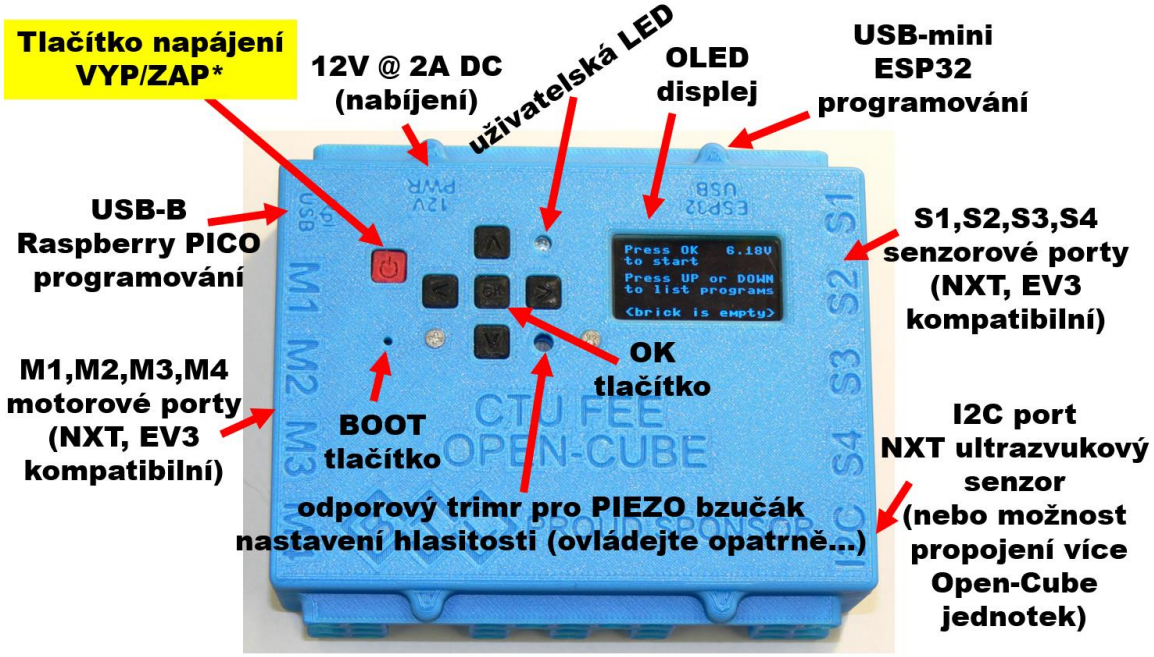
OPEN-CUBE ZJEDNODUŠENÝ NÁVOD

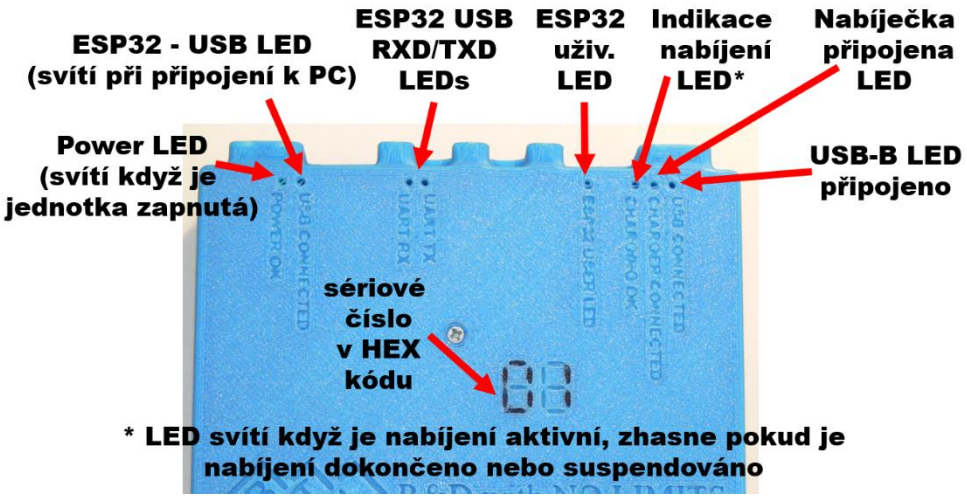
Pro více informací navštivte stránku: open-cube.fel.cvut.cz

Zapněte jednotku Open-Cube a připojte ji k počítači pomocí USB-B (najděte číslo COMportu ve správci zařízení)

1. Použijte program Thonny k vytvoření a ladění programů v Micropythonu (vyberte Raspberry Pi PICO jako platformu a správný COMport, jděte do Tools/Options/Interpreter...)
2. Na následující straně najdete seznam nejpoužívanějších / nejužitečnějších příkazů, podrobný a úplný popis najdete na následujících stránkách stejně jako některé tipy ohledně programování v MicroPythonu.



***dlouhý stisk (>5s) tohoto tlačítka vypne Open-Cube (zajištěno elektrickým obvodem, bude fungovat i při zamrznutí programu)**



*** LED svítí když je nabíjení aktivní, zhasne pokud je nabíjení dokončeno nebo suspendováno**

SEZNAM NEJUŽITEČNĚJŠÍCH PŘÍKAZŮ PRO OPEN-CUBE

ČASOVÁNÍ

```
from time import sleep , sleep_ms , sleep_us
sleep(1)          # uspi program na 1 sekundu
sleep_ms(2)       # uspi program na 2 milisekundy
sleep_us(3)       # uspi program na 3 mikrosekundy, podívejte se i na třídu "Timer"
```

MOTORY (stejné pro NXT i EV3), více v kapitole 7

```
from lib. robot_consts import Port      # importovat konstanty
robot.init_motor(Port.M1)               # inicializovat motor
robot.init_motor(Port.M2)               # inicializovat motor
robot.motors[Port.M1].set_power(50)     # nastav výkon na 50%, otáčení po směru hod. ručiček
robot.motors[Port.M2].set_power(-20)    # nastav výkon na 20%, proti směru hod. ručiček
```

SENZORY

NXT DOTYKOVÝ

```
from lib.robot_consts import Sensor, Port                    # importovat konstanty a typy...
robot.init_sensor(sensor_type = Sensor.NXT_TOUCH, port = Port.S1) # inicializuj na portu S1
touch_pressed = robot.sensors.touch[Port.S1].pressed()       #přečti stav tlačítka
```

NXT OPTICKÝ více v kapitole 4.2 (další funkcionality)

```
from lib.robot_consts import Sensor, Port                    # importovat konstanty a typy...
robot.init_sensor(sensor_type = Sensor.LIGHT, port = Port.S1)      # inicializuj senzor na portu S1
robot.sensors.light[Port.S1].on()                            # zapni LED
light_intensity = robot.sensors.light[Port.S1].intensity()   # přečti změřenou intenzitu světla
```

NXT MIKROFON

```
from lib.robot_consts import Sensor, Port                    # importovat konstanty a typy...
robot.init_sensor(sensor_type = Sensor.NXT_SOUND, port = Port.S1) # inicializ. senzor na portu S1
sound_intensity = robot.sensors.sound[Port.S1].intensity()   # přečti intenzitu zvuku
```

NXT ULTRAZVUKOVÝ DÁLKOMĚR

```
from lib.robot_consts import Sensor                          # importovat konstanty a typy...
robot.init_sensor(sensor_type = Sensor.NXT_ULTRASONIC)       # inicializuj senzor na portu I2C
distance = robot.sensors.ultra_nxt.distance()                # změř vzdálenost
```

NXT GYROSKOP – aktuálně není implementováno (I2C rozhraní, vyrobeno firmou HiTechnic)

EV3 DOTYKOVÝ

```
from lib.robot_consts import Sensor, Port                    # importovat konstanty a typy...
robot.init_sensor(sensor_type = Sensor.EV3_TOUCH, port = Port.S1) # inicializ. senzor na portu S1
touch_pressed = robot.sensors.touch[Port.S1].pressed()       #přečti stav tlačítka
```

EV3 OPTICKÝ další funkce v kapitole 5.2 (například snímání barvy)

```
from lib.robot_consts import Sensor, Port                    # importovat konstanty a typy...
robot.init_sensor(sensor_type = Sensor.EV3_COLOR, port = Port.S1) # inicializ. senzor na portu S1
reflection = robot.sensors.light[Port.S1].reflection()       # změř odraženou intenzitu světla
```

EV3 ULTRAZVUKOVÝ DÁLKOMĚR

```
from lib.robot_consts import Sensor, Port                    # importovat konstanty a typy...
robot.init_sensor(sensor_type = Sensor.EV3_ULTRA, port = Port.S1) # inicializ. senzor na portu S1
distance = robot.sensors.ultrasonic[Port.S1].distance()      # přečti změřenou vzdálenost
```

