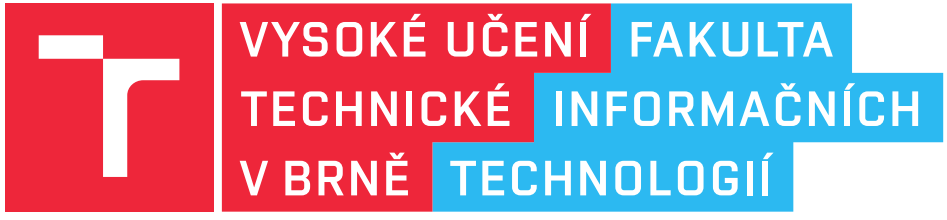
Tutoriál pro robota Trilobot

Jan Beran

Fakulta informačních technologií Vysokého učení technického v Brně
Božetěchova 1/2. 612 66 Brno – Královo Pole

xberan43@stud.fit.vutbr.cz



May 25, 2020

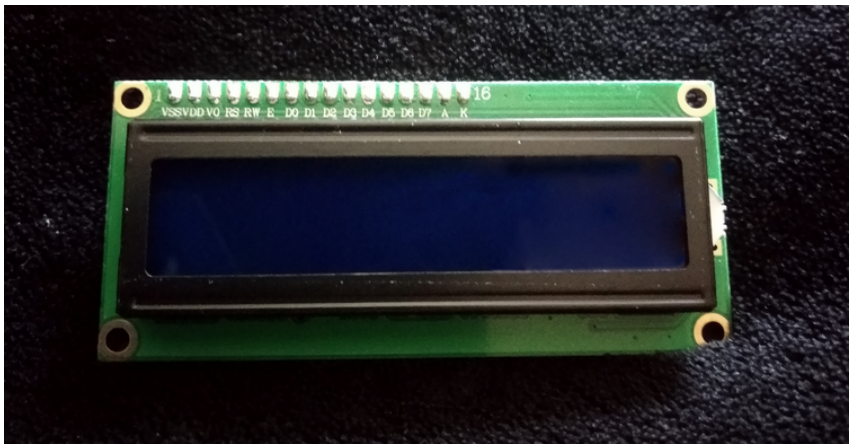
Overview

Úvod

- Původně robot pro akademický vývoj
- Několikrát přestavován
- Na obrázku jeho původní podoba
- Současná verze založená na Arduinu leží před vámi
- Pro potřeby tutoriálu je sestavený a není nutné cokoli fyzicky zapojovat
- Fun fact: původně jezdil opačně :)

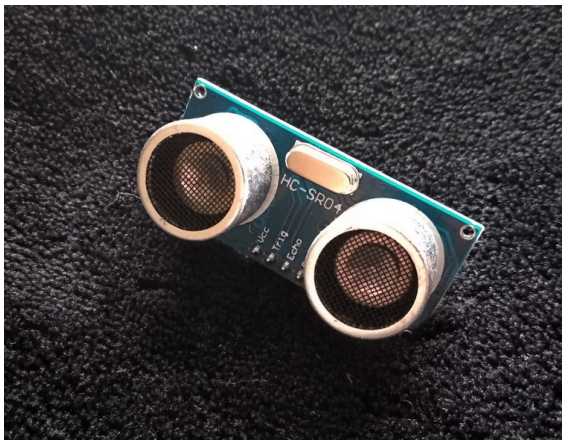
Komponenty Trilobota

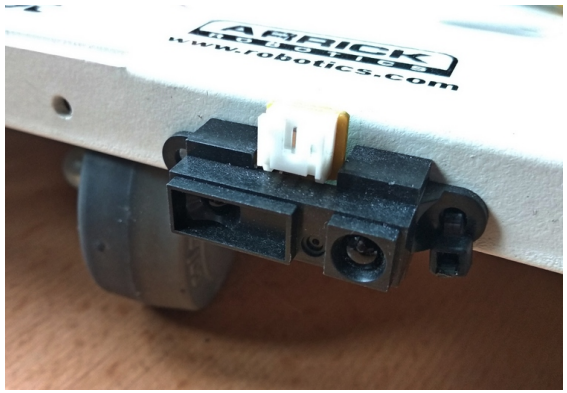
- Klasický 16x2 LCD displej připojený přes I^2C kvůli zjednodušení
- Možno ovládat přes objekt `display` nebo přes knihovnu `LiquidCrystalI2C.h`



