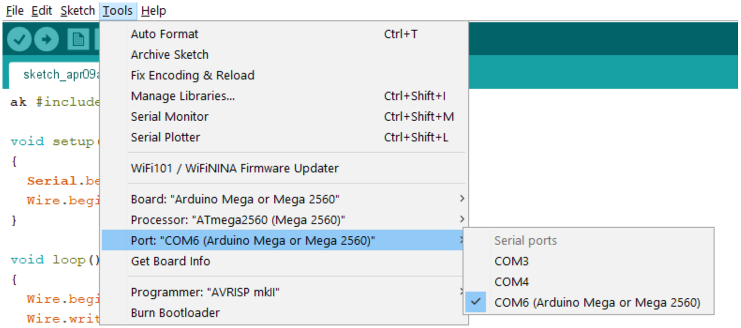
Arduino IDE


Blink | Arduino 1.8.9 (Windows Store 1.8.21.0)
File Edit Sketch Tools Help
(3)
Blink
(1)
(2)
18
Arduino/Genuino Mega or Mega 2560, ATmega2560 (Mega 2560) on COM6

```
/*
  Blink

  Turns an LED on for one second, then off for one second, repeatedly.

  Most Arduinos have an on-board LED you can control. On the UNO, MEGA and ZERO
  it is attached to digital pin 13, on MKR1000 on pin 6. LED_BUILTIN is set to
  the correct LED pin independent of which board is used.
  If you want to know what pin the on-board LED is connected to on your Arduino
  model, check the Technical Specs of your board at:
  https://www.arduino.cc/en/Main/Products

  modified 8 May 2014
  by Scott Fitzgerald
  modified 2 Sep 2016
  by Arturo Guadalupi
  modified 8 Sep 2016
  by Colby Newman

  This example code is in the public domain.

  http://www.arduino.cc/en/Tutorial/Blink
*/

// the setup function runs once when you press reset or power the board
void setup() {
  // initialize digital pin LED_BUILTIN as an output.
  pinMode(LED_BUILTIN, OUTPUT);
}

// the loop function runs over and over again forever
void loop() {
  digitalWrite(LED_BUILTIN, HIGH);   // turn the LED on (HIGH is the voltage level)
  delay(1000);                       // wait for a second
  digitalWrite(LED_BUILTIN, LOW);    // turn the LED off by making the voltage LOW
  delay(1000);                       // wait for a second
}
```

Trilobot connection

- Via USB cable
- After successful connection, Trilobot’s display should turn on.
- Go to *Tools/Port* and select proper port – if Arduino is the original one, there should be its type name next to the COM port



```
File  Edit  Sketch  Tools  Help

Auto Format                                          Ctrl+T
Archive Sketch
Fix Encoding & Reload
Manage Libraries...                                  Ctrl+Shift+I
Serial Monitor                                       Ctrl+Shift+M
Serial Plotter                                       Ctrl+Shift+L
WiFi101 / WiFiNINA Firmware Updater
Board: "Arduino Mega or Mega 2560"
Processor: "ATmega2560 (Mega 2560)"
Port: "COM6 (Arduino Mega or Mega 2560)"       Serial ports
Get Board Info                                 COM3
                                               COM4
Programmer: "AVRISP mkII"                    ✓ COM6 (Arduino Mega or Mega 2560)
Burn Bootloader
```

```
sketch_apr09a
ak #include
void setup(
{
  Serial.be
  Wire.begi
}
void loop()
{
  Wire.begi
  Wire.writ
```

Connection test

- Open new sketch using *File/New* or keyboard shortcut ctrl + N. Sketch is a Arduino–specific file with .ino suffix
- After Trilobot is connected, copy the following code into `loop()` and load it to Arduino using the green arrow button in the left top part of Arduino IDE:

```
void loop(){
    /*pin 13 into HIGH - LED turns on*/
    digitalWrite(13, HIGH);
    /*500 ms wait */
    delay(500);
    /*Change the LED state */
    digitalWrite(13, LOW);
    /*Wait again*/
    delay(500);
}
```