EV3 GYROSKOP *(jednoosý)*

```
from lib.robot_consts import Sensor, Port                    # importovat konstanty a typy...
robot.init_sensor(sensor_type = Sensor.EV3_GYRO, port = Port.S1) # inicializuj senzor na portu S1
robot.sensors.gyro[Port.S1].reset_angle(0)                   # resetuj úhel
(angle, speed ) = robot.sensors.gyro[Port.S1].angle_and_speed()  # změř úhlovou rychlost a pozici
```

O-C OPTICKÝ *více a další módy v kapitole 6.1 (snímaní barev, modrý a zelený mód měření)*

```
from lib.robot_consts import Sensor, Port                    # importovat konstanty a typy...
robot.init_sensor(sensor_type = Sensor.OC_COLOR, port = Port.S1) # inicializuj senzor na portu S1
reflection = robot.sensors.light[Port.S1].reflection()       # změř odraženou intenzitu světla
```

O-C LASEROVÝ DÁLKOMĚR více v kapitole 6.2

```
from lib.robot_consts import Sensor, Port                    # importovat konstanty a typy...
robot.init_sensor(sensor_type = Sensor.OC_LASER, port = Port.S1) # inicializuj senzor na portu S1
distance = robot.sensors.laser[Port.S1].distance()           # změří vzdálenost
```

O-C ULTRAZVUKOVÝ DALKOMĚR vice v kapitole 6.3

```
from lib.robot_consts import Sensor, Port                    # importovat konstanty a typy...
robot.init_sensor( sensor_type = Sensor.OC_ULTRASONIC, port = Port.S1)  #inicializovat senzor
distance = robot.sensors.ultrasonic[Port.S1].distance()      # změřit vzdálenost
```

O-C AHRS see chapter 6.4

```
from lib.robot_consts import Sensor, Port                    # importovat konstanty a typy...
robot.init_sensor(sensor_type = Sensor.OC_GYRO, port = Port.S1)     # inicializovat senzor
euler_angles = robot.sensors.gyro[Port.S1].euler_angles()    # přečíst data  (další možné výstupy a kalibrace senzoru v kapitole 6.4)
```

INTERNÍ GYROSKOP *v kapitole 3.6 příklad použití přerušení pro vyčítání senzoru*

```
robot.init_sensor(sensor_type = Sensor.GYRO_ACC)
a_g_data = (0, 0, 0, 0, 0, 0)                                # ax:-, ay:-, az:-, gx:°/s, gy:°/s, gz:°/s
a_g_data = robot.sensors.gyro_acc.read_value()               # změř zrychlení a úhlové rychlosti
```

DALŠÍ FUNKCIONALITY

DISPLEJ *kompletní seznam funkcí pro ovládání displeje v kapitole 3.5*

```
robot.display.fill(0)                                        # vymaž displej
robot.display.text("test message",0,0,1)                     # vytiskni text, na danou pozici
x, y = 1.23, 56.789                                          # inicializuj proměnné
robot.display.text(f"x={x:.2 f}, y: {y:.2f}", 0, 10, 1)      # vytiskni dvě float čísla
robot.display.show()                                         # zobraz data z paměti driveru displeje
```

ZVUK - PIEZO

```
robot.buzzer.set_freq_duty(4000,50)                          # 4000 Hz, 50 % střída
robot.buzzer.off()                                           # vypni generování PWM
```

TLAČÍTKA

```
from lib. robot_consts import Button                         # importovat konstanty a typy...
buttons = robot.buttons.pressed()                            # přečti stav všech tlačítek
if buttons[Button.LEFT]: break                               # použij výsledek třeba v if podmínce
# další tlačítka: Button.LEFT, Button.RIGHT, Button.OK, Button.UP, Button.DOWN
```

LED *(červená, na předním panelu Open-Cube)*

```
robot.led.on()                                               # zapni LED
robot.led.off()                                              # vypni LED
```

AKUMULÁTOR
voltage = robot.battery.voltage() *#přečti napětí interního O-C Li-ion akumulátoru*

I2C MASTER, I2C SLAVE (řízení Open-Cube nebo obecné použití pro komunikaci), aktuálně není implementováno

BLUETOOTH sériová komunikace **&** **Wi-Fi** přístupový bod – více v kapitole 3.7.2

MicroPython také podporuje: if x < 9: while x < 9: for y in range(0, 9):
print("debug message!") *# vytiskni ladící zprávu do konzole programu Thonny*
def add(number1, number2): return number1 + number2 *# definice funkce*
add(1, 2) *# použití funkce*

send_data = 0.1 *#vytvoř proměnnou*
buffer = "Values: " + str(send_data) + "," + str(-send_data) + ";" *# Values: 0.1,-0.1;*
print(buffer) *#Vytiskni string do PC*
or
robot.esp.bt_write(buffer) *# nebo ho pošli přes Bluetooth*

Chcete vizualizovat hodnoty proměnných v závislosti na čase? Hledejte program "*DataPlotter*" *https://github.com/jirimaier/DataPlotter* šikovný program vytvořený ČVUT-FEL studentem Jiřím Maierem – funguje přes sériový port (USB-serial nebo Bluetooth-serial)

send_data = 0.1
buffer = "$$P"+str(send_data)+","+str(-send_data)+";"
robot.esp.bt_write(buffer) #další info v *bt_serial.py v kostce nebo na gitlabu*
"$$P" je specifická hlavička pro přidání jednoho bodu do grafu, najdete v dokumentaci *DataPlotter*u

Open-Cube

Katedra měření ČVUT FEL

2024

Obsah

1 Instalace firmwaru **4**

2 Editor kódu **4**
2.1 Nastavení editoru . . . 4
2.2 Nahrání kódu do kostky . . . 4
2.3 Spuštění programu samostatně . . . 5

3 Robot **6**
3.1 Tlačítka . . . 7
3.2 LED . . . 8
3.3 Bzučák . . . 9
3.4 Baterie . . . 9
3.5 Displej . . . 10
3.6 Gyroskop a akcelerometr . . . 14
3.7 ESP32 komunikace . . . 15
3.7.1 Bluetooth propojení ESP32 s jiným zařízením . . . 16
3.7.2 Kostka . . . 18
3.7.3 Serial Bluetooth Terminal . . . 21
3.7.4 Matlab . . . 21
3.7.5 Simulink . . . 22

4 NXT senzory **26**
4.1 Tlačítko . . . 26
4.2 Světelný senzor . . . 26
4.3 Ultrazvukový dálkoměr . . . 28
4.4 Zvukový senzor . . . 29

5 EV3 senzory **30**
5.1 Tlačítko . . . 30
5.2 Světelný senzor . . . 30
5.3 Ultrazvukový dálkoměr . . . 32
5.4 Infračervený senzor . . . 32
5.5 Gyroskop . . . 33

6 Open-Cube senzory **35**
6.1 Světelný senzor . . . 35
6.2 Laserový dálkoměr . . . 37
6.3 Ultrazvukový dálkoměr . . . 38
6.4 AHRS senzor . . . 38

7 Servomotor **40**

8 Parametry a konstanty **43**
8.1 Sensor . . . 43
8.2 Port . . . 44
8.3 Button . . . 44
8.4 Light . . . 45
8.5 GyroAcc . . . 45
8.6 Color . . . 46

9 Užitečné funkce MicroPythonu **47**
9.1 Výpis informací . . . 47
9.2 Generování náhodných čísel . . . 47
9.3 Čas . . . 48
9.4 Manipulace s binárními daty . . . 48
9.5 Vícejádrové programování . . . 49

10 Použité pojmy Pythonu **50**

Seznam kódů

1 Spuštění programu samostatně. 5
2 Použití globální třídy robot. 7
3 Zjištění stavu tlačítek. 8
4 Použití LED. 9
5 Použití bzučáku. 9
6 Zjištění napětí baterií. 10
7 Použití displeje. 13
8 Čtení dat z gyroskopu s využitím irq. 14
9 Komunikace mezi mikroprocesorem a ESP32. 20
10 Komunikace s kostkou z Matlabu. 21
11 Použití NXT tlačítka. 26
12 Použití NXT světelného senzoru. 28
13 Použití NXT ultrazvukového senzoru. 28
14 Použití NXT zvukového senzoru. 29
15 Použití EV3 tlačítka. 30
16 Použití EV3 světelného senzoru. 31
17 Použití EV3 ultrazvukového dálkoměru. 32
18 Použití EV3 infračerveného senzoru. 33
19 Použití EV3 gyroskopu. 34
20 Použití Open-Cube světelného senzoru. 36
21 Použití Open-Cube laserového dálkoměru. 37
22 Použití Open-Cube ultrazvukového dálkoměru. 38
23 Použití Open-Cube AHRS senzoru. 39
24 Použití motorů. 42
25 Výpis informací. 47
26 Použití knihovny random. 47
27 Použití knihovny time. 48
28 Použití třídy Timer. 48
29 Použití knihovny struct. 49
30 Použití knihovny _thread. 49

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/], který je dostupný pro Windows, Mac i Linux.

