Open-Cube

Katedra měření ČVUT FEL

2023

Obsah

Seznam obrázků

Seznam kódů

1 Instalace firmwaru

MicroPython firmware spolu se všemi Open-Cube knihovnami popsanými v tomto dokumentu je ve výchozím nastavení nahraný v kostce. Aktuální firmware lze také stáhnout z Open-Cube repozitáře [https://gitlab.fel.cvut.cz/open-cube/firmware/] ze složky micropython. Pro nahrání firmwaru při jeho aktualizaci, poškození či přepsání postupujte následujícími kroky:

1. Stáhněte si firmware [https://gitlab.fel.cvut.cz/open-cube/firmware/-/tree/main/micropython/] (uf2 binární soubor) pro kostku.
2. Připojte kostku k počítači pomocí USB kabelu.
3. Držte tlačítko boot select a stiskněte zapínací tlačítko.
4. V počítači otevřete adresář RPI-RP2 zkopírujte do něj firmware.
5. Po nahrání firmwaru se kostka restartuje a na displeji se zobrazí menu.

2 Editor kódu

Pro úpravu, debugování a nahrávání kódu do kostky doporučujeme použít editor Thonny [https://thonny.org/].

2.1 Nastavení editoru

Po prvním spuštění editoru klikněte na tlačítko v pravém dolním rohu a v záložce `Interpreter` nastavte interpretr na `MicroPython (Raspberry Pi Pico)`. Dále doporučujeme nastavit v záložce `General` možnost `UI mode` na `regular` nebo `expert`. V hlavním okně editoru v záložce `View` zvolte zobrazení lišt `Files` a `Shell`. Lišta `Files` slouží k procházení adresářů v počítači a `Shell` ke komunikaci s kostkou, vypisování informací a chybových hlášek.

2.2 Nahrání kódu do kostky

Po propojení zapnuté kostky USB kabelem k počítači se stisknutím tlačítka `Stop/Restart backend` v editoru připojíte ke kostce. V liště `Shell` se zobrazí informace o připojení a v liště `Files` se zobrazí soubory nahrané v kostce. Soubory i celé adresáře můžete kopírovat mezi kostkou a počítačem pravým kliknutím na daný soubor či adresář. Obdobně můžete mazat či vytvářet nové soubory a adresáře.

Upravovat soubory lze jak v adresáři počítače, tak v adresáři kostky. Pokud si otevřete soubor v kostce a po úpravách ho uložíte, automaticky se nahraje do kostky.

Uživatelské programy nahrávejte do adresáře `programs` v kostce. Programem může být jeden soubor s koncovkou `.py` nebo adresář, ve kterém je uživatelský soubor `main.py`. Takto nahrané programy se zobrazí v menu na kostce. Zobrazené jméno programu je určeno názvem souboru bez koncovky `.py` nebo názvem adresáře.

Po úpravě kódu v editoru můžete stisknutím tlačítka `Run current script` spustit na kostce kód aktuálně zobrazený v editoru. Doporučujeme takto spouštět pouze hlavní program `main.py`, který obsahuje framework kostky s inicializací všech potřebných funkcí

pro ovládání periferií. Po spuštění hlavního programu se na kostce zobrazí menu ovládané tlačítky na kostce s možností spuštění nahraného uživatelského programu. Soubor `main.py` se spustí automaticky při zapnutí kostky.

3 Robot

Po zapnutí kostky je hlavním programem `main.py` inicializovaný globální objekt `robot`, který obsahuje všechny funkce pro inicializaci, deinicializaci a přístup k objektům periferií kostky. Funkce jsou popsány v následujících kapitolách. Struktura proměnných objektu `robot`, kterými lze přistupovat k objektům periferií je následující:

```
robot
├── battery
├── buttons
├── buzzer
├── display
├── esp_uart
├── led
├── motors[4]
└── sensors
    ├── ultra
    ├── gyro
    ├── touch[4]
    ├── light[4]
    └── sound[4]
```

class **Robot**()

Inicializace, deinicializace a správa objektů motorů, senzorů a periferií kostky. Automaticky inicializuje tlačítka, LED, bzučák a měření napětí baterií s ochranou podvybití. Uživatelem lze inicializovat senzory, motory a ESP32 Bluetooth komunikaci.

Inicializace: Zapnutí kostky.
Deinicializace: Vypnutí kostky.
Přístup: `robot`

init_sensor(*sensor_type=None, port=None*)

Inicializace senzorů.

Parametry: **sensor_type** (Sensor) – Typ senzoru.
port (Port) – Port senzoru. Pro typ Sensor.GYRO a Sensor.ULTRA není potřeba specifikovat.

deinit_sensor(*sensor_type=None, port=None*)

Denicializace senzorů.

Parametry: **sensor_type** (Sensor) – Typ senzoru. Pokud není specifikován, deinicializuje senzor na daném portu.
port (Port) – Port senzoru. Pokud není specifikován, deinicializuje všechny senzory daného typu.

init_motor(*port=None*)

Inicializace motorů.

Parametry: **port** (Port) – Port motoru.

deinit_motor(*port=None*)

Denicializace motorů. Vypne regulátor, snímání enkodérů a zastaví motor.

Parametry: **port** (Port) – Port motoru.

init_esp_uart()

Inicializace ESP32 Bluetooth komunikace.

deinit_esp_uart()

Denicializace ESP32 Bluetooth komunikace.

Následující kód ukazuje použití třídy robot na jednoduchém programu pro blikání LED na kostce. Po spuštění tohoto programu z menu se periodicky každou sekundu mění stav LED na kostce. Po stisknutí levého tlačítka se program ukončí, na displeji se opět zobrazí menu a lze spustit další program.

```
# Importováni funkce pro uspáni programu
from time import sleep
# Importováni konstant tlačítek na kostce
from lib.robot_consts import Button

# Definice programu
def main:
    # Přistoupení ke globalní proměnné robot
    global robot

    # Smyčka čekající na ukončení programu
    while True:
        # Změna stavu LED
        robot.led.toggle

        # Získání stavu tlačítek
        buttons = robot.buttons.pressed
        # Ukončení programu, pokud je levé tlačítko stisknuté
        if buttons[Button.LEFT]:
            break

        # Uspání programu na 1 sekundu
        sleep(1)

# Spuštění programu
main
```

Kód 1: Použití globální třídy robot.

3.1 Tlačítka

class Buttons()

Informace o stavu stisknutí tlačítek na kostce.

Inicializace:	Zapnutí kostky.
Deinicializace:	Vypnutí kostky.
Přístup:	`robot.buttons`

pressed()

Navrátí stav tlačítek. Pro zjištění stavu konrétního tlačítka lze použít konstanty tlačítek.