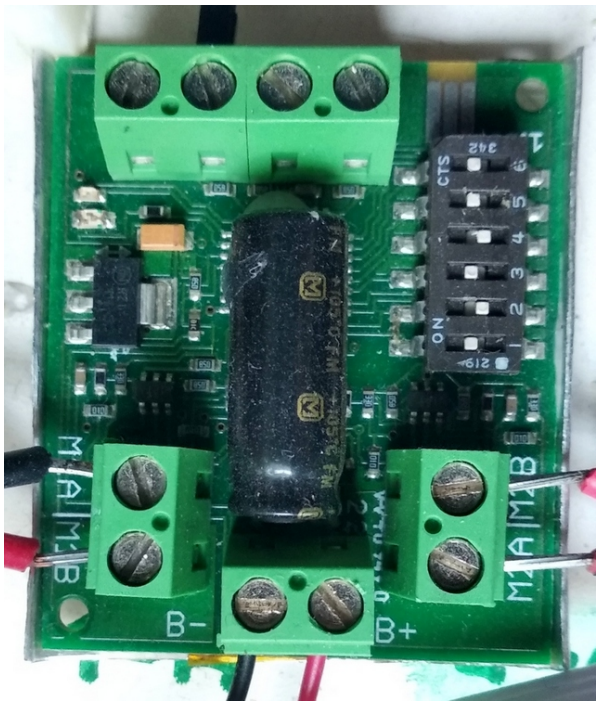
Senzory vzdálenosti

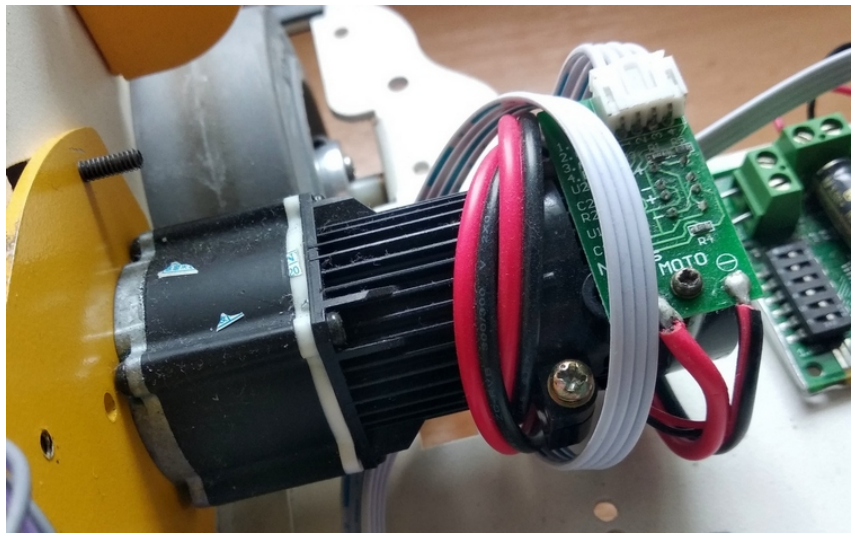
- Ultrazvukové HC–SR04 a SRF–08, infračervený Sharp
- Každý se hodí pro něco trochu jiného
- Ovládají se přes objekty `hcsr04`, `srf08` a `sharp`





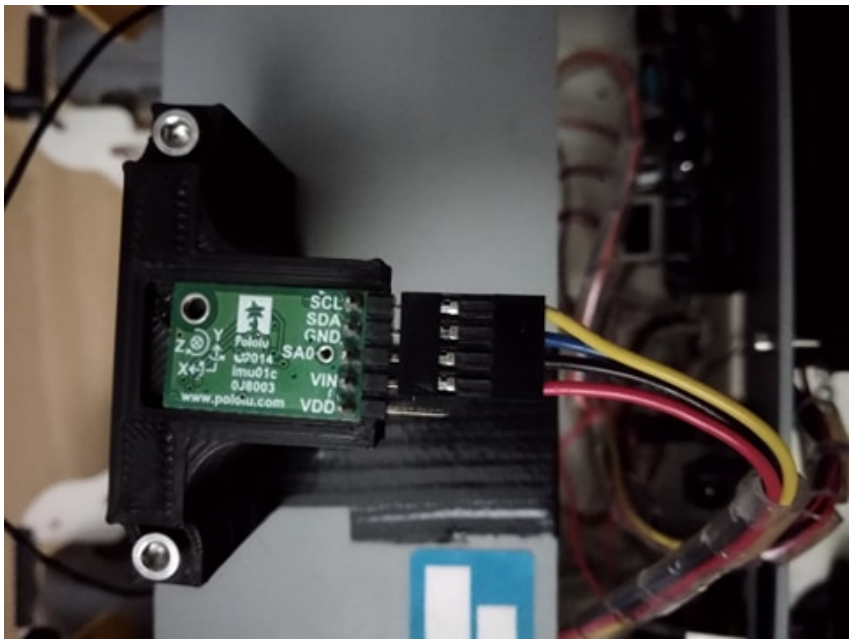
- K pohonu slouží dva motory Faulhaber řízené deskou Sabertooth
- Komunikace mezi Arduinem a Sabertooth probíhá přes seriovou linku `Serial1`
- Zpětná vazba o ujeté vzdálenosti přes rotační enkodéry
- Pro jejich ovládání slouží objekt `motors`





IMU – gyroskop, akcelerometr, magnetometr

- Tři senzory v jednom:
  - Gyroskop: měří úhlovou rychlost
  - Akcelerometr: Měří zrychlení
  - Magnetometr: Měří intenzitu magnetického pole (kompas)
- Data z IMU slouží pro zjišťování polohy – tu počítá AHRS (poziční systém)
- Senzor musí být dál od Trilobota, jinak je jeho magnetometr nepoužitelný (Trilobot je z kovu)
- Každý ze senzorů lze ovládat svým objektem: `gyro`, `accel` nebo `mag`



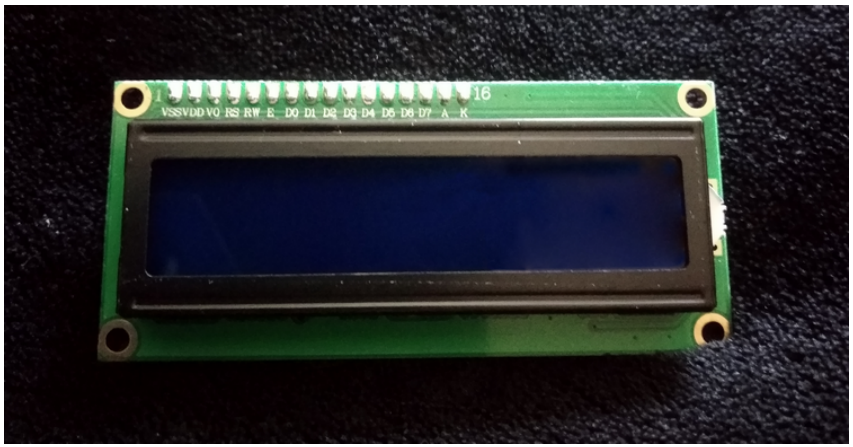
Před začátkem tutoriálu

- Zkratka **TODO** označuje části kódu, které bude třeba doplnit (z EN *To Do*)
- Číslo ve formátu **`0x00`**:
  - Číslo v hexadecimální soustavě
  - V IT se používá běžně, ale není to povinné
  - Chcete–li raději používat čísla v desítkové soustavě, bez problémů je můžete zaměnit, jen nezapomínejte na převod (převodníky na Google)!
- Funkce pro doplnění mají **český** název pro snadnou identifikaci, ostatní kód je psaný **anglicky** – pokud si nejste jistí angličtinou, své názvy proměnných pište česky bez diakritiky, počítač bude rozumět :)

Displej

Displej 16x2

- 16 znaků, 2 řádky
- Připojen přes I^2C
- LCD je relativně pomalá součástka – nezvládne se překreslovat 1000x za vteřinu! – musí se podle toho uzpůsobit kód
- Ovládání:
  - Vlastní knihovna `display.h` a třída `Display`