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.

2.3 Spuštění programu samostatně

Pro urychlení testování lze uživatelský program spustit i samostatně bez spuštění hlavního programu:

```
# Importování třídy Robot
from lib.robot import *

# Pokud je program spuštěn z menu, nic se nestane
# Pokud je spuštěn samostatně, inicializuje se proměnná robot
global robot
try:
  robot
except:
  robot = Robot()
.
.
.
# Při ukončení programu lze resetovat kostku a přejít tak opět do menu
import machine
machine.reset()
```

Kód 1: Spuštění programu samostatně.

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
├── led
├── motors[4]
└── sensors
    ├── ultra_nxt
    ├── gyro_acc
    ├── touch[4]
    ├── light[4]
    ├── sound[4]
    ├── ultrasonic[4]
    ├── gyro[4]
    ├── infrared[4]
    └── laser[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*)

Deinicializace 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*)

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

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

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.

```
# Importování funkce pro uspání programu
from time import sleep
# Importování 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 2: Použití globální třídy robot.

3.1 Tlačítka

class **Buttons**()

Informace o stavu stisknutí tlačítek na čelním panelu kostky. Pokud se po krátkém stisknutí tlačítka POWER kostka sama nevypne, lze ji vypnout dlouhým stisknutím tohoto tlačítka, čímž dojde k odpojení kostky od napájení.

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

pressed()

Navrátí stav tlačítek. Pro zjištění stavu konkrétního 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 uspáni programu
from time import sleep
# Importování konstant tlačítek na kostce
from lib.robot_consts import Button

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 3: Zjištění stavu tlačítek.

3.2 LED

class **Led()**

Zapínání a vypínání červené LED na čelním panelu kostky.

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

on()

Zapne LED.

off()

Vypne LED.

toggle()

Změní stav LED.

```
# Zapnutí LED
robot.led.on()
# Vypnutí LED
robot.led.off()
```

Kód 4: Použití LED.

3.3 Bzučák

class **Buzzer**()

Ovládání bzučáku v kostce. Bzučák má nejvyšší hlasitost při frekvenci přibližně 4500 Hz z důvodu nerovnoměrné frekvenční charakteristiky piezo měniče. Hlasitost lze dále nastavit otočením odporového trimru na čelním panelu kostky.

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

set_freq_duty(*freq, duty*)

Nastaví PWM ovládající bzučák na požadovanou frekvenci a střídu.

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

off()

Vypne bzučák.

```
# Zapnutí bzučáku na frekvenci 4000 Hz se střídou 50 %
robot.buzzer.set_freq_duty(4000, 50)
# Změna frekvence bzučáku na 1000 Hz
robot.buzzer.set_freq_duty(1000, 50)
# Vypnutí bzučáku
robot.buzzer.off()
```

Kód 5: Použití bzučáku.

3.4 Baterie

class **Battery**()

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

Open-Cube je napájena dvěma Li-ion akumulátory (nominální napětí jednoho článku je 3,7 V, maximální nabíjecí 4,2 V, vybití pod 3 V již zkracuje životnost článku). V případě plného nabití lze tedy naměřit 8,2–8,4 V. Akumulátory by neměly být vybíjeny pod 6 V. Kostka se softwarově vypne při poklesu napětí pod 6,5 V, hardwarová podpěťová ochrana by měla články odpojit při poklesu pod 5,8 V.

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.

```
# Zjistí napětí na bateriích ve voltech
voltage = robot.battery.voltage()
```

Kód 6: Zjištění napětí baterií.

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. Displej má rozlišení 128x64 pixelů.

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 vertikální 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** – Souřadnice 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** – Šířka a výška 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** – Šířka a výška 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** (str) – Text.
- **x**, **y** – Souřadnice levého horního rohu začátku textu.
- **color** (0/1) – Barva – 0 černá, 1 bílá.

centered_text(*s, y, color=1*)

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

Parametry:
- **s** (str) – Text.
- **x** – Souřadnice horní hranice 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 výška 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** (str) – 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 výška 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** (str) – 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** (str) – 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 výška grafu.
- **xlo**, **xhi** (int) – Dolní a horní omezení osy x.
- **ylo**, **yhi** (int) – Dolní a horní omezení osy y.
- **label** (str) – Popisek grafu.
- **redraw** (bool) – Překreslení grafu.

```
# Vymazání displeje
robot.display.fill(0)
# Zapsání textu do frame bufferu do levého horního rohu
robot.display.text("test message", 0, 0, 1)

```

```
# Zapsání čísel s přesností na 2 desetinná místa do frame bufferu na
    další řádek
x, y = 1.23, 56.789
robot.display.text(f"x={x:.2f}, y: {y:.2f}", 0, 10, 1)
# Zobrazení frame bufferu na displeji
robot.display.show()

# Funkce, které vrací hodnotu pro snížení problikávání grafu by měly
    být ve smyčce volány následovně
redraw = True
while(True):
  redraw = robot.display.draw_dial(50,128/2,40,20,0,100,10,270,
    "Speedometer",redraw)
```

Kód 7: Použití displeje.

3.6 Gyroskop a akcelerometr

class **ICM42688**()

Integrovaný gyroskop a akcelerometr.

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

Inicializace: `robot.init_sensor(sensor_type=Sensor.GYRO_ACC)`
Deinicializace: `robot.deinit_sensor(sensor_type=Sensor.GYRO_ACC)`
Přístup: `robot.sensors.gyro_acc`

read_value()

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

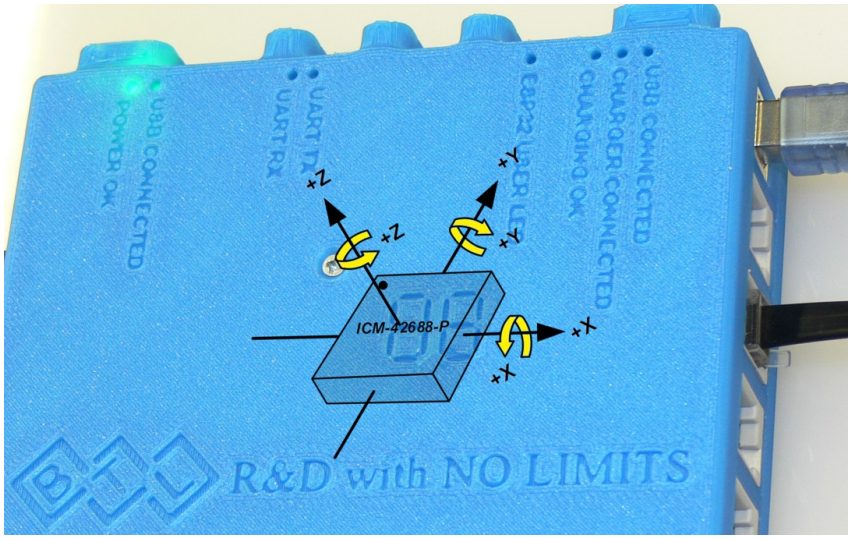
Vrací: Tuple zrychlení a úhlových rychlostí v osách x, y, z.
Typ: (a_x: -, a_y: -, a_z: -, g_x: °/s, g_y: °/s, g_z: °/s)

Příklad čtení dat z gyroskopu s využitím irq (pinu přerušení):

```
# 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_ACC)
  # 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)
```



Obrázek 1: Orientace os integrovaného gyroskopu a akcelerometru.