Vrací:	Tuple stavu tlačítek zapnout, doleva, doprava, ok, nahoru, dolů. True pokud je tlačítko stisknuté, False pokud není.
Typ:	(bool, bool, bool, bool, bool, bool)

```
# Importování funkce pro uspání programu
from time import sleep
# Importování konstant tlačítek na kostce

def main:
  global robot
  while True:

    # Získání stavu tlačítek
    buttons = robot.buttons.pressed

    # Zobrazení stavu tlačítek v~terminálu
    print("Tlačítko stisknuté:",
          "zapnout:", buttons[Button.POWER],
          "doleva:",  buttons[Button.LEFT],
          "doprava:", buttons[Button.RIGHT],
          "OK:",      buttons[Button.OK],
          "nahoru:",  buttons[Button.UP],
          "dolů:",    buttons[Button.DOWN])
    # Ukončení programu, pokud je levé tlačítko stisknuté
    if buttons[Button.LEFT]:
      break
    # Uspání programu na 1 sekundu
    sleep(1)

# Spuštění programu
main()
```

Kód 2: Zjištění stavu tlačítek.

3.2 LED

class **Led**()

Zapínání a vypínání LED uvnitř kostky.

Inicializace: Zapnutí kostky.
Deinicializace: Vypnutí kostky.
Přístup: `robot.led`

on()

Zapne LED.

off()

Vypne LED.

toggle()

Změní stav LED.

3.3 Bzučák

class **Buzzer**()

Ovládání bzučáku (piezoelektrického reproduktoru) v kostce.

Inicializace: Zapnutí kostky.
Deinicializace: Vypnutí kostky.
Přístup: `robot.buzzer`

set_freq_duty(*freq*, *duty*)

Nastaví PWM na bzučáku na požadovanou frekvenci a střídu.

Parametry:
- **freq** (frekvence: Hz) – Frekvence PWM.
- **duty** (střída: %) – Střída PWM.

off()

Vypne bzučák.

3.4 Baterie

class **Battery**()

Měření napětí na napájecích bateriích. Při inicializaci se spustí časovač s periodou 200 ms, který spouští měření napětí.

Inicializace: Zapnutí kostky.
Deinicializace: Vypnutí kostky.
Přístup: `robot.battery`

voltage()

Navrátí poslední změřené napětí na bateriích.

Vrací: Poslední změřené napětí na bateriích.
Typ: float: V

read_voltage()

Změří napětí na bateriích.

Vrací: Poslední změřené napětí na bateriích.
Typ: float: V

deinit()

Vypne periodické měření napětí.

3.5 Displej

class **SH1106_I2C**()

Zobrazování textu, geometrických tvarů a grafů na displeji. Aktivování jednotlivých pixelů se zaznamenává do frame bufferu, který je příkazem `show` celý přenesen do displeje.

Inicializace: Zapnutí kostky.
Deinicializace: Vypnutí kostky.
Přístup: `robot.display`.

show()

Zobrazí aktuální frame buffer na displeji.

fill(*color*)

Vyplní celý frame buffer určenou barvou.

Parametry: **color** (0/1) – Barva – 0 černá, 1 bílá.

pixel(*x, y [, color]*)

Nastaví pixel ve frame bufferu na určenou barvu. Pokud barva není určena, navrátí nastavenou barvu pixelu.

Parametry:
- **x**, **y** – Souřadnice pixelu.
- **color** (0/1) – Barva – 0 černá, 1 bílá.

Vrací: Barva pixelu.

Typ: int (0/1)

hline(*x, y, w, color*)

Nakreslí horizontální linii do frame bufferu.

Parametry:
- **x**, **y** – Souřadnice levého začátku linie.
- **w** – Šířka linie.
- **color** (0/1) – Barva – 0 černá, 1 bílá.

vline(*x, y, h, color*)

Nakreslí horizontální linii do frame bufferu.

Parametry:
- **x**, **y** – Souřadnice horního začátku linie.
- **h** – Výška linie.
- **color** (0/1) – Barva – 0 černá, 1 bílá.

line(*x1, y1, x2, y2, color*)

Nakreslí linii do frame bufferu.

Parametry:
- **x1**, **y1** – Souřadnic začátku linie.
- **x2**, **y2** – Souřadnice konce linie.
- **color** (0/1) – Barva – 0 černá, 1 bílá.

rect(*x, y, w, h, color*)

Nakreslí obdélník do frame bufferu.

Parametry:
- **x**, **y** – Souřadnice levého horního vrcholu obdélníku.
- **w**, **h** – Výška a šířka obdélníku.
- **color** (0/1) – Barva – 0 černá, 1 bílá.

fill_rect(*x, y, w, h, color*)

Nakreslí vyplněný obdélník do frame bufferu.

Parametry:
- **x**, **y** – Souřadnice levého horního vrcholu obdélníku.
- **w**, **h** – Výška a šířka obdélníku.
- **color** (0/1) – Barva – 0 černá, 1 bílá.

ellipse(*x, y, xr, yr, color, f=False, m=1111b*)

Nakreslí elipsu do frame bufferu.

Parametry:
- **x**, **y** – Souřadnice středu elispy.
- **xr**, **yr** – Délka poloos elipsy.
- **color** (0/1) – Barva – 0 černá, 1 bílá.
- **f** (bool) – Vyplnění elipsy pokud je parametr specifikován a má hodnotu True.
- **m** (4 bity) – Omezení vykreslení elipsy do určených kvadrantů. Bit 0 určuje Q1, b1 Q2, b2 Q3 a b3 Q4. Kvadranty jsou číslovány proti směru hodinových ručiček a Q1 je pravý horní kvadrant.

fill_rect(*x1, y1, x2, y2, x3, y3, color*)

Nakreslí vyplněný trojúhelník do frame bufferu.

Parametry:
- **x1**, **y1**, **x2**, **y2**, **x3**, **y3** – Souřadnice vrcholů trojúhelníku.
- **color** (0/1) – Barva – 0 černá, 1 bílá.

text(*s, x, y, color=1*)

Nakreslí text do frame bufferu. Znaky mají rozměr 8x8 pixelů.

Parametry:
- **s** (string) – Text.
- **x**, **y** – Souřadnice levého horního rohu začátku textu.
- **color** (0/1) – Barva – 0 černá, 1 bílá.

draw_bar_chart_v(*value, x, y, w, h, low_lim=0, high_lim=100, no_of_tics=5, label=None, redraw=False*)

Nakreslí vertikální sloupcový graf do frame bufferu.

Vrací: Překreslení grafu pro snížení problikávání grafu.