- If everything is all right, LED next to the display should start blinking

Introduction to Arduino programming and OOP

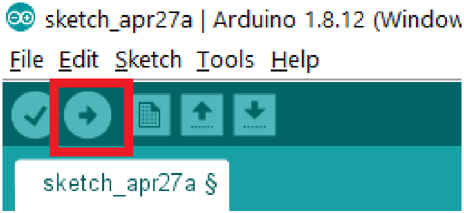
Introduction to Arduino programming

- Language used to write programs for Arduino is named Wiring – it is fully compatible with C++
- Always, there are two functions in the code:
    - `setup()`: This code is called only once; after Arduino starts or after reset
    - `loop()`: This code is executed in an infinite loop

```
void setup()
{
/* Put your setup code here, to run once */
}
void loop()
{
/* Put your main code here, to run repeatedly */
}
```

Programming Arduino

- By clicking the green arrow, the code will translate and load into Arduino or Arduino IDE shows an error message in the console
- Arduino IDE wants to save your sketch before uploading. It is not mandatory, but recommended.
- If syntax error appears, Arduino IDE shows, where


sketch_apr27a | Arduino 1.8.12 (Window
File Edit Sketch Tools Help
sketch_apr27a §

Introduction to OOP

- OOP is a way how to write your code based on *objects*
- *Objects* in the code correspond to the real world objects in two ways:
  - Characteristics – atributes: cat's color or I^{2}C address of the specific sensor
  - Abilities – methods: cat can meow and sensor can collect data
- *Objects* have its templates – classes
  - For example: The `Cat` class is a template for an instance if a cat named `rose`

Using objects and calling methods

- Everything will be shown on `rose` object (imaginary cat) and `display` object – really implemented in the code and can be used.
- If we want to call a methods on an object, we use following syntax:

```
object_name->methos_name(parameters);
```

- If we want our cat `rose` to meow, we just type:

```
rose->meow();
```

- And for printing a line on the display we type:

```
display->print_first_line("Hello world!");
```

- Briefly: Method call is similar to a function call. We only type object name and arrow−> before the method name.

Tutorial

Photosensor

Light intensity measurement

Physical principle:

- Photosensor is unenriched semiconductor –> it conducts the current very poorly (almost not at all)
- However, light/photons gives the electrons enough energy in the valence layer –> pair electron–hole are created
- These pairs can conduct current very well
- The more the light shines, the more pairs appears and the more current can be conducted

Photosensor

- Mostly a little sensor, distinguishable by its characteristics curves
- In our case, it is a part of SRF–08 sensor

Photosensor: Tasks

- Test if Trilobot and Arduino IDE are connected correctly
- Complete following into the function template `SRF08::zmer_svetlo()` (placed in `srf08.cpp`):
  - Register number, in which SRF–08 stores value from the photosensor (see documentation, page 4, address is in *Location* column)
  - Return the value that was read
  - Print the value on on the display (using `display->print_first_line()`)
  - (optional) Turn the LED on when it gets too dark

Photosensor: Template from `srf08.cpp`

```
byte SRF08::zmer_svetlo()
{   this->set_measurement();
    /*We request the data from register PHOTO. REGISTER*/
    Wire.beginTransmission(SRF08_ADDRESS);
    /* 0x00 is only placeholder to make code compilable */
    Wire.write(0x00/*TODO PHOTOSENSOR REGISTER*/);
    Wire.endTransmission();

    /*Request for one byte...*/
    Wire.requestFrom(SRF08_ADDRESS, 1);
    /*...and wait until is available*/
    while(Wire.available() < 0);
    byte light_intensity  = Wire.read();
    return light_intensity ;
}
```

Photosensor: Instructions for `SRF08::zmer_svetlo()`

- Address of the register with light intensity information from photosensor is **`0x01`**
- Displaying the values is done via:

```
display->print_first_line(data);
```

- Now, you just need to retype this to the function itself.