```
17
18    # Smyčka čekající na ukončení programu
19    while True:
20      # Zobrazení posledních hodnot zrychlení a úhlové rychlosti
21      print("Acc:",  a_g_data[GyroAcc.AX],
22                     a_g_data[GyroAcc.AY],
23                     a_g_data[GyroAcc.AZ],
24            "Gyro:", a_g_data[GyroAcc.GX],
25                     a_g_data[GyroAcc.GY],
26                     a_g_data[GyroAcc.GZ],)
27      # Získání stavu tlačítek
28      buttons = robot.buttons.pressed()
29      # Ukončení programu, pokud je levé tlačítko stisknuté
30      if buttons[Button.LEFT]:
31        break
32      # Uspání programu na 1 sekundu
33      sleep(1)
34
35    # Zrušení callbacku před deinicializací senzoru
36    icm_irq_pin.irq(handler=None)
37    # Deinicializace senzoru
38    robot.deinit_sensor(sensor_type=Sensor.GYRO_ACC)
39
40  def callback(p):
41    global robot, a_g_data
42    # Přečtení nových hodnot ze senzoru
43    a_g_data = robot.sensors.gyro_acc.read_value()
44
45  # Spuštění programu
46  main()
```

Kód 8: Čtení dat z gyroskopu s využitím irq.

3.7 ESP32 komunikace

ESP32 komunikuje s kostkou (RP2040) pomocí sériové linky. S dalším zařízením může komunikovat pomocí Wi-Fi nebo Bluetooth. ESP je programovatelné po připojení USB

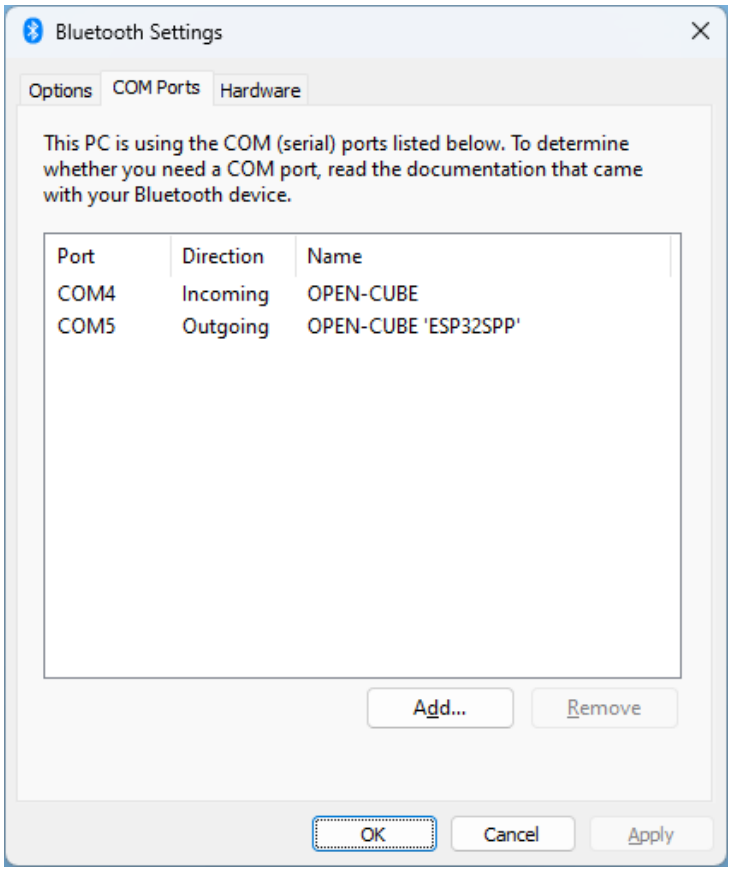
kabelem na vrchní straně kostky. Výchozí firmware ESP (dostupný z Open-Cube repozitáře [https://gitlab.fel.cvut.cz/open-cube/firmware/-/tree/main/ESP]) umožňuje v Bluetooth režimu přeposílání dat z kostky přes Bluetooth virtuální COM port počítači nebo mobilu a ve Wi-Fi režimu funguje jako access point s web serverem s univerzálními ovládacími prvky (tlačítka, přepínače, indikátory).

3.7.1 Bluetooth propojení ESP32 s jiným zařízením

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

1. Pro spárování musí být kostka zapnutá v menu.
2. Ve Windows Bluetooth nastavení zvolte Přidat nové zařízení, dále Bluetooth a spárujte počítač s kostkou. Viz obrázek 2.
3. Potvrďte pin na kostce a na počítači.
4. Pro zjištění čísla COM portu pokračujte do pokročilých nastavení Bluetooth. Požadovaný COM port má název obsahující ESP32SPP a směr Outgoing. Viz obrázek 3. Tento COM port lze použít pro přijímání i odesílání dat.

Obrázek 2: Spárování počítače s kostkou.



Obrázek 3: Zjištění čísla COM portu.

3.7.2 Kostka

class **ESP**()

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

ESP může pracovat ve dvou režimech – Bluetooth a Wi-Fi access point.

1. V případě Bluetooth režimu se data 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.
2. Wi-Fi access point režim s web serverem poskytuje univerzální ovládací prvky – tlačítka, přepínače a indikátory.

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

reset()

Resetuje ESP a nastaví Bluetooth režim.

bt()

Nastaví Bluetooth režim.

wifi()

Nastaví Wi-Fi access point režim. Výchozí adresa web serveru je 192.168.4.1. Indexace prvků web serveru je zobrazena na obrázku 4.

set_name(*name*)

Nastaví jméno ESP pro Bluetooth a Wi-Fi. Výchozí jméno je "OPEN-CUBE".

Parametry: **name** (str) – Jméno ESP.

set_password(*password*)

Nastaví heslo ESP pro Wi-Fi. Heslo musí mít alespoň 8 znaků. Výchozí heslo je "12345678".

Parametry: **password** (str) – Heslo ESP.

bt_read()

Navrátí poslední přijatou zprávu v Bluetooth režimu.

Vrací: Přijatá zpráva. Pokud není nová zpráva k dispozici: None.

Typ:
- ASCII režim: str
- Binární režim: bytes

bt_write(*buff*)

Odešle zprávu v Bluetooth režimu.

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

bt_set_binary(*header, data_size*)

Bluetooth režimu nastaví komunikaci binárního režimu. Pokud je header None, komunikace se nastaví do ASCII režimu.

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

wifi_get_buttons()

Navrátí stav stisknutí 9 tlačítek web serveru.

Vrací: Tuple stavu tlačítek.

Typ: 9x bool

wifi_get_switches()

Navrátí stav 5 přepínačů web serveru.

Vrací: Tuple stavu přepínačů.

Typ: 5x bool

wifi_set_indicators(*indicators*)

Nastaví stav 5 indikátorů web serveru.

Parametry: **indicators** (tuple) – 5x bool stavu indikátorů.

wifi_set_numbers(*numbers*)

Nastaví 6 čísel web serveru.

Parametry: **indicators** (tuple) – 6x float.

wifi_set_buttons_labels(*labels*)

Nastaví popisek 9 tlačítek web serveru. String jednoho tlačítka může mít délku maximálně 9 znaků a prázdný string skryje dané tlačítko.

Parametry: **labels** (tuple) – 9x str popisků tlačítek.

wifi_set_switches_labels(*labels*)

Nastaví popisek 5 přepínačů web serveru. String jednoho přepínače může mít délku maximálně 9 znaků a prázdný string skryje daný přepínač.

Parametry: **labels** (tuple) – 5x str popisků přepínačů.

wifi_set_indicators_labels(*labels*)

Nastaví popisek 5 indikátorů web serveru. String jednoho indikátoru může mít délku maximálně 9 znaků a prázdný string skryje daný indikátor.

Parametry: **labels** (tuple) – 5x str popisků indikátorů.

wifi_set_numbers_labels(*labels*)

Nastaví popisek 6 čísel web serveru. String jednoho čísla může mít délku maximálně 9 znaků a prázdný string skryje dané číslo.

Parametry: **labels** (tuple) – 6x str popisků čísel.

Příklad použití komunikace mezi mikroprocesorem a ESP32:

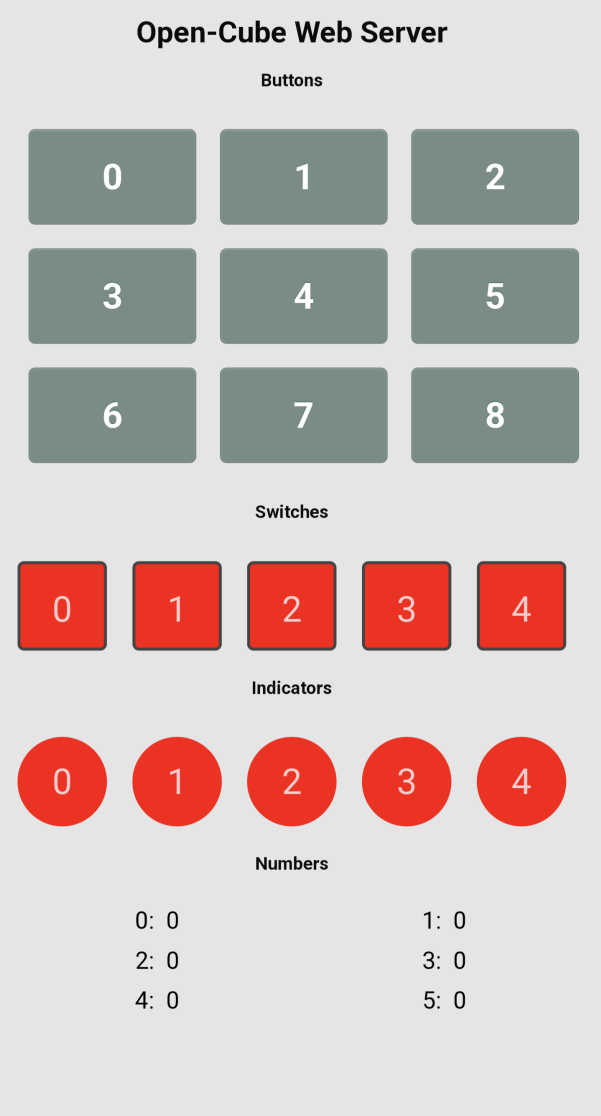
```
# Importování knihovny pro práci s binárními daty
import struct