Typ: bool

Parametry:
- **value** (float) – Zobrazovaná hodnota.
- **x**, **y** (int) – Souřadnice levého dolního rohu začátku grafu.
- **w**, **h** (int) – Šířka a výska grafu.
- **low_lim**, **high_lim** (int) – Dolní a horní omezení grafu.
- **no_of_tics** (int) – Počet rozdělení grafu pro lepší čitelnost. Doporučená maximální hodnota je 5.
- **label** (string) – Popisek grafu.
- **redraw** (bool) – Překreslení grafu.

draw_bar_chart_h(*value, x, y, w, h, low_lim=0, high_lim=100, no_of_tics=5, label=None, redraw=False*)

Nakreslí horizontální sloupcový graf do frame bufferu.

Vrací: Překreslení grafu pro snížení problikávání grafu.
Typ: bool
Parametry:
- **value** (float) – Zobrazovaná hodnota.
- **x**, **y** (int) – Souřadnice levého dolního rohu začátku grafu.
- **w**, **h** (int)– Šířka a výska grafu.
- **low_lim**, **high_lim** (int) – Dolní a horní omezení grafu.
- **no_of_tics** (int) – Počet rozdělení grafu pro lepší čitelnost. Doporučená maximální hodnota je 5.
- **label** (string) – Popisek grafu.
- **redraw** (bool) – Překreslení grafu.

draw_dial(*value, x, y, r, loval, hival, no_of_steps, label, redraw*)

Nakreslí kruhový číselník do frame bufferu.

Vrací: Překreslení grafu pro snížení problikávání číselníku.
Typ: bool
Parametry:
- **value** (float) – Zobrazovaná hodnota.
- **x**, **y** (int) – Souřadnice středu číselníku
- **r** (int)– Poloměr číselníku.
- **loval**, **hival** (int) – Dolní a horní omezení číselníku.
- **no_of_steps** (int) – Počet rozdělení číselníku. Doporučená maximální hodnota je 10.
- **label** (string) – Popisek číselníku.
- **redraw** (bool) – Překreslení číselníku.

continuous_graph(*x, y, gx, gy, w, h, xlo, xhi, ylo, yhi, label, redraw*)

Nakreslí průběžný graf do frame bufferu. Funkce si vnitřně udržuje hodnoty grafu. Při vyplnění šířky grafu se předchozí hodnoty odstraní a graf se začne vykreslovat od počátku.

Vrací: Překreslení grafu pro snížení problikávání grafu.
Typ: bool
Parametry:
- **x**, **y** (float) – Zobrazovaná souřadnice v grafu.
- **gx**, **gy** (int) – Souřadnice levého dolního rohu grafu.
- **w**, **h** (int) – Šířka a výska grafu.
- **xlo**, **xhi** (int) – Dolní a horní omezení osy x.
- **ylo**, **yhi** (int) – Dolní a horní omezení osy y.
- **label** (string) – Popisek grafu.
- **redraw** (bool) – Překreslení grafu.

```
# Vymazání displeje
robot.display.fill(0)
# Zapsání textu do frame bufferu
robot.display.text("test message", 0, 0, 1)
# Zobrazení frame bufferu na displeji
robot.display.show()
```

Kód 3: Použití displeje.

3.6 Gyroskop a akcelerometr

class ICM20608()

Gyroskop a akcelerometr. Měření zrychlení a úhlové rychlosti ve třech osách. Interrupt pin senzoru je připojen na GyroAcc.IRQ_PIN (GPIO 28) mikrokontroleru.

Inicializace:	`robot.init_sensor(sensor_type=Sensor.GYRO)`
Deinicializace:	`robot.deinit_sensor(sensor_type=Sensor.GYRO)`
Přístup:	`robot.sensors.gyro`

read_value()

Přečte ze senzoru a navrátí změřené hodnoty úhlových rychlostí a zrychlení. Pro zjištění konkrétní hodnoty lze použít GyroAcc konstanty gyroskopu a akcelerometru.

Vrací:	Touple zrychlení a úhlových rychlostí v osách x, y, z.
Typ:	(a_x: -, a_y: -, a_z: -, g_x: °/s, g_y: °/s, g_z: °/s)

Příklad čtení dat z gyroskopu s využitím irq:

```
# Importování knihoven
from time import sleep
from machine import Pin
# Importování konstant
from lib.robot_consts import Button, Sensor, GyroAcc

a_g_data = (0,0,0,0,0,0)

def main:
  global robot, a_g_data
  # Inicializace senzoru
  robot.init_sensor(sensor_type=Sensor.GYRO)
  # Nastavení GPIO 28
  icm_irq_pin = Pin(GyroAcc.IRQ_PIN, Pin.IN)
  # Nastavení callback funkce volané na náběžnou hranu irq, pokud jsou
    nová data k dispozici
  icm_irq_pin.irq(trigger=Pin.IRQ_RISING, handler=callback)

  # Smyčka čekající na ukončení programu
  while True:
    # Zobrazení posledních hodnot zrychlení a úhlové rychlosti
    print("Acc:",  a_g_data[GyroAcc.AX],
                   a_g_data[GyroAcc.AY],
                   a_g_data[GyroAcc.AZ],
          "Gyro:", a_g_data[GyroAcc.GX],
                   a_g_data[GyroAcc.GY],
                   a_g_data[GyroAcc.GZ],)
    # Získání stavu tlačítek
    buttons = robot.buttons.pressed
    # Ukončení programu, pokud je levé tlačítko stisknuté
    if buttons[Button.LEFT]:
      break
    # Uspání programu na 1 sekundu
    sleep(1)

  # Zrušení callbacku před deinicializací senzoru
```

```
36   icm_irq_pin.irq(handler=None)
37   # Deinicializace senzoru
38   robot.deinit_sensor(sensor_type=Sensor.GYRO)
39
40 def callback(p):
41   global robot, a_g_data
42   # Přečtení nových hodnot ze senzoru
43   a_g_data = robot.sensors.gyro.read_value
44
45 # Spuštění programu
46 main
```

Kód 4: Čtení dat z gyroskopu s využitím irq.