Photosensor: Solution of `SRF08::zmer_svetlo()`

```
byte SRF08::zmer_svetlo()
{   this->set_measurement();
    /*We request the data from register 0x01*/
    Wire.beginTransmission(SRF08_ADDRESS);
    Wire.write(0x01);
    Wire.endTransmission();

    /*Request for one byte...*/
    Wire.requestFrom(SRF08_ADDRESS, 1);
    /*...and wait until is available*/
    while(Wire.available() < 0);

    byte light_intensity = Wire.read();
    return light_intensity;
}
/* V loop()*/
display->print_first_line(srf08->photosensor());
```

Distance measurement

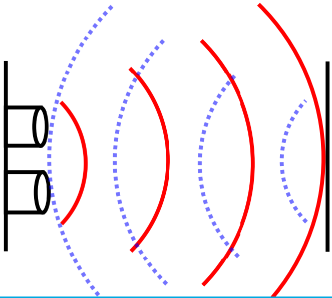
Distance measurement

In this chapter we will explain:

- How can we measure the distance
- Measurement using HC−SR04, SRF−08 and Sharp−GP2D120 sensors

How can we measure distance

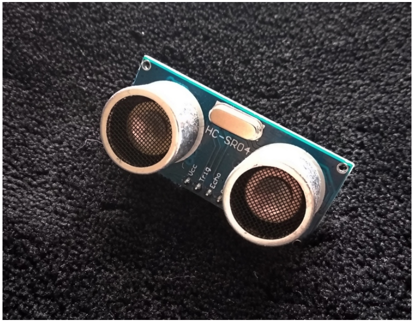
- With ruler :) – robots cannot do that
- If the robot is movable, we can use motor/wheel spin. Not very useful or accurate – wheels can slip etc.
- Using movement sensors (accelerometer...) – measurement of distance traveled, ot the distance from a barrier
- Wirelessly: using some kind of wave (light, sound...)
  - This is what we will try with all three sensors
  - The principle can be seen in the picture: red waves are sent by the sensor, blue waves are received. From the time between transmitting and receiving we can compute the distance (if we know the speed of the wave).



HC_SR04

HC_SR04: Introduction

- Ultrasound distance measurement sensor
- 4 wires: 2x power, `ECHO_PIN` and `TRIGGER_PIN` (you can use these two names with working with these pins in the code)
- Documentation



HC_SR04: Principle

- When the `TRIGEER_PIN` is in HIGH state for at least 10 microseconds, sensor start measurement
- `ECHO_PIN` is in HIGH state until the wave comes back
- After the reflected wave comes back, `ECHO_PIN` goes back to LOW
- Value in microseconds has to be converted to distance using the speed of sound: **340 m/s or 0.0343cm/us**

HC_SR04: Tasks

- Complete the function template named `HCSR04::zmer_vzdalenost()`:
    - Set the `TRIGGER_PIN` to HIGH for 10 microseconds (Use the `delayMicroseconds()` function) Sensor will send the wave and wait for it to come back
    - Listen on `ECHO_PIN` and get the duration of the pulse in microseconds. We will need this function: `pulseIn(pinName, state HIGH or LOW)`
      Once the wave comes back, sensor ends the pulse (HIGH state on `ECHO_PIN`)
    - Convert microseconds to centimeters using the speed of sound 340 m/s (or 0.0343cm/us, which should be more useful)

HC_SR04: Template

```
long HCSR04::zmer_vzdalenost()
{/* Set LOW to be sure */
    digitalWrite(TRIGGER_PIN, LOW);
    delayMicroseconds(2);
    /* TODO: Set TRIGGER_PIN to HIGH HIGH for 10
     * microseconds, then return it to LOW*/

     /*TODO: using pulseIn(pin, state) find out, how
      * long will be ECHO_PIN in HIGH  a convert
      * time to distance */
    long return_value = 0;
    return return_value;
}
```