# Odeslání a přijetí zprávy typu string v režimu ASCII
robot.esp.bt_write("test message")
received_message = robot.esp.bt_read()

# Odeslání a přijetí zprávy o velikosti 8 bytů v binárním režimu
robot.esp.bt_set_binary((112, 241, 3, 62), 8)
# Vytvoření a odeslání zprávy typu bytes se dvěma čísly typu single
write_message = struct.pack("<ff", 521.9, -11.821)
robot.esp.bt_write(write_message)
# Příjem a dekódování zprávy obsahující dvě čísla typu single
received_message2 = robot.esp.bt_read()
(value1, value2) = struct.unpack("<ff", received_message2)

# Přepne ESP do Wi-Fi access point režimu
robot.esp.wifi()
# Nastaví popisky indikátorů
robot.esp.set_indicators_labels(["1", "2", "3", "4", "5"])
# Zjistí stav přepínačů
switches = robot.esp.get_switches()
```

Kód 9: Komunikace mezi mikroprocesorem a ESP32.



Obrázek 4: Indexace prvků ESP web serveru.

3.7.3 Serial Bluetooth Terminal

Pro jednoduchou komunikaci z mobilu lze použít aplikaci Serial Bluetooth Terminal [https://play.google.com/store/apps/details?id=de.kai_morich.serial_bluetooth_terminal].

3.7.4 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/].

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

Kód 10: Komunikace s kostkou z Matlabu.

3.7.5 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:

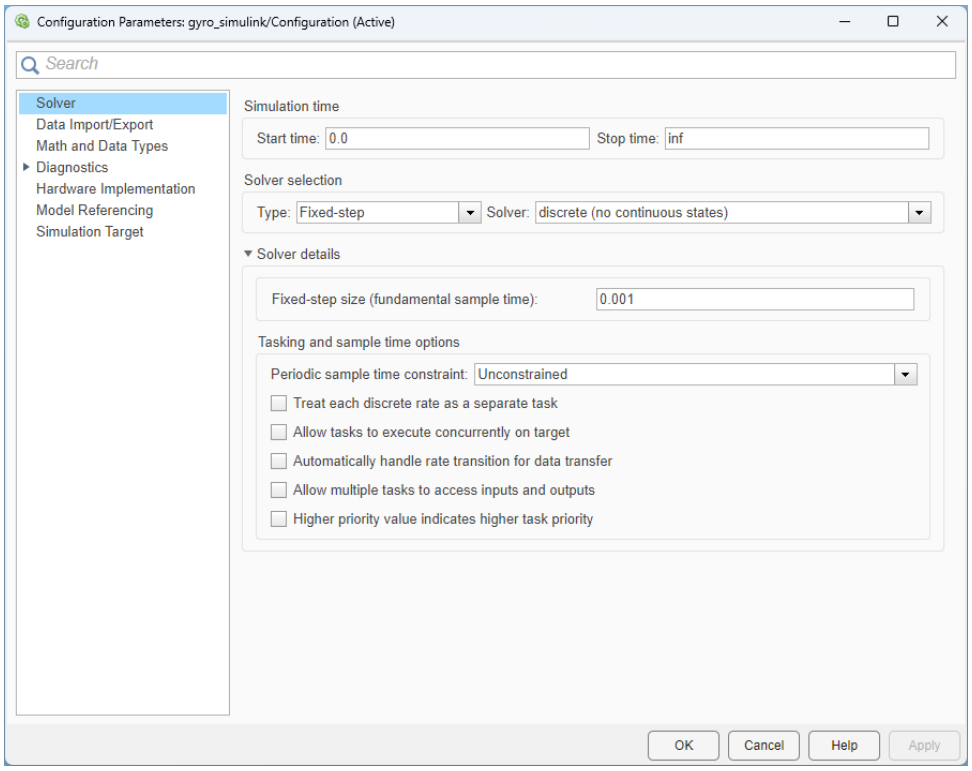
1. Kliknutím na tlačítko v pravém dolním rohu nastavit řešič na `Fixed-Step`, `discrete` a `Fixed-step size` na hodnotu podle frekvence odesílání a přijímání dat (např. 0,001). Viz obrázek 5

2. Kliknutím na šipku pod tlačítkem `Run` zapnout `Simulation Pacing`, viz obrázky 6 a 7.

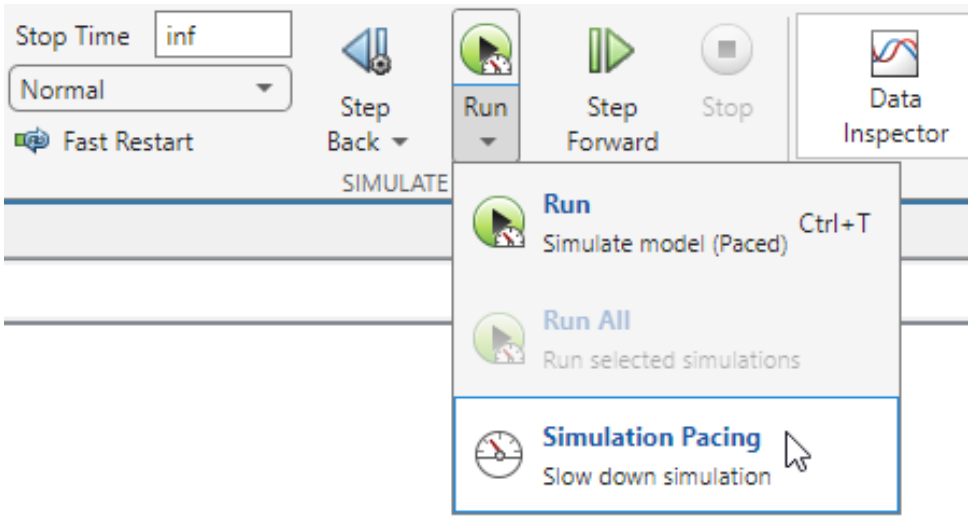
3. V modelu vytvořit blok `Serial Configuration`, ve kterém nastavit COM port, zjištěný v kapitole 3.7.1 a ostatní parametry podle obrázku 8.

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].

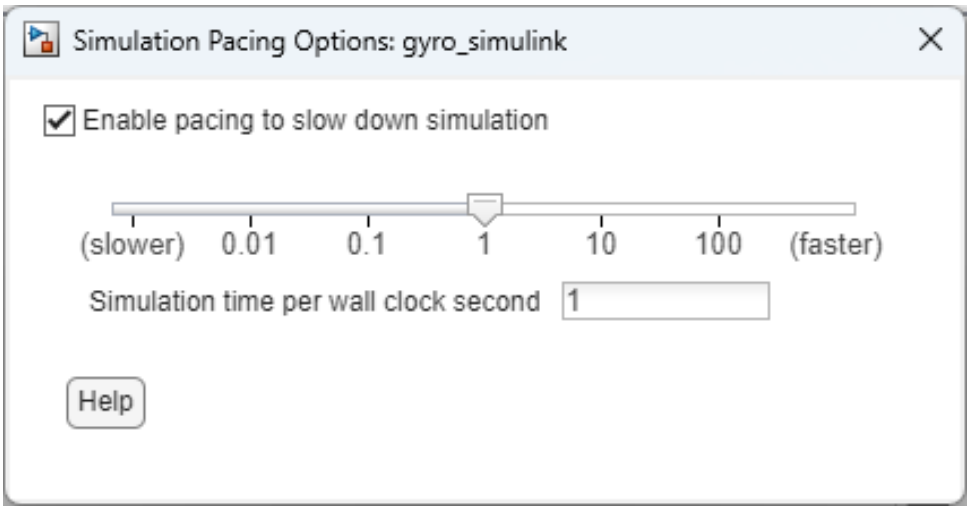
Příklad použití sériové komunikace pro nastavení konstant PID regulátoru v kostce a zobrazení aktuálních složek PID je na obrázku 9. V tomto příkladu se hodnoty odesílají a přijímají v binárním formátu, je nastavena hlavička pro odesílání a přijímaní a data jsou typu single (4 byty). Data se přijímají 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`.



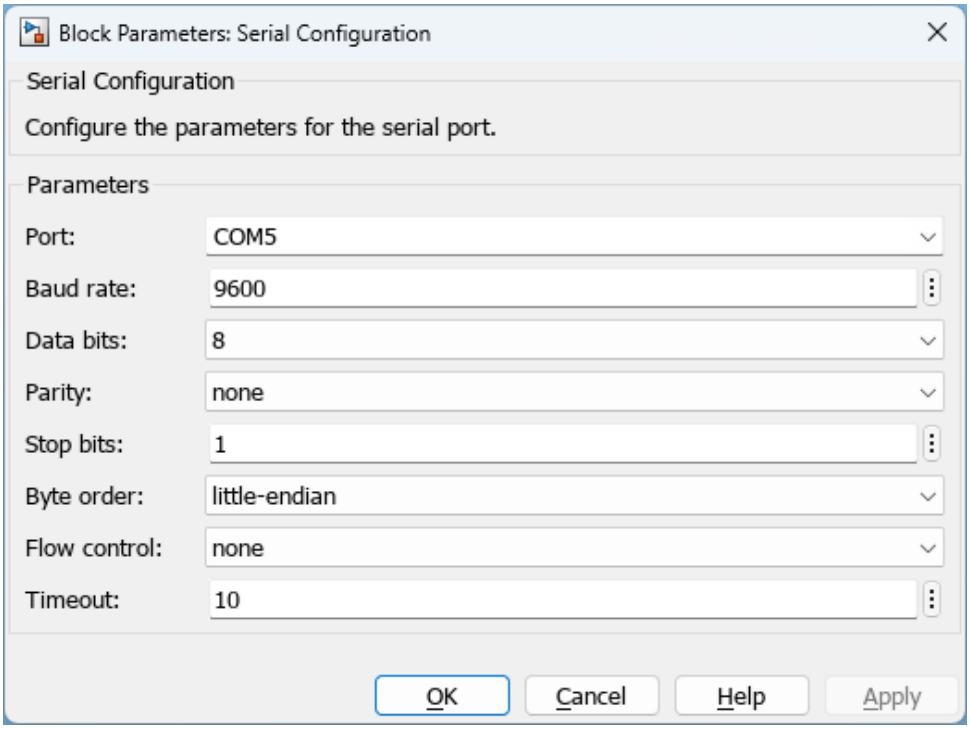
Obrázek 5: Nastavení řešiče Simulinku.



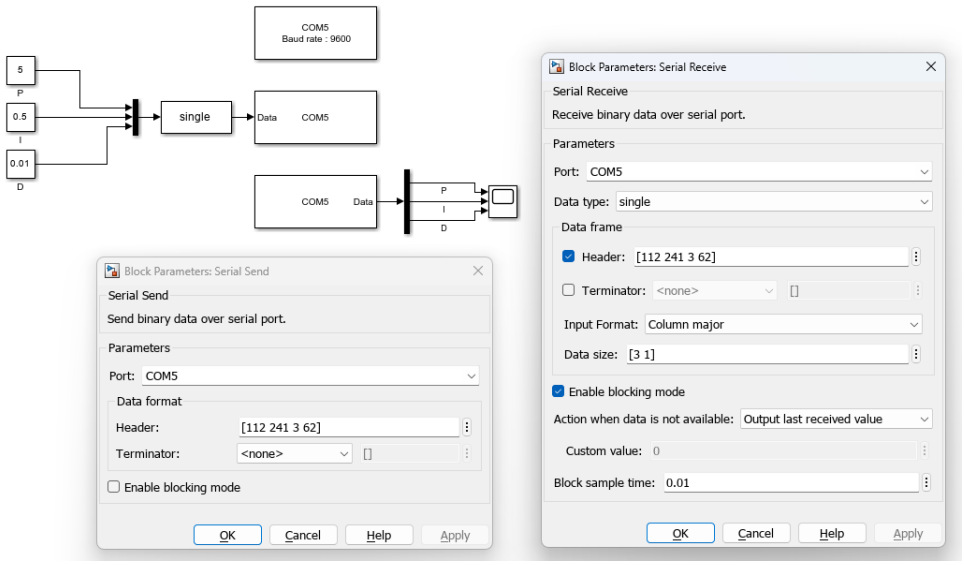
Obrázek 6: Umístění nastavení Simulation pacing v Simulinku.



Obrázek 7: Nastavení Simulation Pacing.



Obrázek 8: Nastavení bloku Serial Configuration v Simulinku.



Obrázek 9: Příklad použití sériové komunikace v Simulinku.

4 NXT senzory

4.1 Tlačítko

class **TouchSensor**(*port*)

NXT Touch Sensor.

Informace o stavu stisknutí tlačítka.

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

pressed()

Navrátí stav tlačítka.

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