- Velmi jednoduchý wrapper* pro výpis např. dat ze senzorů bez znalosti práce s displejem
- Objekt třídy `Display` (defaultně pojmenovaný `display`) má tři funkce:

```
  #include <display.h> /* Knihovna pro praci s displejem*/
  ...
  /** Funkce smazou dany radek a vytisknou data*/
  display->print_first_line(to_print);
  display->print_second_line(to_print);
  display->clear(); /* Smaze cely displej */
 }
```

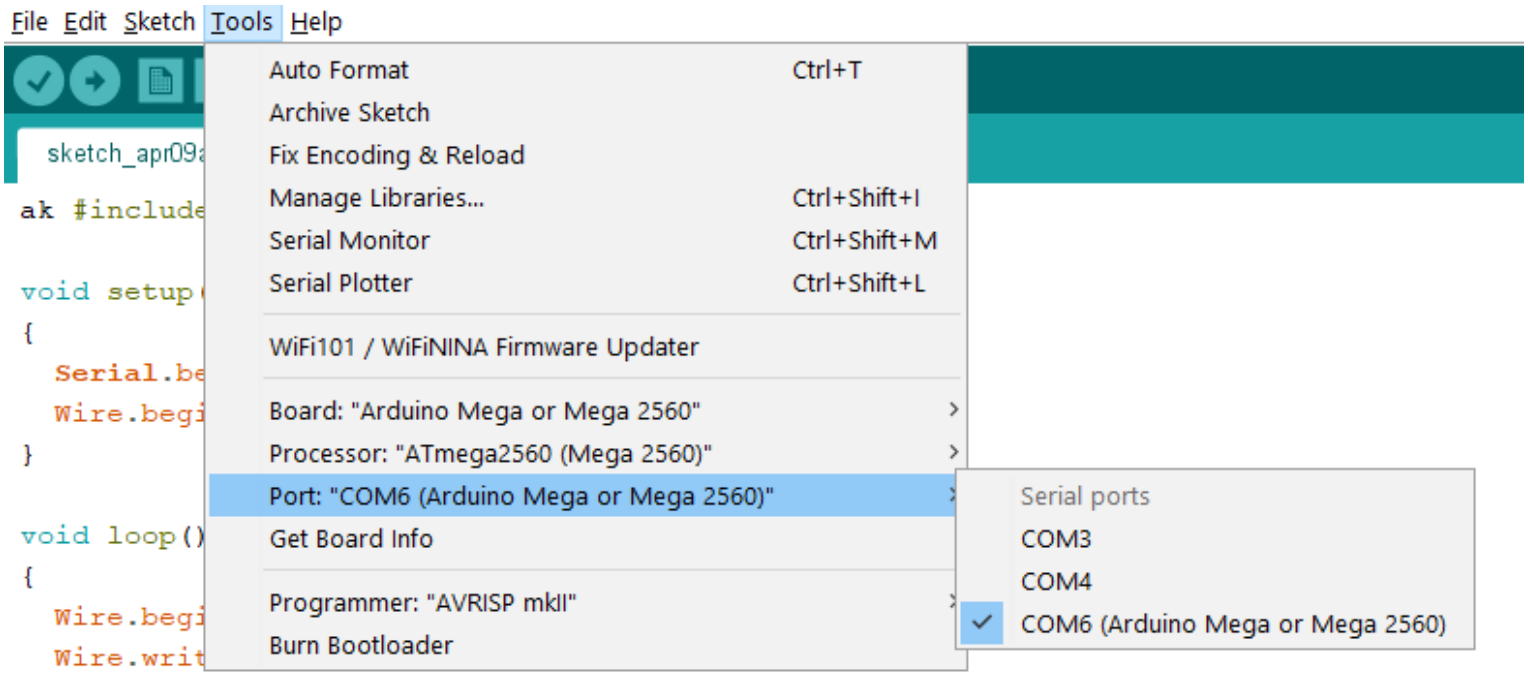
*wrapper = třída, která obaluje jinou třídu a umožňuje ji používat jinak. V našem případě jednodušeji.

Arduino IDE a připojení Trilobota

- Vývojové prostředí pro psaní a nahrávání kódu do desek Arduino
- Jednoduché – nepodporuje našeptávání apod.
- Pro naprogramování Arduina je třeba vybrat port, na kterém je připojené:
  - Přes *Tools/Port* vybereme správný COM port
  - Pokud nevíme, který to je, otevřeme si nabídku COM portů, odpojíme a znovu připojíme Trilobota. Port, který zmizí a zase se objeví bude ten správný.
- Popis Arduino IDE (obrázek na dalším slajdu):
  - (1): Hlavní část, kam se píše kód
  - (2): Konzole, kde se objevují informace při nahrávání kódu
  - (3): Tlačítka pro (zleva) kontrolu, nahrání, nový soubor, otevření a uložení


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

- Pomocí USB kabelu
- Po úspěšném zapojení by se měl rozsvítit displej a v Arduino IDE (viz dále) se objeví nový COM port
- Přes *Tools/Port* vybereme požadovaný port – je u něj napsáno, že se jedná o Arduino Mega



Test spojení

- Pomocí *File/New* nebo klávesové zkratky ctrl + N si otevřeme nový sketch (zdrojový soubor s koncovkou .ino)
- Po připojení Trilobota k počítači podle předchozího slajdu zkopírujte do funkce `loop()` následující kód a nahrajte ho kliknutím na zelenou šipku vlevo nahoře:

```
void loop(){
    /*pin 13 do logicke jednicky - rozsviti se dioda*/
    digitalWrite(13, HIGH);
    /*Pul vteriny/500 ms pockame */
    delay(500);
    /*Zmenime stav na logickou nulu - dioda zhasne*/
    digitalWrite(13, LOW);
    /*Opet pockame 500 ms*/
    delay(500);
}
```