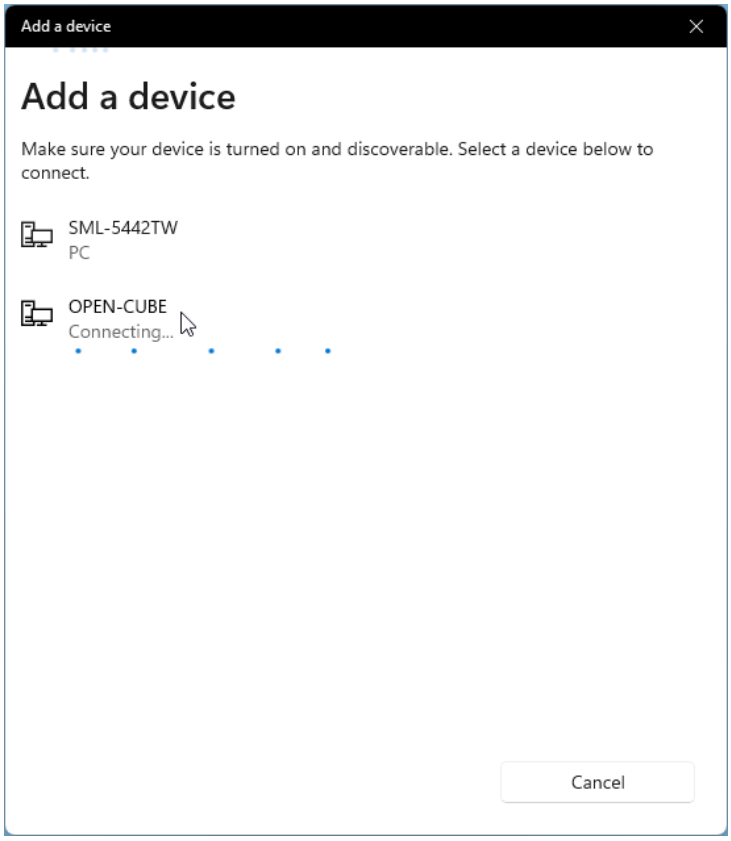
3.7 ESP32 komunikace

ESP32 komunikuje s kostkou pomocí sériové linky. S dalším zařízením může komunikovat pomocí Wi-Fi nebo Bluetooth. ESP je programovatelné po připojení USB kabelem na vrchní straně kostky. Výchozí firmware ESP (dostupný z Open-Cube repozitáře [https://gitlab.fel.cvut.cz/open-cube/firmware/-/tree/main/ESP]) pouze přeposílá data z kostky přes Bluetooth virtuální COM port do počítače a obráceně. Zkušenější uživatel si může naimplementovat vlastní komunikaci.

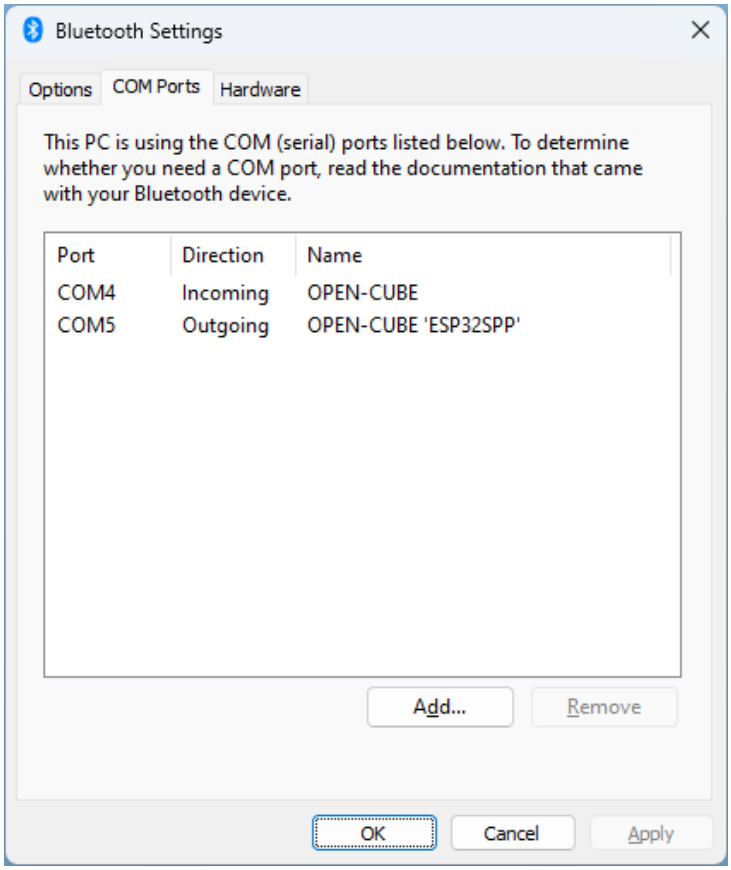
3.7.1 Bluetooth propojení ESP32 s počítačem

Komunikace probíhá sériově přes Bluetooth virtuální COM port a je otestováno pouze pro Windows.

1. Zapněte kostku.
2. V Bluetooth nastavení zvolte Přidat nové zařízení, dále Bluetooth a spárujte počítač s kostkou. Viz obrázek 1.
3. Pro zjištění čísla COM portu pokračujte do pokročilých nastavení Bluetooth. Požadovaný COM port má název obsahující ESP32SPP a směr Outgoing. Viz obrázek 2.
4. Tento COM port lze použít pro přijímání i odesílání dat.



Obrázek 1: Spárování počítače s kostkou.



Obrázek 2: Zjištění čísla COM portu.

3.7.2 Kostka

class **EspUart**()

Oboustranné posílání dat mezi mikroporcesorem a ESP32 pomocí sběrnice UART. Při inicializaci se spustí časovač s periodou 10 ms, při kterém se přijímají a ukládají data posílaná ESP.

Data se posílají po zprávách, což jsou celky dat patřící k sobě. Odesílání a přijímání dat může probíhat v ASCII nebo v binárním režimu.

ASCII režim: Odesílaná zpráva je zakončena terminátorem LF (new line). Při přijímání dat je zpráva k dispozici po přijetí terminátoru LF nebo CR. Data jsou typu string.

Binární režim: Pro komunikaci je nutné specifikovat hlavičku odesílaných a přijímaných dat, aby bylo možné rozeznat začátek zprávy. Aby bylo možné zprávu správně přijmout a určit její konec, musí být předem specifikovaná velikost zprávy v bytech. Data jsou typu bytes a jejich interpretace je ponechána na uživateli.

Inicializace: `robot.init_esp_uart()`
Deinicializace: `robot.deinit_esp_uart()`
Přístup: `robot.esp_uart`

read()

Navrátí poslední přijatou zprávu.

Vrací: Přijatá zpráva. Pokud není nová zpráva k dispozici: None.
Typ:
- ASCII režim: string
- Binární režim: bytes

write(*buff*)

Odešle zprávu.

Parametry: **buff** (buffer: string(ASCII)/bytes(binární)) – Buffer odesílaných dat.

set_binary(*header, data_size*)

Nastaví komunikaci binárního režimu. Pokud je header None, komunikace se nastaví do ACII režimu.

Parametry: **header** (tuple) – Hlavička libovolné délky obsahující čísla 0–255, nebo prvek None.
data_size (int) – Velikost přijímané zprávy v bytech.