```
# Importování 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.NXT_TOUCH, port=Port.S1)

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

Kód 11: Použití NXT tlačítka.

4.2 Světelný senzor

class **LightSensor**(*port*)

NXT Ligth Sensor.

Měření intenzity dopadajícího světla.

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.NXT_LIGHT,port=Port)`
Deinicializace: `robot.deinit_sensor(sensor_type=Sensor.NXT_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.

```
# Importování 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 12: Použití NXT světelného senzoru.

4.3 Ultrazvukový dálkoměr

class **UltrasonicSensor**()

NXT Ultrasonic Sensor.

Měření vzdálenosti objektu od senzoru v rozmezí 0–255 cm s přesností ±3 cm. Senzor musí být připojen do portu označeného I2C, nelze připojit více senzorů typu NXT Ultrasonic Sensor.

Inicializace: `robot.init_sensor(sensor_type=Sensor.NXT_ULTRASONIC)`
Deinicializace: `robot.deinit_sensor(sensor_type=Sensor.NXT_ULTRASONIC)`
Přístup: `robot.sensors.ultra_nxt`

distance()

Změří vzdálenost a navrátí ji.

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

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

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

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

Kód 13: Použití NXT ultrazvukového senzoru.

4.4 Zvukový senzor

class **SoundSensor**(*port*)

NXT Sound Sensor.

Měření intenzity okolního zvuku.

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

```
# Importování 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.NXT_SOUND, port=Port.S1)

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

Kód 14: Použití NXT zvukového senzoru.

5 EV3 senzory

5.1 Tlačítko

class **TouchSensor**(*port*)

EV3 Touch Sensor.

Informace o stavu stisknutí tlačítka.

Parametry: **port** (Port) – Port, ke kterému je senzor připojen.
Inicializace: `robot.init_sensor(sensor_type=Sensor.EV3_TOUCH,port=Port)`.
Deinicializace: `robot.deinit_sensor(sensor_type=Sensor.EV3_TOUCH,port=Port)`.
Přístup: `robot.sensors.touch[Port]`.

pressed()

Navrátí stav tlačítka.

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

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

# Inicializace EV3 tlačítka na senzorovém portu 1
robot.init_sensor(sensor_type=Sensor.EV3_TOUCH, port=Port.S1)

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

Kód 15: Použití EV3 tlačítka.

5.2 Světelný senzor

class **ColorSensor**(*port*)

EV3 Color Sensor.

Měření světelné intenzity.

Parametry: **port** (Port) – Port, ke kterému je senzor připojen.
Inicializace: `robot.init_sensor(sensor_type=Sensor.EV3_COLOR)`
Deinicializace: `robot.deinit_sensor(sensor_type=Sensor.EV3_COLOR)`
Přístup: `robot.sensors.light[Port]`

reflection()

Navrátí poslední změřenou odraženou intenzitu světla. Červená LED se střídavě zapíná a vypíná. Je změřena hodnota pro zapnutou LED, od které je odečtena hodnota pro vypnutou LED. Výsledná hodnota je upravena za použití kalibrace v senzoru a přemapována na interval 0-100 %.

Vrací: Odražená intenzita světla.
Typ: int (0-100 %)

reflection_raw()

Navrátí poslední změřenou odraženou intenzitu světla v neupraveném formátu. Červená LED se střídavě zapíná a vypíná. Je změřena hodnota pro zapnutou LED, od které je odečtena hodnota pro vypnutou LED.

Vrací: Odražená intenzita světla.
Typ: int

ambient()

Navrátí poslední změřenou intenzitu okolního světla. Výsledná hodnota je upravena za použití kalibrace v senzoru a přemapována na interval 0-100 %.

Vrací: Intenzita okolního světla.
Typ: int (0-100 %)

rgb_raw()

Navrátí poslední změřenou odraženou intenzitu světla pro zapnutou červenou, modrou a zelenou LED.

Vrací: Tuple odražené intenzity světla (červená, zelená, modrá).
Typ: (int, int, int)

rgb()

Navrátí poslední změřenou odraženou intenzitu světla pro zapnutou červenou, modrou a zelenou LED. Výsledná hodnoty jsou upraveny za použití OpenCube kalibrace a přemapovány na interval 0-100 %.

Vrací: Tuple odražené intenzity světla (červená, zelená, modrá).
Typ: (int, int, int) (0-100 %)

hsv()

Navrátí hodnoty HSV pro poslední změřenou odraženou intenzitu světla při zapnuté červené, modré a zelené LED.

Vrací: Tuple HSV.
Typ: (hue, saturation, value)

color()

Navrátí konstantu pro senzorem změřenou barvu. Lze použít konstanty barev Color.