HC_SR04: Guide

- We set the TRIGGER_PIN to HIGH for 10 microseconds using

```
    digitalWrite(TRIGGER_PIN, HIGH);
    delayMicroseconds(10);
    digitalWrite(TRIGGER_PIN, LOW);
```

- Listening on ECHO_PIN is done by pulseIn:

```
    long echo = pulseIn(ECHO_PIN, HIGH);
```

- Conversion between time and distance when speed is known is simple :) (distance = speed $*$ time)

HC_SR04: Solution

```
long HCSR04::zmer_vzdalenost()
{/* Set to LOW for a while to be sure */
    digitalWrite(TRIGGER_PIN, LOW);
    delayMicroseconds(2);

    digitalWrite(TRIGGER_PIN, HIGH);
    delayMicroseconds(10);
    digitalWrite(TRIGGER_PIN, LOW);

    long echo = pulseIn(ECHO_PIN, HIGH);
    long return_value = echo * 0.0343;
    return return_value;
}
```

We can write the returned value to the display in the `loop()` function. Is is done by `display->print_first_line(what wee want to write)`.

SRF–08

SRF–08: Introduction

- The same principle as with HC_SR04
- Communication via I^{2}C bus – everything about it does the template for us, so we do not need to use it directly
- Documentation
- In addition to distance measurement, is can also measure light intensity
- It has way more complicated setting, however, we will use only the basics

SRF–08: Principle

- We will choose the unit we want:
  - 0x50: inches
  - 0x51: centimetres
  - 0x52: microseconds
- Wait until the measurement is completed. According to the documentation, 60 ms should be enough, but sometimes, it is not true, so we will use 100 ms.
- Request the measured value via I^{2}C :
  - The values is divided into two registers (addresses 0x02 and 0x03)
  - It is divided into the high byte and the low byte – we have to join them

SRF–08: Tasks

- Complete the `SRF08::zmer_vzdalenost()` function:
    - Send the right value to register `0`. We will choose the right one from the previous slide
    - Wait at least `60` ms, but `100` is safer (we don't mind time efficiency)
    - Join values from registers `0x02` a `0x03` (high and low byte)
        - It is not simple and we will discuss it. And dont't worry, there will be the solution as well in the next few slides.
    - Return the value
    - We can write the value to the dispaly again

SRF–08: Template

```
int SRF08::zmer_vzdalenost(){
  /*Communication start*/
  Wire.beginTransmission(SRF08_ADDRESS);
  /*Where we want to write...*/
  Wire.write(REG_CMD); /* expands into 0x00 */
  /*...and what we want to write.*/
  Wire.write(/*TODO unit*/0);
  /*End the communication in the end */
  Wire.endTransmission();
  /* TODO: wait 100 ms*/
  /* We need 2 bytes starting at 0x02*/
  Wire.beginTransmission(SRF08_ADDRESS);
  Wire.write(0x02);
  Wire.endTransmission();
  Wire.requestFrom(SRF08_ADDRESS, 2);
  while(Wire.available() < 0);
  byte high =  Wire.read();
  byte low =  Wire.read();
  /* TODO join the data and return them */
  uint16_t return_value = 0;
  return return_value;}
```

SRF–08: Guide

- We can choose the right unit, but centimeters (`0x51`) are the best
- Waiting is a piece of cake – `delay()` is what we need
- Joining two bytes into one number is a little bit more difficult. There is the solution and explanation:

```
uint16_t number = (uint16_t)(high << 8)+low;
```

$<<$ operator is something like a *shifter*, which shifts the number into the left by x positions. So 111 $<<$ 3 will be 111000 (empty positions are filled by 0). And 101 $<<$ 3 + 101 = 101000+101 = 101101 – absolutely same principle as we have used above.