Příklad použití komunikace mezi mikroprocesorem a ESP32:

```
# Importování knihovny pro práci s binárními daty
import struct

robot.esp_uart_init()
# Odeslání a přijetí zprávy typu string v režimu ASCII
robot.esp_uart.write("test message")
received_message = robot.esp_uart.read()
```

```
8
9  # Odeslání a přijetí zprávy o velikosti 8 bytů v binárním režimu
10 robot.esp_uart.set_binary((112, 241, 3, 62), 8)
11 # Vytvoření a odeslání zprávy typu bytes se dvěma čísly typu single
12 write_message = struct.pack("<ff", 521.9, -11.821)
13 robot.esp_uart.write(write_message)
14 # Příjem a dekódování zprávy obsahující dvě čísla typu single
15 received_message2 = robot.esp_uart.read()
16 (value1, value2) = struct.unpack("<ff", received_message2)
```

Kód 5: Komunikace mezi mikroprocesorem a ESP32.

3.7.2.1 Matlab

Pro komunikaci pomocí Matlabu je zapotřebí mít nainstalovaný Communications Toolbox. Toto rozhraní používá Bluetooth SPP a umožňuje přijímat a odesílat ASCII a binární data. Při posílání ASCII dat je zpráva zakončena terminátorem. V případě binárních dat zpráva není zakončena terminátorem a kostka i program v Matlabu musí předem znát délku zprávy, aby mohla být správně interpretovaná.

Příklad použití rozhraní je uveden v následujícím kódu. Další informace k použití rozhraní naleznete v oficiální dokumentaci [https://www.mathworks.com/help/matlab/bluetooth-communication.html/].

```
1  % Propojení kostky s Matlabem.
2  cube = bluetooth('OPEN-CUBE')
3  % Nastavená terminátoru LF (new line) pro odesílání a příjem
      ASCII dat.
4  configureTerminator(device, 'LF')
5  %%
6  % Poslání testovací zprávy do kostky. Zpráva je zakončena
      nastaveným terminátorem.
7  writeline(cube, 'test message')
8  % Příjmutí zprávy z kostky. Funkce přijímá data a čeká na
      zakončení nastaveným terminátorem.
9  cube_message = readline(cube)
10 %%
11 % Poslání testovací zprávy do kostky. Zpráva je
      interpretovaná jako formát string a není zakončena
      terminátorem.
12 write(cube, 'test message', 'string')
13 % Příjmutí zprávy z kostky. Zpráva má délku 8 bytů a je
      interpretovaná jako formát float.
14 cube_message2 = read(cube, 8, 'float')
```

Kód 6: Komunikace s kostkou z Matlabu.

3.7.2.2 Simulink

Pro komunikaci pomocí Simulinku je zapotřebí mít nainstalovaný Instrument Control Toolbox, který poskytuje bloky `Serial Configuration`, `Serial Send` a `Serial Receive` pro sériovou komunikaci. Při vytvoření nového modelu je potřeba upravit nastavení řešiče simulace a nastavit parametry sériové komunikace:

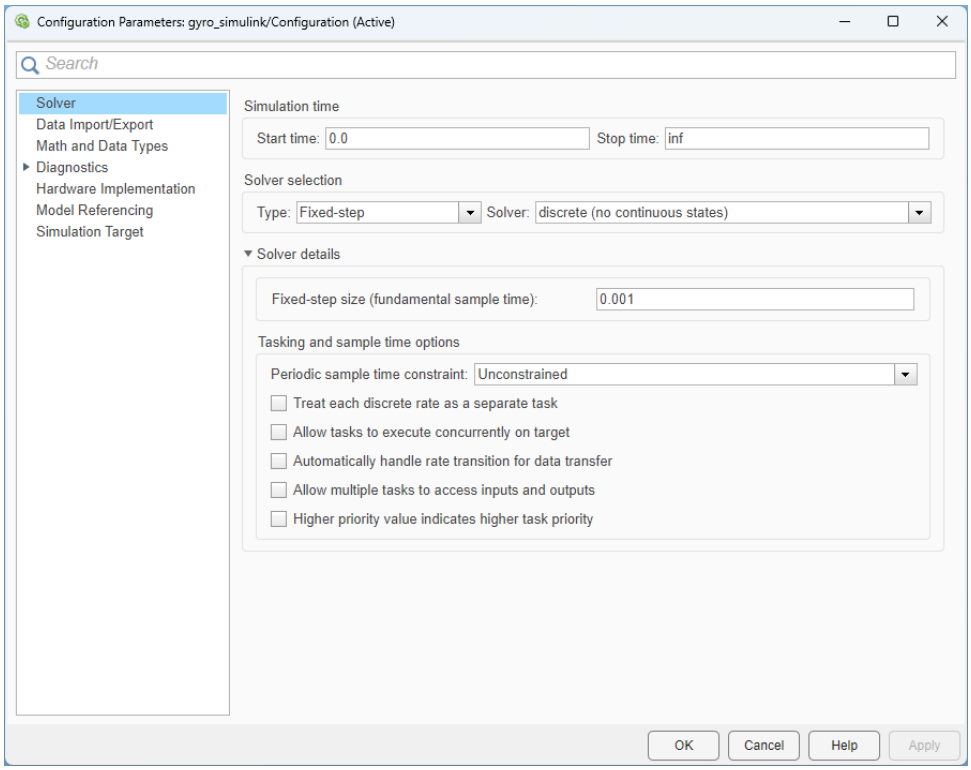
1. Kliknutím na tlačítko v pravém dolním rohu nastavit řešič na `Fixed-Step`, `discrete` a `Fixed-step size` na hodnotu podle frekvence odesílání a přijímání dat (např. 0,001). Viz obrázek 3