Vrací: Konstanta změřené barvy.
Typ: int (1-7 – black, blue, green, yellow, red, white, brown)

```
# Importování konstant typů senzorů
from lib.robot_consts import Sensor, Port

# Inicializace EV3 světelného senzoru na senzorovém portu 1
robot.init_sensor(sensor_type=Sensor.EV3_COLOR, port=Port.S1)

# Zjistění intenzity odraženého světla
```

```
reflection = robot.sensors.light[Port.S1].reflection()
```

Kód 16: Použití EV3 světelného senzoru.

5.3 Ultrazvukový dálkoměr

class **UltrasonicSensor**(*port*)

EV3 Ultrasonic Sensor.

Měření vzdálenosti objektu od senzoru v rozmezí 0–2550 mm s přesností ±30 mm.

Parametry:	**port** (Port) – Port, ke kterému je senzor připojen.
Inicializace:	`robot.init_sensor(sensor_type=Sensor.EV3_ULTRASONIC)`
Deinicializace:	`robot.deinit_sensor(sensor_type=Sensor.EV3_ULTRASONIC)`
Přístup:	`robot.sensors.ultrasonic[Port]`

distance()

Navrátí poslední změřenou vzdálenost.

Vrací:	Vzdálenost objektu od senzoru.
Typ:	int (0-2550)

presence()

Zjistí přítomnost jiného ultrazvukového senzoru.

Vrací:	True pokud je přítomen jiný ultrazvukový senzor, jinak False.
Typ:	bool

```
# Importování konstant typů senzorů
from lib.robot_consts import Sensor, Port

# Inicializace EV3 ultrazvukového senzoru na senzorovém portu 1
robot.init_sensor(sensor_type=Sensor.EV3_ULTRASONIC, port=Port.S1)

# Změření vzdálenosti
distance = robot.sensors.ultrasonic[Port.S1].distance()
```

Kód 17: Použití EV3 ultrazvukového dálkoměru.

5.4 Infračervený senzor

class **InfraredSensor**(*port*)

EV3 Infrared Sensor.

Měření vzdálenosti k nejbližšímu povrchu a ovládání dálkovým ovladačem.

Parametry:	**port** (Port) – Port, ke kterému je senzor připojen.
Inicializace:	`robot.init_sensor(sensor_type=Sensor.EV3_INFRARED)`
Deinicializace:	`robot.deinit_sensor(sensor_type=Sensor.EV3_INFRARED)`
Přístup:	`robot.sensors.infrared[Port]`

distance()

Změří relativní vzdálenost k nejbližšímu povrchu. 100 % je přibližně 70 cm.

Vrací: Relativní vzdálenost povrchu od senzoru.
Typ: int (0-100 %)

seeker(*channel*)

Zjistí přítomnost dálkového ovladače na daném kanálu.

Parametry: **channel** (int) – Analyzovaný kanál.
Vrací: Tuple směru (-25 je nejvíce vlevo, 25 nejvíce vpravo) a vzdálenosti (100 % je přibližně 200 cm, -128, pokud není ovladač nalezen) ovladače.
Typ: (int, int) (-25-25, 0-100 %)

```
# Importování konstant typů senzorů
from lib.robot_consts import Sensor, Port

# Inicializace EV3 ultrazvukového senzoru na senzorovém portu 1
robot.init_sensor(sensor_type=Sensor.EV3_INFRARED, port=Port.S1)

# Změření vzdálenosti
distance = robot.sensors.infrared[Port.S1].distance()
```

Kód 18: Použití EV3 infračerveného senzoru.

5.5 Gyroskop

class GyroSensor(*port*)

EV3 Gyro Sensor.

Měření úhlu a úhlové rychlosti v jedné ose.

Parametry: **port** (Port) – Port, ke kterému je senzor připojen.
Inicializace: `robot.init_sensor(sensor_type=Sensor.EV3_GYRO)`
Deinicializace: `robot.deinit_sensor(sensor_type=Sensor.EV3_GYRO)`
Přístup: `robot.sensors.gyro[Port]`

angle()

Navrátí úhel natočení senzoru.

Vrací: Úhel natočení.
Typ: int (°)

speed()

Navrátí úhlovou rychlost otáčení senzoru.

Vrací: Úhlová rychlost otáčení.
Typ: int (°/s)

reset_angle(*angle*)

Umožňuje vynulovat měření úhlu nebo nastavit novou počáteční hodnotu úhlu.

Parametry: **angle** (int) – Nová hodnota, kterou zobrazí měření, pokud se senzor nepohne.

angle_and_speed()

Navrátí úhel natočení a úhlovou rychlost otáčení senzoru.

Vrací: Tuple úhlu natočení a úhlové rychlosti otáčení.

Typ: (int, int) (°, °/s)

coarse_speed()

Navrátí úhlovou rychlost otáčení senzoru s nižším rozlišením a větším rozsahem.

Vrací: Úhlová rychlost otáčení.

Typ: int (°/s)

```
# Importování konstant typů senzorů
from lib.robot_consts import Sensor, Port

# Inicializace EV3 gyroskopu na senzorovém portu 1
robot.init_sensor(sensor_type=Sensor.EV3_GYRO, port=Port.S1)

# Vynulování úhlu natočení
robot.sensors.gyro[Port.S1].reset_angle(0)

# Změření úhlu natočení a úhlové rychlosti
(angle, speed) = robot.sensors.gyro[Port.S1].angle_and_speed()
```

Kód 19: Použití EV3 gyroskopu.

6 Open-Cube senzory

6.1 Světelný senzor

class **ColorSensor**(*port*)

Open-Cube RGB optical reflective and color sensor.

Měření světelné intenzity.

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

reflection()

Navrátí poslední změřenou odraženou intenzitu světla. Červená LED se střídavě zapíná a vypíná. Je změřena hodnota pro zapnutou LED, od které je odečtena hodnota pro vypnutou LED. Výsledná hodnota je upravena za použití kalibrace v senzoru a přemapována na interval 0-100 %.

Vrací: Odražená intenzita světla.
Typ: int (0-100 %)

reflection_raw()

Navrátí poslední změřenou odraženou intenzitu světla v neupraveném formátu. Červená LED se střídavě zapíná a vypíná. Je změřena hodnota odražené intenzity světla pro zapnutou LED a referenční hodnota intenzity světla pro vypnutou LED.

Vrací: Tuple odražené a referenční intenzity světla.
Typ: (int, int)

reflection_raw_green()

Navrátí poslední změřenou odraženou intenzitu světla v neupraveném formátu. Zelená LED se střídavě zapíná a vypíná. Je změřena hodnota odražené intenzity světla pro zapnutou LED a referenční hodnota intenzity světla pro vypnutou LED.

Vrací: Tuple odražené a referenční intenzity světla.
Typ: (int, int)

reflection_raw_blue()

Navrátí poslední změřenou odraženou intenzitu světla v neupraveném formátu. Modrá LED se střídavě zapíná a vypíná. Je změřena hodnota odražené intenzity světla pro zapnutou LED a referenční hodnota intenzity světla pro vypnutou LED.

Vrací: Tuple odražené a referenční intenzity světla.
Typ: (int, int)

ambient()

Navrátí poslední změřenou intenzitu okolního světla. Výsledná hodnota je upravena za použití kalibrace v senzoru a přemapována na interval 0-100 %.

Vrací: Intenzita okolního světla.
Typ: int (0-100 %)

rgb_raw()

Navrátí poslední změřenou odraženou intenzitu světla pro zapnutou červenou, modrou a zelenou LED referenční intenzity světla pro vypnuté LED.

Vrací: Tuple odražené intenzity světla (červená, zelená, modrá) a referenční intenzity světla.
Typ: (int, int, int, int)

rgb()

Navrátí poslední změřenou odraženou intenzitu světla pro zapnutou červenou, modrou a zelenou LED. Výsledná hodnoty jsou upraveny za použití Open-Cube kalibrace a přemapovány na interval 0-100 %.

Vrací: Tuple odražené intenzity světla (červená, zelená, modrá).
Typ: (int, int, int) (0-100 %)

hsv()

Navrátí hodnoty HSV pro poslední změřenou odraženou intenzitu světla při zapnuté červené, modré a zelené LED.

Vrací: Tuple HSV.
Typ: (hue, saturation, value)

color()

Navrátí konstantu pro senzorem změřenou barvu. Lze použít konstanty barev Color.

Vrací: Konstanta změřené barvy.
Typ: int (1-7 – black, blue, green, yellow, red, white, brown)

```
# Importování konstant typů senzorů
from lib.robot_consts import Sensor, Port

# Inicializace Open-Cube světelného senzoru na senzorovém portu 1
robot.init_sensor(sensor_type=Sensor.OC_COLOR, port=Port.S1)

# Zjistění intenzity odraženého světla
reflection = robot.sensors.light[Port.S1].reflection()
```

Kód 20: Použití Open-Cube světelného senzoru.