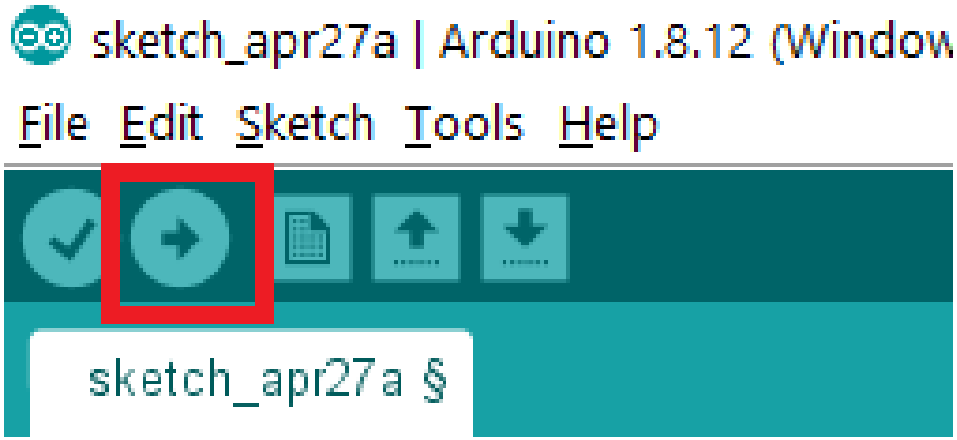
- Pokud je vše v pořádku, měla by začít blikat dioda vedle displeje

Úvod do programování Arduina a OOP

- Arduino se programuje v jazyce Wiring – plně kompatibilní s C++
- V programu se vždy nachází dvě funkce:
  - `setup()`: Tento kód se volá jen jednou, a to při spuštění nebo resetu Arduina
  - `loop()`: Kód v této funkci se volá opakovaně stále dokola – je to nekonečná smyčka

```
void setup()
{
/* Tento kod se provede jen jednou - treba nastaveni */
}
void loop()
{
/* Tento kod se bude volat opakovane
 * - treba cteni ze senzoru */
}
```

- Kliknutím na zelenou šipku (viz obrázek) se kód přeloží a nahraje do Arduina nebo Arduino IDE oznámí, že nastala chyba
- Arduino IDE pravděpodobně bude chtít před přeložením kód uložit – není to nutné, ale doporučené
- Pokud nastala syntaktická chyba, Arduino IDE oznámí, kde



- OOP je způsob programování, kdy se vytvářejí *objekty*
- *Objekty* v kódu pak odpovídají objektům reálného světa ve dvou věcech:
  - Charakteristikou – atributy: barva psa nebo I^2C adresa konkrétního senzoru
  - Vlastnostmi – metodami: pes může štěkat nebo senzor měřit
- *Objekty* mají šablony, podle kterých se tvoří – třídy
  - Například třída `Pes` je šablonou pro konkrétní objekt psa `rex`

- Vše si ukážeme na objektu `rex` a `display`. Ten druhý je i v kódu a dá se použít
- Pokud chceme zavolat metodu na konkrétním objektu, používáme následující syntaxi:

```
jmeno_objektu->nazev_metody(parametry);
```

- Kdybychom chtěli, aby náš `rex` zaštěkal, napíšeme:

```
rex->zastekej();
```

- A pro vytisknutí řádku na displej napíšeme:

```
display->print_first_line("Ahoj svete!");
```

- Ve zkratce: Volání metody se podobá volání funkce, jen před ni připíšeme název objektu a šipku `->`

Tutoriál

Fotosenzor

Fyzikální princip jednoduše:

- Pokud na fotosenzor nesvítíme, proud vede velmi špatně (téměř vůbec)
- Světlo ale dodá energii elektronům v atomech a ty se odtrhnou od svého atomu –> vznikají mj. volné elektrony a ty už vedou proud
- Čím více světla, tím více elektronů a tím více proudu, který se dá měřit

Fotosenzor

- Většinou maličký senzor, rozpoznatelný podle charakteristických kliček
- V našem případě součást senzoru SRF–08

- Otestovat, zda jsou Trilobot a Arduino IDE správně připojené
- Do kostry funkce `SRF08::zmer_svetlo()` (k nalezení v souboru `srf08.cpp`) doplnit:
  - Číslo registru, ve kterém SRF–08 ukládá hodnotu z fotočidla (viz dokumentaci, strana 4, adresa je ve sloupci *Location*)
  - Vrátit přečtenou hodnotu
  - Vypsat hodnotu na displej (pomocí `display->print_first_line()`)
  - (nepovinné) Rozsvítit LED při nízké hladině světla

```
byte SRF08::zmer_svetlo()
{   this->set_measurement(); /* Zahajeni mereni */
    /*Pozadame SRF-08 o informaci
    pocinajici REGISTREM FOTOCIDLA*/
    Wire.beginTransmission(SRF08_ADDRESS);
    /* 0x00 je jen zastupny znak, aby sel kod prelozit */
    Wire.write(0x00/*TODO REGISTR FOTOCIDLA*/);
    Wire.endTransmission();

    /*Pozadame pouze o jeden bajt...*/
    Wire.requestFrom(SRF08_ADDRESS, 1);
    /*... a pockame, dokud nebude dostupny*/
    while(Wire.available() < 0);

/*Promenna pro ulozeni prectene hodnoty intenzity svetla*/
    byte light_intensity = Wire.read();

   /* TODO vratit hodnotu, kterou jsme precetli */
    return 0;
```

- Adresa registru s informací z fotočidla je **0x01**. Informace se nachází i v dokumentaci.
- Na displej vypíšeme data přes

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

- Teď už stačí pouze napsat vše do funkce.