2. Kliknutím na šipku pod tlačítkem `Run` zapnout `Simulation Pacing`, viz obrázky 4 a 5.

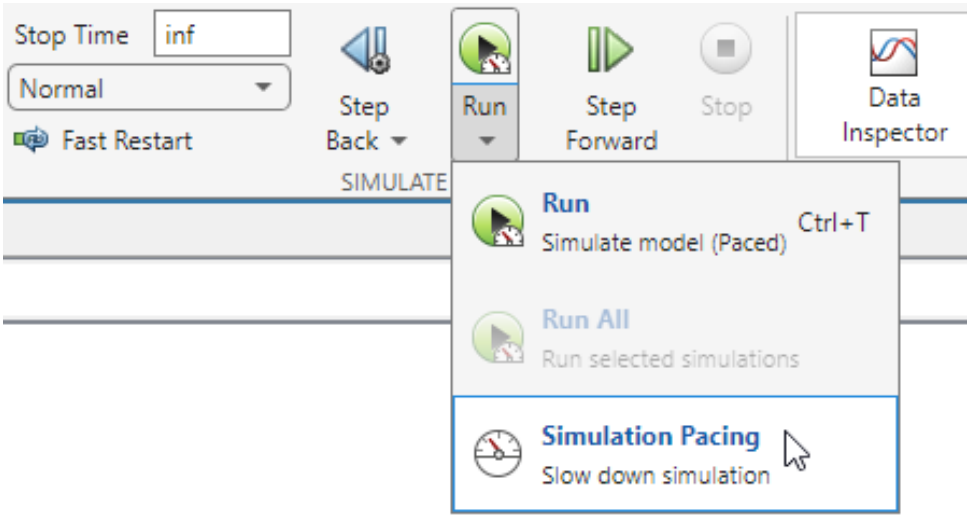
3. V modelu vytvořit blok `Serial Configuration`, ve kterém nastavit COM port, zjištěný v kapitole 3.7.1 a ostatní parametry podle obrázku 6.

Po tomto nastavení je možné použít bloky `Serial Send` a `Serial Receive` pro odesílání a přijímání dat. Informace k použití bloků naleznete v oficiální dokumentaci [https://www.mathworks.com/help/instrument/direct-interface-communication-in-simulink.html].

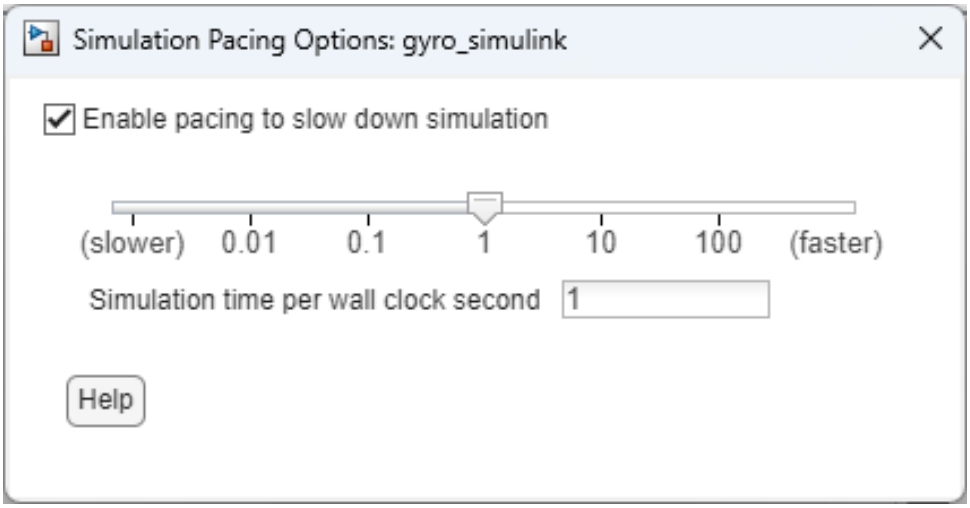
Příklad použití sériové komunikace pro nastavení konstant PID regulátoru v kostce a zobrazení aktuálních složek PID je na obrázku 7. V tomto příkladu se hodnoty odesílají a přijímají v binárním formátu, je nastavena hlavička pro odesílání a přijímaní a data jsou typu single (4 byty). Data se přijímají každé 0,01 sekundy, což je nastaveno v bloku `Serial Send` v kolonce `Block sample time`. Data se odesílají každou sekundu, což je nastaveno v kolonkách `Block sample time` tří bloků `Constant`. Nastavený `Fixed-step size` řešiče by měl být menší než `Block sample time`.



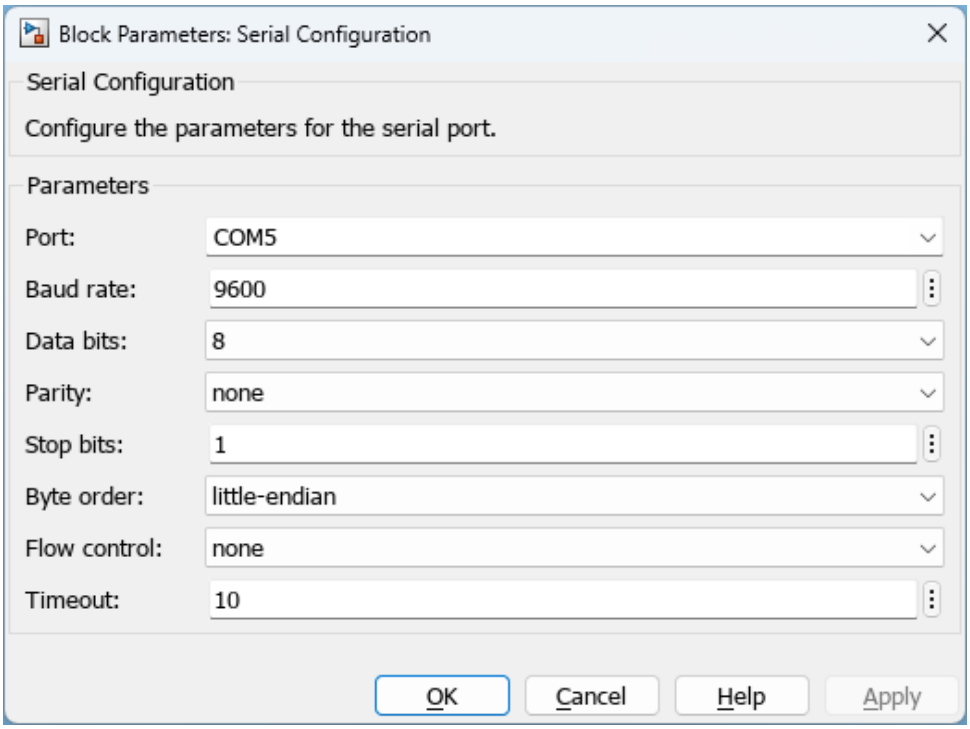
Obrázek 3: Nastavení řešiče Simulinku.