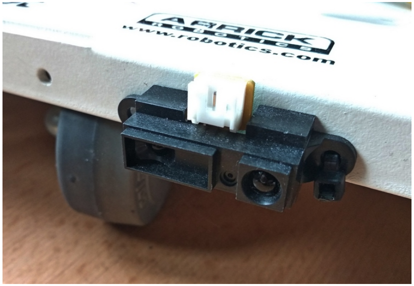
SRF–08: Solution

```
int SRF08::zmer_vzdalenost()
{
  Wire.beginTransmission(SRF08_ADDRESS);
  Wire.write(REG_CMD); /* expands into 0x00 */
  Wire.write(0x51); /* cm */
  Wire.endTransmission();
  delay(100); /* wait */
  Wire.beginTransmission(SRF08_ADDRESS);
  Wire.write(0x02);
  Wire.endTransmission();

  Wire.requestFrom(SRF08_ADDRESS, 2);
  while(Wire.available() < 0);
  byte high =  Wire.read();
  byte low =  Wire.read();

  uint16_t return_value = (uint16_t)(high << 8)+low;
  return return_value;
}
```

Sharp-GP2D120

Sharp–GP2D120: Introduction

- This sensor uses infrared light instead of sound
- Works only for 3 cm and more
- Very simple: the distance is derived from the voltage on the data pin
- There is no linear relationship between the voltage and distance. In other words, it is not so simple to convert voltage to the distance. However, it is solved by the program itself in this case.
- Documentation



Sharp–GP2D120: Principle

- Very shortly after start the sensor starts measuring
- On the data pin (`SHARP_PIN` macro), voltage can be measured
- (For those interested) We can compute the distance from the graph in documentation.
  - Analytically: We find a function which suits the graph the most
  - Numerically: the original graph from the documentation will be divided into small intervals. these intervals are replaced by lines.
  - For our needs, there is function called `aproximate()`, using numerical approximation – the function is replaced by several lines.

Sharp–GP2D120: Tasks

- Complete the function `Sharp::zmer_vzdalenost()`, located in `sharp.cpp`:
  - Read the value on the data pin (`SHARP_PIN` is the pin's name, you will need `analogRead()` function)
  - Convert the value from interval `0–1024` to interval `0–5`
    - Because function `analogRead()` returns value in interval `0–1024` and the graph in documentation is for values `0–5`, we have to convert one value to another. And converting the measured value to voltage is much simpler.
  - Using function `aproximate()`, get the distance in cm
  - We can write the values to the dipsplay once again

Sharp–GP2D120: Template

```
float Sharp::zmer_vzdalenost()
{
/*TODO:
 * read the value on SHARP_PIN
 * convert the value from 0-1024 to 0-5
 * using this->aproximate() get the value in cm
  * and return it.  */
  float aproximated_value = 0;
  return aproximated_value;
}
```

Sharp–GP2D120: Guide

- Read the value on `SHARP_PIN` is simple:

```
int value = analogRead(SHARP_PIN);
```

- When converting from range 0–1024 to 0–5, we have to think a bit, but it is not difficult in general:
  - `value/1024` gives us number from 0 to 1
  - YMultiplications by 5 gives us number from 0 to 5

```
float normalized_value = 0;
normalized_value = (value/1024)*DEFAULT_VOLTAGE;
```

- And calling a function is just... Calling a function :)

```
aproximated_value = this->aproximate(normalized_value);
```

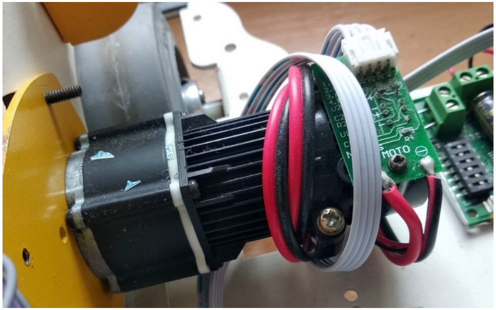
Sharp–GP2D120: Solution

```
float Sharp::zmer_vzdalenost()
{
  int value = analogRead(SHARP_PIN);

  /*ANALOG_MAX = 1024. DEFAULT_VOLTAGE = 5*/
  float normalized_value = 0;
  normalized_value = (value/ANALOG_MAX)*DEFAULT_VOLTAGE;

  float aproximated_value = 0;
  aproximated_value = this->aproximate(normalized_value);
  return aproximated_value;
}
```