Fotosenzor: Řešení SRF08::zmer_svetlo()

```
byte SRF08::zmer_svetlo()
{   this->set_measurement(); /* Zahajeni mereni */
    /*Pozadame SRF-08 o informaci
    pocinajici REGISTREM FOTOCIDLA*/
    Wire.beginTransmission(SRF08_ADDRESS);
    Wire.write(0x01);
    Wire.endTransmission();

    /*Pozadame pouze o jeden bajt...*/
    Wire.requestFrom(SRF08_ADDRESS, 1);
    /*... a pockame, dokud nebude dostupny*/
    while(Wire.available() < 0);

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

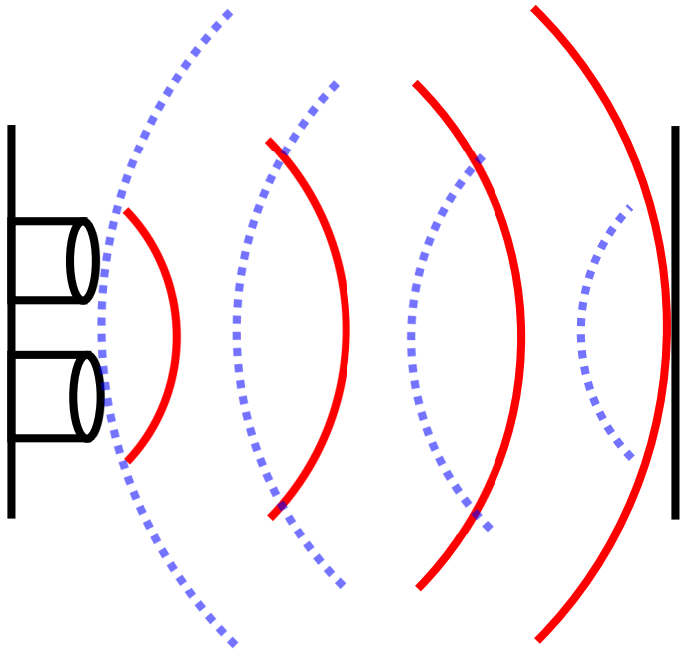
Měříme vzdálenost

V této kapitole si vysvětlíme:

- Jak můžeme měřit vzdálenost
- Měření pomocí senzorů HC–SR04, SRF–08 a Sharp–GP2D120

Jak měříme vzdálenost

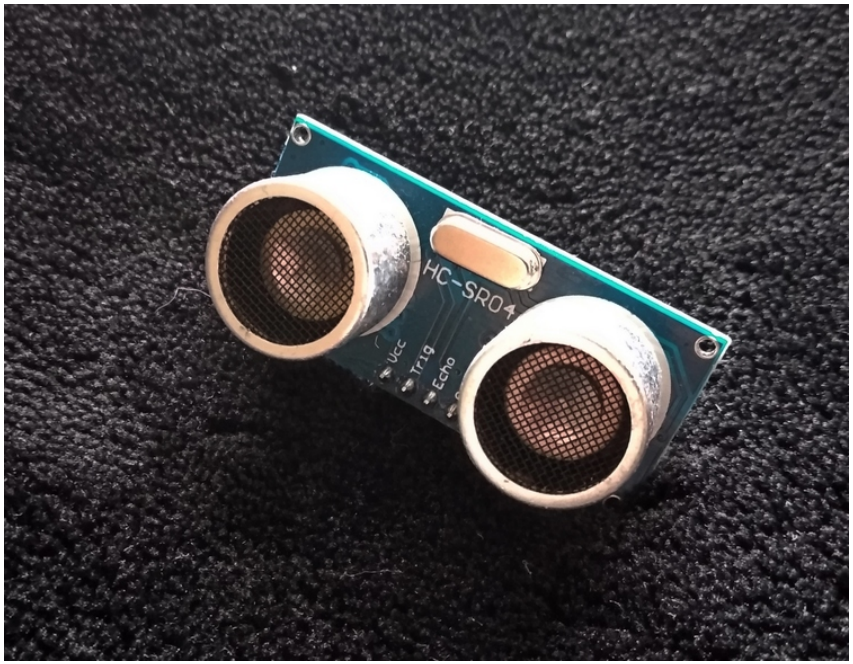
- Pravítkem :) – to roboti nemohou
- Pokud se robot pohybuje, tak pomocí otáček kol. To je ale vcelku nepřesné – prokluzování a podobně
- Pomocí pohybových senzorů – měří ujetou vzdálenost, ne vzdálenost od překážky
- Bezdrátově – odrazem nějaké vlny (zvuk, světlo…)
  - A to si vyzkoušíme i my na všech třech senzorech
  - Princip vidíme na obrázku: červené vlny vysílá senzor, modré (odražené) zachytává. Z uplynulého času mezi vysláním a přijetím a rychlosti vlny lze spočítat vzdálenost



HC_SR04

HC_SR04: Úvod

- Ultrazvukový senzor pro měření vzdálenosti – používá podobný princip jako netopýr nebo velryba
- 4 vodiče: 2x napájení, `ECHO_PIN` a `TRIGGER_PIN` (tyhle dva názvy se dají použít i v kódu pro práci s danými piny)
- Dokumentace



- Když je na `TRIGEER_PIN` alespoň `10` mikrosekund stav HIGH, senzor začne měřit
- Dokud se nevrátí odražená vlna, drží `ECHO_PIN` ve stavu HIGH
- Sotva dorazí ozvěna, pin se vrátí do stavu LOW
- Hodnota v mikrosekundách se musí převést na vzdálenost pomocí rychlosti zvuku: **`340 m/s neboli 0.0343cm/us`**

- Doplnit kostru funkce `HCSR04::zmer_vzdalenost()`:
  - Na `TRIGGER_PIN` nastavit hodnotu HIGH na 10 mikrosekund (použijeme funkci `delayMicroseconds()`)
    Senzor vyšle zvukovou vlnu a čeká na její odraz
  - Naslouchat na `ECHO_PIN` a získat délku pulzu v mikrosekundách (k tomu slouží funkce `pulseIn(nazevPinu, uroven HIGH nebo LOW)`)
    Sotva se vlna vrátí, senzor ukončí pulz a vrátí `ECHO_PIN` do stavu LOW
  - Převést mikrosekundy na centimetry či jinou vhodnou jednotku (pomocí rychlosti zvuku)
    Spokojíme se s hodnotou 340 m/s (nebo 0.0343cm/us, což se nám hodí víc)

```
long HCSR04::zmer_vzdalenost()
{/* Pro jistotu na chvili nastavime stav LOW */
    digitalWrite(TRIGGER_PIN, LOW);
    delayMicroseconds(2);
    /* TODO: Nastavit TRIGGER_PIN do stavu HIGH na 10
     * mikrosekund, pote ho vratit do LOW*/

     /*TODO: pomoci funkce pulseIn(pin, stav) zjistit, jak
      * dlouho bude na ECHO_PIN stav HIGH  a prevest
      * cas na vzdalenost*/
    long return_value = 0;
    return return_value;
}
```

- Na `TRIGGER_PIN` nastavíme HIGH na 10 mikrosekund pomocí

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

- Na `ECHO_PIN` naslouchám pomocí funkce `pulseIn`:

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

- Převod z času na vdálenost, známe–li rychlost, je už hračka :)
  (s = v*t, kde `s` je vzdálenost, `v` je rychlost a `t` je čas)