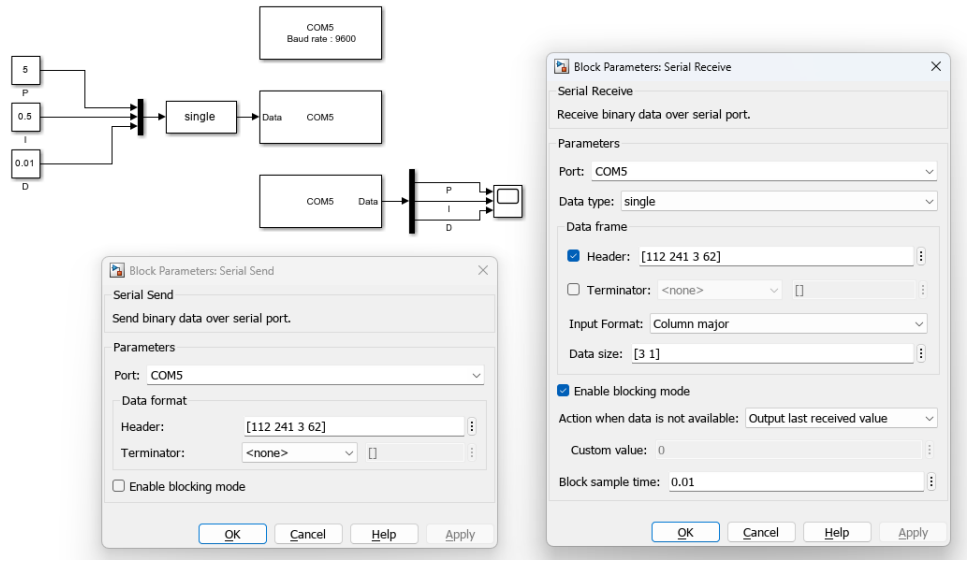
Obrázek 4: Umístění nastavení Simulation pacing v Simulinku.



Obrázek 5: Nastavení Simulation Pacing.



Obrázek 6: Nastavení bloku Serial Configuration v Simulinku.



Obrázek 7: Příklad použití sériové komunikace v Simulinku.

4 NXT senzory

4.1 Tlačítko

class **TouchSensor**(*port*)

Informace o stavu stisknutí NXT tlačítka.

Parametry: **port** (Port) – Port, ke kterému je senzor připojen.
Inicializace: `robot.init_sensor(sensor_type=Sensor.TOUCH,port=Port)`.
Deinicializace: `robot.deinit_sensor(sensor_type=Sensor.TOUCH,port=Port)`.
Přístup: `robot.sensors.touch[Port]`.

pressed()

Navrátí stav tlačítka.

Vrací: True pokud je tlačítko stisknuté, False pokud není.
Typ: bool

```
# Importováni konstant typů senzorů a portů
from lib.robot_consts import Sensor, Port

# Inicializace NXT tlačítka na senzorovém portu 1
robot.init_sensor(sensor_type=Sensor.TOUCH, port=Port.S1)

# Zjištění stavu stisknutí tlačítka
touch_pressed = robot.sensors.touch[Port.S1].pressed()
```

Kód 7: Použití NXT tlačítka.

4.2 Světelný senzor

class **LightSensor**(*port*)

Měření intenzity dopadajícího světla NXT světelného senzoru.

Měření je umožněno dvěma módy – manuální, při kterém je intenzita změřena na požádání a kontinuální, při kterém je intenzita měřena periodicky a na požádání je vrácena poslední změřená hodnota. Kontinuální mód dále poskytuje režimy neblikající a blikající. Při nastavení neblikajícího režimu je intenzita měřena ve stavu LED nastaveném uživatelem. V blikajícím režimu je intenzita měřena při vypnutém i zapnutém stavu LED. Změna stavu LED je prováděna automaticky a uživateli je na požádání k dispozici poslední změřená intenzita pro oba stavy LED.

AD převodník vzorkuje signál s periodou 1 ms. Intenzita dopadajícího světla je získána měřením napětí na senzoru. Při změně stavu LED se toto napětí ustálí za přibližně 10 ms.

Parametry: **port** (Port) – Port, ke kterému je senzor připojen.
Inicializace: `robot.init_sensor(sensor_type=Sensor.LIGHT,port=Port)`
Deinicializace: `robot.deinit_sensor(sensor_type=Sensor.LIGHT,port=Port)`
Přístup: `robot.sensors.light[Port]`

on()

Zapne LED.

off()

Vypne LED.

togggle()

Změní stav LED.

intensity(*pin_on*)

V manuálním módu změří intenzitu světla a navrátí ji. V kontinuálním módu navrátí poslední změřenou intenzitu.

Parametry: **pin_on** (bool) – Volba navrácené změřené intenzity dopadajícího světla. V manuálním módu a v kontinuálním módu s neblikajícím režimem na hodnotě nezáleží. V kontinuálním módu s blikajícím režimem navrátí pro True intenzitu změřenou při zapnuté LED a pro False při vypnuté LED.

Vrací: Intenzita dopadajícího světla na senzor. 0 pro největší intenzitu, 32768 pro nejmenší.

Typ: int (0–32768)

set_continuous(*period_us, wait_us*)

Nastaví měření do kontinuálního módu.

Parametry:
- **period_us** (int) – Doba mezi začátkem konverze analogové hodnoty a vyčtením konvertované digitální hodnoty z AD převodníku. Doporučená minimální hodnota je 1100.
- **wait_us** (int) – Doba čekání před měřením po změně stavu LED v mikrosekundách. Doporučená minimální hodnota je 10000.

stop_continuous()

Nastaví měření do manuálního módu.

set_switching()

Nastaví měření kontinuálního módu do blikajícího režimu.

stop_switching()

Nastaví měření kontinuálního módu do neblikajícího režimu.

```
# Importováni konstant typů senzorů a portů
from lib.robot_consts import Sensor, Port

# Inicializace NXT světelného senzoru na senzorovém portu 1
robot.init_sensor(sensor_type=Sensor.LIGHT, port=Port.S1)

# Zapnutí LED senzoru
robot.sensors.light[Port.S1].on()
# Změření intenzity světla
light_intensity = robot.sensors.light[Port.S1].intensity()
```

Kód 8: Použití NXT světelného senzoru.

4.3 Ultrazvukový senzor

class **UltrasonicSensor**()

Měření vzdálenosti objektu od senzoru v rozmezí 0–255 cm s přesností ±3 cm.

Parametry: **port** (Port) – Port, ke kterému je senzor připojen.
Inicializace: `robot.init_sensor(sensor_type=Sensor.ULTRA)`
Deinicializace: `robot.deinit_sensor(sensor_type=Sensor.ULTRA)`
Přístup: `robot.sensors.ultra`

distance()

Změří vzdálenost a navrátí ji.

Vrací: Vzdálenost objektu od senzoru.
Typ: int (0-255)

```
# Importováni konstant typů senzorů
from lib.robot_consts import Sensor

# Inicializace NXT ultrazvukového senzoru
robot.init_sensor(sensor_type=Sensor.ULTRA)

# Změření vzdálenosti
distance = robot.sensors.ultra.distance()
```

Kód 9: Použití NXT ultrazvukového senzoru.