We’re moving

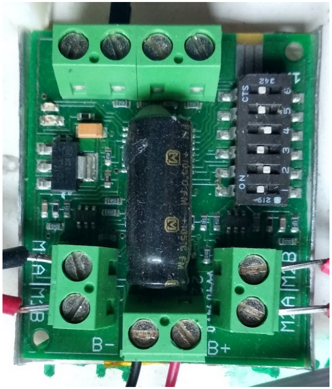
We’re moving

- Two Faulhaber 16002 motors with encoders (we will tell what is encoder later)
- Motors are controlled by Sabertooth 2x5 (board specially designed for controlling motors)
- Communication with Sabertooth is via Serial line, but only in one direction (Arduino –> Sabertooth) only one wire needed (and one function: `Serial1.write(value)`.
- Encoders provides feedback on movement.



Motor controlling with Sabertooth 2x5

- Sabertooth works in Simplified Serial mode
    - Principle: two bytes (one for each wheel) are transferred via Serial line. The number provides the information what motor is controlled and how fast (and in hat direction) it should move.
- Specific values on the next slide
- Communication is extremely simple and fully sufficient for Trilobot.



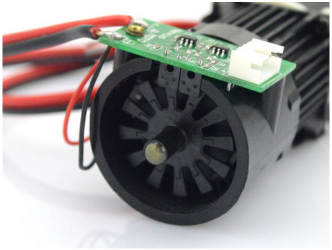
Basic values to control the motors

- 0 is both motor stop
- 1–127 controls left motor:
    - 1 is full backward
    - 64 is left motor stop
    - 127 is full forward
- 128–255 controls right motor (these are only number for left motor + 127):
    - 128 is full backward
    - 192 is right motor stop
    - 255 is full forward

```
/* Example: */
Serial1.write(0x0); /* Both motors stop */
...
/* Both motors full forward - we can use dec as well*/
Serial1.write(0x7F); /* = 127 dec*/
Serial1.write(0xFF); /* = 255 dec*/
```

Encoders

- Encoders gives us a feedback about how much the wheel rotated.
- For one full circle (360 degrees), encoder makes 768 transitions from LOW to HIGH, if we count these changes, we will find out, how much the motor rotated. And, if we know the wheel diameter, we can compute how far we have moved.



Tasks

- Movement straight
- Turning

Movement straight: Tasks

- One advantage, compared to turning: the value sent for both motors is the same (only shifted by 127 for one of them to actually control both motors).
- Principle*:
  - **Inputs**:
    - Desired distance travelled in cm
    - Speed in interval <-100,100>, negative values is for backward movement
  - We compute what number we have to send to Sabertooth (`driving_byte` variable). It is done by `Motors->get_speed_from_percentage()` function
  - We send the bytes via Serial line `Serial1`
  - We wait until Trilobot ends its move. you can see an infinite loop in the function template, which is not completely infinite
  - We stop the motors

Note: We are going to use interrupts. If you are interested, look at Google or, if you can speak Czech, look at the tutorial text to the advanced section of this chapter. However, you don't need to know it to pass the tutorial.

Movement straight: Template

```
void Motors::jed_rovne(int distance, int speed)
{
  int steps_to_go = (int)(distance/WHEEL_CIRCUIT*
  STEPS_ONE_CHANNEL);
  byte driving_byte;
  driving_byte = Motors::get_speed_from_percentage(speed);
  attach_interrupts();
  /*TODO send driving byte to both motors*/
  /*This cycle will be explained*/
  while(1)
  {
    if(steps_left == steps_to_go ||
    steps_right == steps_to_go)
      break;
  }
  /*TODO stop motors*/
  detach_interrupts();
}
```

How can we compute the steps?