```
long HCSR04::zmer_vzdalenost()
{/* Pro jistotu na chvili nastavime stav LOW */
    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;
}
```

Ve funkci `loop()` můžeme třeba vypsat hodnotu, kterou nám funkce vrací, na displej. K tomu nám slouží funkce `display->print_first_line(to co chceme vypsat)`.

SRF-08

SRF–08: Úvod

- Stejný princip jako u HC_SR04 (echolokace)
- Komunikace přes I^2C – nic extrémně složitého, ale my se obejdeme bez jejího vysvětlení (všechno za nás vyřeší připravená kostra)
- Dokumentace
- Kromě měření vzdálenosti i senzor světla
- Má mnohem podrobnější nastavení, nám ale stačí základní

- Vybereme jednotku, ve které chceme získat data:
  - `0x50`: palce
  - `0x51`: centimetry
  - `0x52`: mikrosekundy
- Počkáme, než se měření provede. V dokumentaci se píše, že stačí 60 ms, občas je ale potřeba více. My budeme používat 100 ms.
- Vyžádáme si první hodnotu přes I^2C :
  - První hodnota není v prvním registru (ale v registrech 2 a 3)
  - Je rozdělena na horní a spodní bajt, horní je v registru 2, spodní v registru 3
  - Musíme je spojit

- Doplnit funkci `SRF08::zmer_vzdalenost()`:
  - Odeslat do registru `0` správnou jednotku z předešlého slajdu.
  - Počkat `100` milisekund (`60` ne vždycky stačí)
  - Spojit hodnoty z registrů 2 a 3 (horní a dolní bajt)
    - Tohle není úplně snadné, ale vysvětlíme si to. A pokud by se to někomu ani poté nedařilo, je k dispozici i řešení, takže žádný strach.
  - Vrátit spojenou hodnotu
  - Opět můžeme hodnotu vypsat na displej.

```
int SRF08::zmer_vzdalenost(){
  /*Navazeme komunikaci*/
  Wire.beginTransmission(SRF08_ADDRESS);
  /*Napiseme, kam chceme zapisovat...*/
  Wire.write(REG_CMD); /* expanduje do 0x00 */
  /*...a co chceme zapisovat.*/
  Wire.write(0x00/* TODO doplnit spravnou jednotku */);
  /*Nakonec ukoncime prenos*/
  Wire.endTransmission();
  /* TODO: pockat 100 ms*/
  /* Zajimaji nas data od registru 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 spojit data a vratit je*/
  uint16_t return_value = 0;
  return return_value;}
```

- Správnou jednotku si můžeme vybrat, ale pro nás se nejvíc hodí 0x51 neboli centimetry
- Čekání je brnkačka – stačí `delay()`
- Spojování dvou bajtů do jednoho čísla už je trochu složitější. Níže je k dispozici řešení, pod ním i vysvětlení.

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

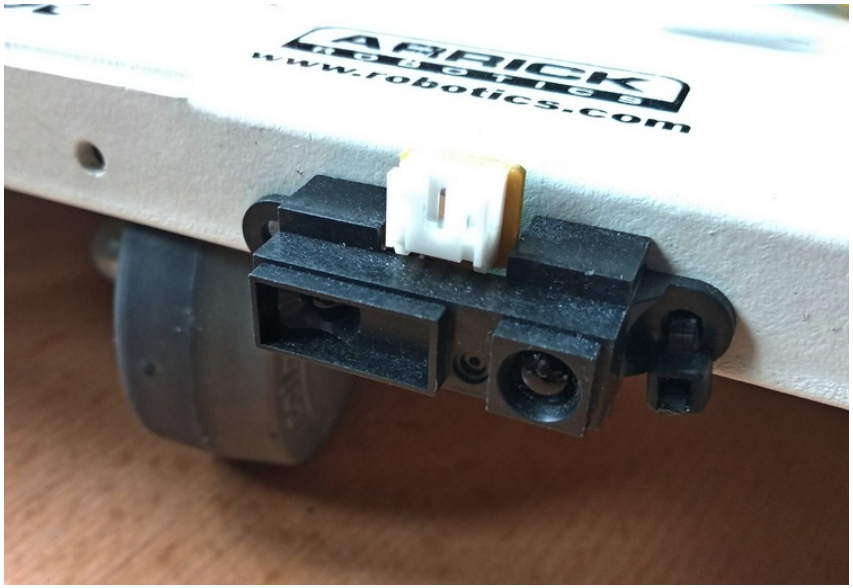
- Operátor << si představte jako posuvník, který posune číslo vlevo od něj o x řádových míst doleva.
  - Třeba výsledek 111 << 3 bude 111000 (na volné místo dosadí nuly).
  - Příklad:
    - 101 << 3 = 101000 (posunutí o tři pozice vlevo)
    - 101000 + 011 = 101011 (obyčejné sčítání)
  - My používáme úplně stejný princip: `high` posuneme o 8 pozic doleva a vytvoříme tím místo pro `low`, které poté přičteme.

```
int SRF08::zmer_vzdalenost()
{
  Wire.beginTransmission(SRF08_ADDRESS);
  Wire.write(REG_CMD); /* expanduje do 0x00 */
  Wire.write(0x51); /* centimetry*/
  Wire.endTransmission();
  delay(100); /* cekani */
  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: Úvod

- Místo zvuku používá odrazu světla
- Funguje až od 3 centimetrů
- Velmi jednoduchý princip: Vzdálenost se odvodí z hodnoty napětí na datovém vodiči
- Neexistuje lineární vztah mezi napětím a naměřenou vzdáleností. Jinak řečeno, převod mezi hodnotou napětí a skutečnou vzdáleností je poněkud složitý. V našem případě se ale o to postará sám program.



- Krátkou chvíli (na poměry Arduina prakticky okamžitě) po přivedení napětí začíná senzor měřit
- Na datovém vodiči (makro `SHARP_PIN`) se objevuje napětí
- (pro zájemce) Pomocí vztahu z dokumentace lze dopočítat vzdálenost :
  - Analyticky: Najdeme funkci, která se co nejvíc blíží vztahu z dokumentace
  - Numericky: původní vztah z dokumentace po částech nahradíme nějakými funkcemi – třeba několika přímkami. Výhodou je, že postup bude mnohem rychlejší, nevýhodou bude nižší přesnost
  - Pro naši potřebu existuje funkce `aproximate()`, používající numerickou aproximaci – jednoduše řečeno, složitá funkce se nahradí několika přímkami, které plus minus kopírují její tvar.