4.4 Zvukový senzor

class **SoundSensor**(*port*)

Měření intenzity zvuku NXT zvukového senzoru.

Parametry:	**port** (Port) – Port, ke kterému je senzor připojen.
Inicializace:	`robot.init_sensor(sensor_type=Sensor.SOUND,port=Port)`
Deinicializace:	`robot.deinit_sensor(sensor_type=Sensor.SOUND,port=Port)`
Přístup:	`robot.sensors.sound[Port]`

intensity()

Změří intenzitu zvuku a navrátí ji.

Vrací:	Intenzita zvuku. 0 pro největší intenzitu, 32768 pro nejmenší.
Typ:	int (0–32768)

```
# Importováni konstant typů senzorů a portů
from lib.robot_consts import Sensor, Port

# Inicializace NXT zvukového senzoru na senzorovém portu 1
robot.init_sensor(sensor_type=Sensor.SOUND, port=Port.S1)

# Změření hladiny zvuku
sound_intensity = robot.sensors.sound[Port.S1].intensity()
```

Kód 10: Použití NXT zvukového senzoru.

5 Servomotor

Motory ze sady NXT a EV3 jsou vnitřními parametry identické, liší se pouze vzhledem a plastovou konstrukcí.

class **Motor**(*port*)

Na motorech se nachází rotační enkodéry umožňující snímání polohy natočení motoru a aproximaci rychlosti otáčení.

Poskytuje tři módy řízení – výkon, poloha, rychlost.

V módu řízení výkonu uživatel přímo nastavuje požadovaný výkon na motoru v rozmezí -100–100 %.

V módu řízení polohy uživatel volí požadovanou polohu natočení motoru ve °. Regulace je zprostředkována PID regulátorem s nastavitelnými konstantami.

V módu řízení rychlosti uživatel volí požadovanou rychlost otáčení motoru ve °/s. Maximální rychlost při nezatíženém motoru je přibližně 9500 °/s. Regulace je zprostředkována PID regulátorem s nastavitelnými konstantami.

Parametry:	**port** (Port) – Port, ke kterému je motor připojen.
Inicializace:	`robot.init_motor(port=Port)`
Deinicializace:	`robot.deinit_motor(port=Port)`
Přístup:	`robot.motor[Port]`

Parametry: **port** (Port) – Port, ke kterému je motor připojen.

set_power(*power*)

Nastaví výkon na motoru.

Parametry: **power** (float) – Požadovaný výkon na motoru v rozmezí -100–100 %.

init_encoder()

Inicializuje enkodér s měřením polohy a rychlosti motoru.

deinit_encoder()

Denicializuje enkodér s měřením polohy a rychlosti motoru.

position()

Navrátí polohu motoru.

Vrací: Poloha motoru.

Typ: úhel: °

speed()

Navrátí polohu motoru.

Vrací: Rychlost motoru.

Typ: úhlová rychlost: °/s

init_regulator()

Inicializuje regulátor motoru.

deinit_regulator()

Denicializuje regulátor motoru.

set_regulator_position(*position*)

Nastaví motor do módu řízení polohy.

Parametry: **position** (int) – Požadovaná konečná poloha motoru ve °.

regulator_pos_set_consts(*p, i, d*)

Nastaví PID konstanty regulátoru polohy.

Parametry:
- **p** (float) – Konstanta proporcionální složky.
- **i** (float) – Konstanta integrační složky.
- **d** (float) – Konstanta derivační složky.

set_regulator_speed(*speed*)

Nastaví motor do módu řízení polohy.

Parametry: **speed** (int) – Požadovaná konečná rychlost motoru ve °/s.

regulator_speed_set_consts(*p, i, d*)

Nastaví PID konstanty regulátoru rychlosti.

Parametry:
- **p** (float) – Konstanta proporcionální složky.
- **i** (float) – Konstanta integrační složky.
- **d** (float) – Konstanta derivační složky.

regulator_error()

Navrátí aktuální regulační odchylku.

Vrací: Regulační odchylka.

Typ:
- **mód řízení polohy** – poloha: °
- **mód řízení rychlosti** – úhlová rychlost: °/s

regulator_power()

Navrátí aktuální akční zásah regulátoru.

Vrací: Akční zásah.

Typ: výkon (-100–100 %)

```
# Importováni konstant portů
from lib.robot_consts import Port

# Inicializace motorů na motorových portech 1 a 2
robot.init_motor(Port.M1)
robot.init_motor(Port.M2)

# Nastavení výkonu motorů
robot.motors[Port.M1].set_power(50)
robot.motors[Port.M2].set_power(-20)

# Zjištění aktuální pozice a rychlosti motorů
pos1 = robot.motors[Port.M1].position()
pos2 = robot.motors[Port.M2].position()
speed1 = robot.motors[Port.M1].speed()
speed2 = robot.motors[Port.M2].speed()
```

Kód 11: Použití motorů.

6 Parametry a konstanty

6.1 Sensor

class **Sensor**

Typy senzorů.

NO_SENSOR

Žádný senzor.

LIGHT

NXT světelný senzor.

ULTRA

NXT ultrazvukový senzor vzdálenosti.

TOUCH

NXT tlačítko.

SOUND

NXT zvukový senzor.

GYRO

ICM20608-G gyroskop a akcelerometr.

Příklad použití motorů je uveden

6.2 Port

class **Port**

Porty motorů a senzorů na kostce.

M1

Port motoru 1.

M2

Port motoru 2.

M3

Port motoru 3.

M4

Port motoru 4.

S1

Port senzoru 1.

S2

Port senzoru 2.

S3

Port senzoru 3.

S4

Port senzoru 4.

6.3 Button

class **Button**

Tlačítka na kostce.

POWER

Tlačítko zapnout.

LEFT

Tlačítko doleva.

RIGHT

Tlačítko doprava.

OK

Tlačítko OK.

UP

Tlačítko nahoru.

DOWN

Tlačítko dolů.

6.4 Light

class **Light**

Změřená hodnota při vypnuté a zapnuté LED světelného senzoru.

OFF

Změřená hodnota při vypnuté LED.

ON

Změřená hodnota při zapnuté LED.

6.5 GyroAcc

class **GyroAcc**

Změřená hodnota gyroskopu a akcelerometru ICM20608-G v osách x, y, z a pin přerušení senzoru.

AX

Zrychlení z akcelerometru v ose x.

AY

Zrychlení z akcelerometru v ose y.

AZ

Zrychlení z akcelerometru v ose z.

GX

Úhlová rychlost z gyroskopu v ose x.

GY

Úhlová rychlost z gyroskopu v ose y.

GZ

Úhlová rychlost z gyroskopu v ose z.

IRQ_PIN

Pin přerušení senzoru ICM20608-G.