- We need two values: the number of steps per one full rotation of the wheel and the wheel diameter
- There are macros for both the values: `STEPS_ONE_CHANNEL` and `WHEEL_CIRCUIT`
- The exact computation is only a simple mathematics (if x steps correspond to one distance, then how many steps correspond to another distance?) :)

Infinite loop? No, interrupts!

- There is something like an infinite loop in the code
- Actually, we can break the cycle and jump out thanks to *the interrupts*
- What is an interrupt? Where can be used?
    - Imagine it as a teacher lecturing a class. Once in a while, some of the student raises their hand and asks a question. The teacher interrupts (see the word? :)) their lecture, answers the question and continues.
- In other words, we will not get caught in that while cycle :).

Movement straight: Guide

- Sending via Serial line is simple (stopping both motors is done by sending a 0):

```
Serial1.write(driving_byte);
Serial1.write(driving_byte+127);
...
Serial1.write(STOP_BYTE); /* = 0x00*/
```

Movement straight: Solution

```
void Motors::jed_rovne(int distance, int speed)
{
 int steps_to_go = (int)(distance/
                   WHEEL_CIRCUIT*STEPS_ONE_CHANNEL);
  byte driving_byte;
  driving_byte = Motors::get_speed_from_percentage(speed);
  attach_interrupts();
  Serial1.write(driving_byte);
  Serial1.write(driving_byte+127);
  while(1)
  {
    if(steps_left == steps_to_go ||
    steps_right == steps_to_go)
      break;
  }
  Serial1.write(STOP_BYTE); /* = 0x00*/
  detach_interrupts();
}
```

Turning

Turning: Introduction

- Turning is not very different from moving straight – the functions will be similar
- Turning can be done in many ways:
  - The simplest: the motor on the side we turn stops, the second one moves forward
  - In place: the motor on the side we turn moves backward and the other one forward
  - Like a car: robot makes a circuit – both motors are moving forward, but with different speeds...
- We will choose the first one – the simplest one

Turning: Tasks

- Turning function principle:
  - **Inputs**:
    - The angle in degrees from interval $< -180, 180 >$, where the positive angle is for turning right and the negative one is for turning left
    - Speed: this time only from `0` to `100`
  - We send the speed information to Sabertooth – we choose right or left motor depending on the direction of the turn
  - We stop the motor

Turning: Template in `motors.h`

```
void Motors:zatoc(int angle, int speed)
{
  int steps_to_go = (int) (((2 * TURN_RADIUS * PI
  * angle)/ 360) / WHEEL_CIRCUIT * STEPS_ONE_CHANNEL);
  attach_interrupts();
  byte driving_byte;
  driving_byte = Motors::get_speed_from_percentage(speed);
 /*TODO turning depending on the sign of the
  * angle parameter*/
  while(1)
  {
    if(steps_left == steps_to_go ||
      steps_right == steps_to_go)
      break;
  }
  /*TODO stop the motor */
  detach_interrupts();
}
```

Turning: Guide

- We can send data via Serial line as well:

```
Serial1.write(driving_byte);
...
Serial1.write(STOP_BYTE);
```

Turning: Solution

```
void Motors::zatoc(int angle, int speed)
{
  int steps_to_go = (int) (((2 * TURN_RADIUS * PI * angle)
                / 360) / WHEEL_CIRCUIT * STEPS_ONE_CHANNEL);
  attach_interrupts();
  byte driving_byte;
  driving_byte = Motors::get_speed_from_percentage(speed);
  Serial1.write(driving_byte);

  while(1)
  {
    if(steps_left == steps_to_go ||
      steps_right == steps_to_go)
      break;
  }
  Serial1.write(STOP_BYTE);
  detach_interrupts();
}
```

The End