- Doplňte funkci `Sharp::zmer_vzdalenost()`:
  - Přečtěte hodnotu na datovém pinu (název pinu je `SHARP_PIN`, potřebná funkce je `analogRead()`).
  - Převeďte hodnotu z intervalu `0`–`1024` na interval `0`–`5`:
    - Jelikož funkce vrací hodnotu v rozmezí `0`–`1024`, ale funkce `aproximate()` potřebuje rozmezí `0`–`5`, je nutné naměřenou hodnotu převést.
  - Pomocí funkce `aproximate()` získejte vzdálenost v cm.
  - Tu následně vrátíme a opět můžeme třeba vypsat na displej pomocí `display->print_first_line(to co chceme vypsat)`.

```
float Sharp::zmer_vzdalenost()
{
/*TODO:
 * precist hodnotu na SHARP_PIN
 * ziskanou hodnotu prevest do rozsahu 0-5
 * Pomoci funkce this->aproximate ziskat hodnotu v cm
 * a vratit ji.
  */
  float aproximated_value = 0;
  return aproximated_value;
}
```

- Přečíst hodnotu na `SHARP_PIN` je snadné:

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

- Při převodu z intervalu `0`–`1024` do `0`–`5` už musíme trochu zapřemýšlet, ale v konečném důsledku je to jenom trojčlenka. `value/1024` nám konvertuje hodnotu do rozsahu `0`–`1`
- Vynásobením `5` dostaneme hodnotu `0`–`5`

```
float normalized_value = 0;
normalized_value = (value/1024)*5;
```

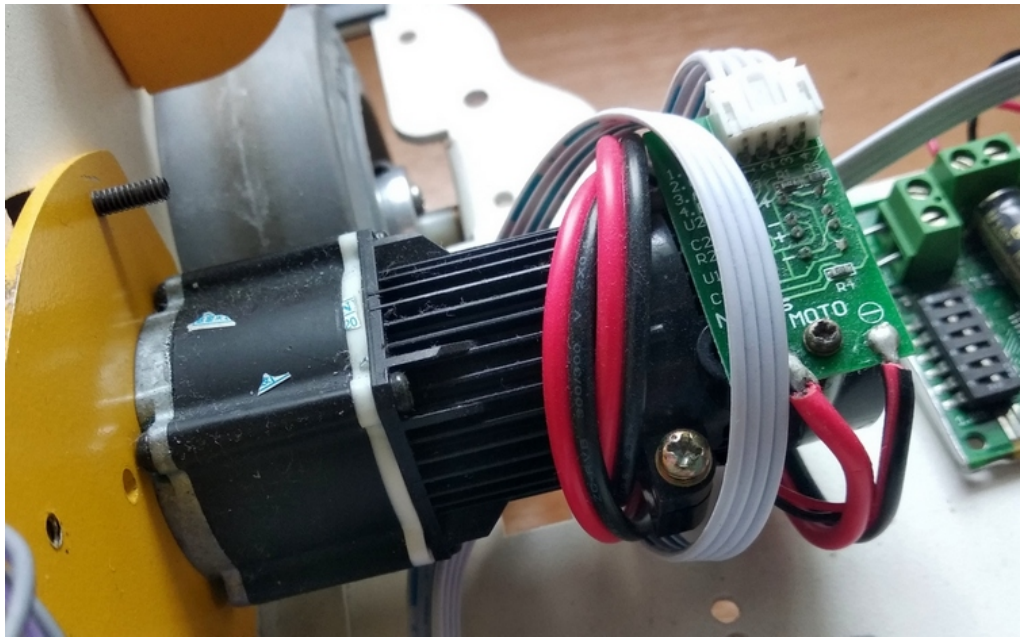
- A zavolání funkce zvládnou všichni :)

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

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

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

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

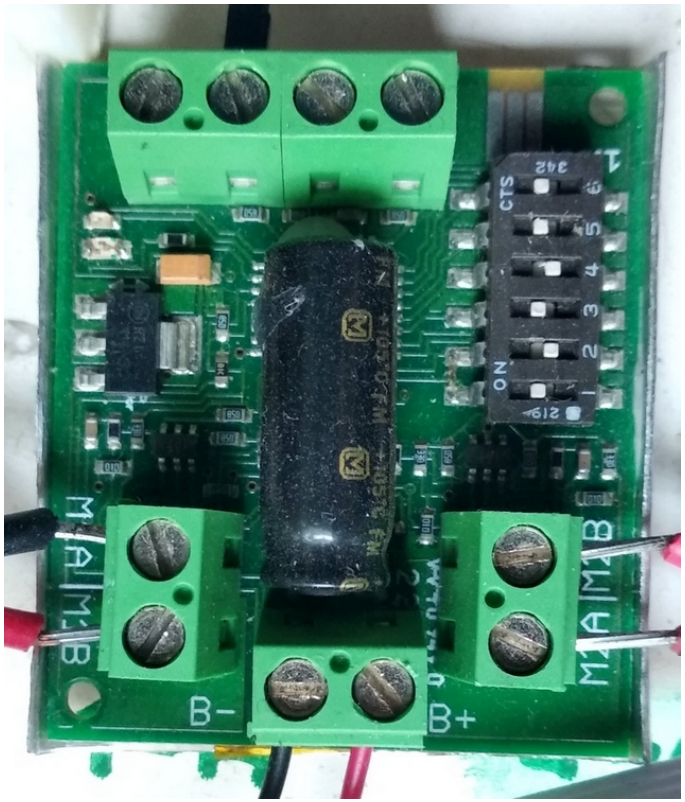
Jezdíme

- K pohybu slouží dvojice motorů s tzv. enkodéry – jeden z dvojice motorů na obrázku dole.
- Motory řízeny pomocí Sabertooth 2x5 (deska speciálně určená k ovládání motorů)
- Komunikace se Sabertooth přes seriovou linku, ale pouze v jednom směru – stačí jediný vodič (a jediný příkaz pro ovládání: `Serial1.write(hodnota)`).
- Enkodéry zajišťují zpětnou vazbu o pohybu



Řízení motorů přes Sabertooth 2x5

- Sabertooth pracuje v tzv. Simplified Serial módu
  - Princip: přes seriovou linku `Serial1` se odešle jeden bajt s danou hodnotou pro pravý a druhý bajt pro levý motor. Tím se levý, resp. pravý motor uvedou do pohybu
- Konkrétní hodnoty na dalším slajdu
- Komunikace je extrémně jednoduchá a pro Trilobota plně dostatečná.

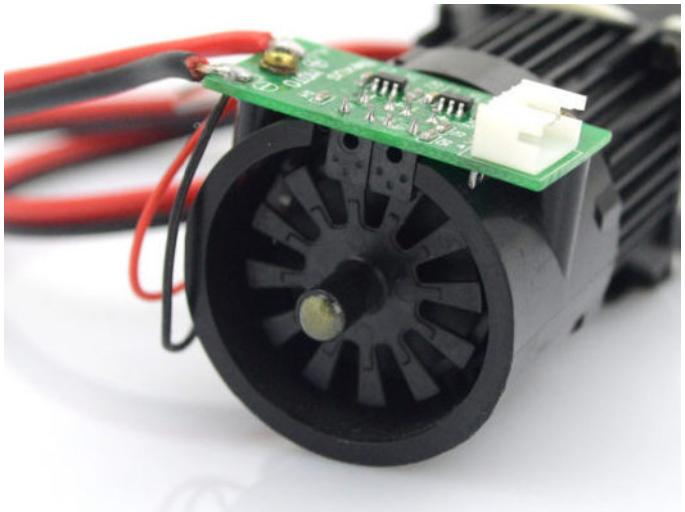


Základní hodnoty pro ovládání motorů

- 0 znamená oba motory stop
- 1–127 ovládá levý motor motor následovně:
  - 1 je naplno vzad
  - 64 je levý motor stop
  - 127 je naplno vpřed
- 128–255 ovládá pravý motor (jsou to čísla pro levý motor, ke kterým je přičteno 127):
  - 128 je naplno vzad
  - 192 je pravý motor stop
  - 255 je naplno vpřed

```
/* Priklad: */
Serial1.write(0x0); /* zastavi oba motory */
...
/* Oba motory naplno vpred - muzeme pouzit i desitkovou
 * soustavu, cislo je v komentarich */
Serial1.write(0x7F); /* = 127 */
Serial1.write(0xFF); /* = 255 */
```

- Enkodry nám dávají zpětnou vazbu o tom, kolik toho kolo ujelo, resp. jak hodně se otočil motor
- Za jednu otočku motoru o 360 stupňů přejde enkodér 768krát ze stavu LOW do stavu HIGH – pokud budeme počítat tyto změny, zjistíme, o kolik se motor otočil. A pokud známe i průměr kola, zjistíme i, jak daleko jsme se dostali.



Úkoly aneb co nás čeká

- Jízda rovně
- Zatáčení

- Oproti zatáčení výhoda: hodnota pro oba motory bude vždy posunutá o 127
- Princip funkce pro jízdu rovně:
  - **Vstupy**:
    - Vzdálenost v centimetrech, kterou chceme urazit
    - Rychlost v intervalu <−100,100>, kde záporná rychlost znamená jízdu dozadu
  - Hodnotu kontrolujících bajtů nám vrátí funkce `Motors->get_speed_from_percentage(rychlost)`, kde rychlost bude z intervalu výše.
  - Odešleme kontrolující bajty přes seriovou linku (`Serial1.write()`)
  - Počkáme, až robot ujede danou vzdálenost – v kódu je to nekonečný cyklus, který tak úplně nekonečný není
  - Zastavíme motory (opět odešleme správné číslo přes seriovou linku)

```
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 odeslat kontrolní bajt do obou motoru*/
  /*Toto si vysvetlime mimo kod*/
  while(1)
  {
    if(steps_left == steps_to_go ||
    steps_right == steps_to_go)
      break;
  }
  /*TODO zastavit motory*/
  detach_interrupts();
}
```

- V kódu se vyskytuje něco, co vypadá jako nekonečný cyklus
- Ve skutečnosti ale jde z cyklu vyskočit díky *přerušení*
- Co je to přerušení? Kdy se používá?
  - Představte si to jako učitele, který přednáší před třídou a jednou za čas se během přednášky přihlásí student a položí otázku bez toho, aniž by se učitel ptal, zda někdo nějaké otázky má. Učitel na ni odpoví a pokračuje tam, kde skončil.
  - Přerušení funguje úplně stejně.
- Pro nás to znamená jen tolik, že v tom nekonečném cyklu nijak neuvázneme :).

- Jediné, co potřebujeme, je odeslat hodnoty kontrolujících bytů:

```
Serial1.write(driving_byte);
Serial1.write(driving_byte+127);
...
Serial1.write(STOP_BYTE);
```

```
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);
  detach_interrupts();
}
```

Zatáčíme

Zatáčíme: Úvod

- Zatáčení se od jízdy vpřed v principu neliší – funkce budou velmi podobné
- Lze zatáčet mnoha různými způsoby:
  - Nejjednodušší: motor ve směru zatáčení stojí, druhý motor se pohybuje dopředu
  - Na místě: motor ve směru zatáčení se pohybuje opačně než druhý motor
  - Jako u auta: robot opíše oblouk…
- My zvolíme nejjednodušší variantu

- Princip funkce pro zatáčení:
  - **Vstupy**:
    - Úhel ve stupních z intervalu $< -180, 180 >$, kde kladný úhel bude znamenat zatáčení doprava, záporný zatáčení doleva.
    - Rychlost: tentokrát jen od `0` do `100`
  - Odešleme informaci o rychlosti motoru (podle směru zatáčení zvolíme pravý nebo levý motor).
  - Motor zastavíme, až dosáhneme požadované vzdálenosti.

```
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 zatoceni podle znamenka u parametru angle*/

  while(1)
  {
    if(steps_left == steps_to_go ||
      steps_right == steps_to_go)
      break;
  }
  /*TODO zastavit dany motor */
  detach_interrupts();
}
```

- Odeslání po seriové lince již umíme:

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

```
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();
}
```

Konec