6.2 Laserový dálkoměr

class **LaserSensor**(*port*)

Open-Cube LIDAR distance sensor.

Měření vzdálenosti objektu od senzoru v rozmezí 0–4000 mm. Minimální zaručená vzdálenost měření je 4 cm. Pokud se objekt nachází blíže, než je tato vzdálenost, senzor změří nepřesnou hodnotu.

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

distance()

Navrátí poslední změřenou vzdálenost.

Vrací: Vzdálenost objektu od senzoru.
Typ: int (0-4000)

distance_fov()

Navrátí poslední změřenou vzdálenost. Tento režim používá větší zorné pole senzoru.

Vrací: Vzdálenost objektu od senzoru.
Typ: int (0-4000)

distance_short()

Navrátí poslední změřenou vzdálenost. Tento režim je přesnější pro kratší vzdálenosti (do 1300 mm) a silné okolní světlo.

Vrací: Vzdálenost objektu od senzoru.
Typ: int (0-4000)

set_led_distance(*distance*)

Nastaví mezní vzdálenost, při které se rozsvítí zelená LED senzoru.

Parametry: **distance** (int) – Mezní vzdálenost v mm.

```
# Importování konstant typů senzorů
from lib.robot_consts import Sensor, Port

# Inicializace EV3 ultrazvukového senzoru na senzorovém portu 1
robot.init_sensor(sensor_type=Sensor.OC_LASER, port=Port.S1)

# Změření vzdálenosti
distance = robot.sensors.laser[Port.S1].distance()
```

Kód 21: Použití Open-Cube laserového dálkoměru.

6.3 Ultrazvukový dálkoměr

class **UltrasonicSensor**(*port*)

Open-Cube Ultrasonic Sensor.

Měření vzdálenosti objektu od senzoru v rozmezí 0–9999 mm.

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

distance()

Navrátí poslední změřenou vzdálenost.

Vrací: Vzdálenost objektu od senzoru.
Typ: int (0-9999)

set_led_distance(*distance*)

Nastaví mezní vzdálenost, při které se rozsvítí zelená LED senzoru.

Parametry: **distance** (int) – Mezní vzdálenost v mm.

```
# Importování konstant typů senzorů
from lib.robot_consts import Sensor, Port

# Inicializace Open-Cube ultrazvukový senzor na senzorovém portu 1
robot.init_sensor(sensor_type=Sensor.OC_ULTRASONIC, port=Port.S1)

# Změření vzdálenosti
distance = robot.sensors.ultrasonic[Port.S1].distance()
```

Kód 22: Použití Open-Cube ultrazvukového dálkoměru.

6.4 AHRS senzor

class **GyroSensor**(*port*)

Open-Cube AHRS (Attitude and Heading Reference System) Sensor.

Měření zrychlení, úhlové rychlosti a intenzity magnetického pole ve třech osách.

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

euler_angles()

Navrátí aktuální orientaci senzoru v Eulerových úhlech vypočtenou z gyroskopu a akcelerometru.

Vrací: Tuple Eulerových úhlů yaw, pitch, roll.
Typ: (yaw: -180–180°, pitch: -90–90°, roll: -180–180°)

euler_angles_mag()

Navrátí aktuální orientaci senzoru v Eulerových úhlech vypočtenou z gyroskopu, akcelerometru a magnetometru.

Vrací: Tuple Eulerových úhlů yaw, pitch, roll.
Typ: (yaw: -180–180°, pitch: -90–90°, roll: -180–180°)

quaternions()

Navrátí aktuální orientaci senzoru v kvaternionech vypočtenou z gyroskopu, akcelerometru.

Vrací: Tuple kvaternionů.
Typ: (0–1, 0–1, 0–1, 0–1)

raw()

Navrátí poslední změřené úhlové rychlosti, zrychlení a intenzity magnetického pole v osách x, y, z.

Vrací: Tuple úhlových rychlostí, zrychlení a intenzit magnetického pole v osách x, y, z.
Typ: (g_x: °/s, g_y: °/s, g_z: °/s, a_x: -, a_y: -, a_z: -, m_x: mg, m_y: mg, m_z: mg)

calibrate_gyro()

Spustí kalibraci offsetu gyroskopu. Senzor se nesmí pohybovat po celou dobu kalibrace.

Vrací: 1 pokud je kalibrace dokončena.
Typ: int (0/1)

calibrate_mag()

Spustí hard iron kalibraci magnetometru. Senzorem otáčejte dokud není kalibrace dokončena.

Vrací: 1 pokud je kalibrace dokončena.
Typ: int (0/1)

```
# Importování konstant typů senzorů
from lib.robot_consts import Sensor, Port

# Inicializace OC AHRS senzoru na senzorovém portu 1
robot.init_sensor(sensor_type=Sensor.OC_GYRO, port=Port.S1)

# Kalibrace gyroskopu
calibrated = 0
while not calibrated:
  calibrated = robot.sensors.gyro[Port.S1].calibrate_gyro()

# Zjištění orientace senzoru
euler_angles = robot.sensors.gyro[Port.S1].euler_angles()
yaw, pitch, roll = euler_angles[0], euler_angles[1], euler_angles[2]
```

Kód 23: Použití Open-Cube AHRS senzoru.

7 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 950 °/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 %. Při zadání hodnoty mimo toto rozmezí je nastavena nejbližší krajní hodnota.

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()

Deinicializuje 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. Výchozí hodnoty jsou 0.9, 0.1 a 0.

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. Výchozí hodnoty jsou 0.1, 0.8 a 0.

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 %)

```
# Importování 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 na 50 % a 20 % v opačném směru otáčení
robot.motors[Port.M1].set_power(50)
robot.motors[Port.M2].set_power(-20)

# Inicializace enkodérů pro měření pozice a rychlosti motorů
robot.motors[Port.M1].init_encoder()
robot.motors[Port.M2].init_encoder()

# 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 24: Použití motorů.

8 Parametry a konstanty

Parametry a konstanty importovatelné pomocí `lib.robot_consts`.

8.1 Sensor

class **Sensor**

Typy senzorů.

NO_SENSOR

Žádný senzor.

NXT_LIGHT

NXT světelný senzor.

NXT_ULTRA

NXT ultrazvukový dálkoměr.

NXT_TOUCH

NXT tlačítko.

NXT_SOUND

NXT zvukový senzor.

GYRO_ACC

ICM20608-G gyroskop a akcelerometr.

EV3_COLOR

EV3 světelný senzor.

EV3_GYRO

EV3 gyroskop.

EV3_INFRARED

EV3 infračervený senzor.

EV3_TOUCH

EV3 tlačítko.

EV3_ULTRASONIC

EV3 ultrazvukový dálkoměr.

OC_LASER

Open-Cube laserový dálkoměr.

OC_COLOR

Open-Cube světelný senzor.

OC_ULTRASONIC

Open-Cube ultrazvukový dálkoměr.

OC_GYRO

Open-Cube AHRS senzor.

8.2 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.

8.3 Button

class **Button**

Tlačítka na kostce.

POWER

Tlačítko zapnout.

LEFT

Tlačítko doleva.

RIGHT

Tlačítko doprava.

OK

Tlačítko OK.

UP

Tlačítko nahoru.

DOWN

Tlačítko dolů.

8.4 Light

class **Light**

Změřená hodnota při vypnuté a zapnuté LED NXT světelného senzoru.

OFF

Změřená hodnota při vypnuté LED.

ON

Změřená hodnota při zapnuté LED.

8.5 GyroAcc

class **GyroAcc**

Změřená hodnota vestavěného gyroskopu a akcelerometru ICM42688 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.

8.6 Color

class **Color**

Barva detekovaná EV3 a Open-Cube světelnými senzory.

BLACK

BLUE

GREEN

YELLOW

RED

WHITE

BROWN

9 Užitečné funkce MicroPythonu

9.1 Výpis informací

```
x, y = 64, 128.4096
y_string = "y"

# 1. varianta - oddělení čárkami
# Tato varianta je vhodná na jednoduché a rychlé zobrazení textu
# Jednotlivými prvky jsou odděleny mezerou
print("x =", x, ",", y_string, "=", y)
# x = 64, y = 128.4096

# 2. varianta - použití f-stringu
# Tato varianta umožňuje formátovat float čísla
# d označuje typ integer, s typ string a f typ float
# .2 určuje počet zobrazených desetinných míst
print(f"x = {x:d}, {y_string:s} = {y:.2f}")
# x = 64, y = 128.41

# Použití f-stringu je vhodné i pro zobrazení na displeji kostky
robot.display.fill(0)
# první řádek displeje
robot.display.text(f"x = {x:d}", 0, 0, 1)
# druhý řádek displeje
robot.display.text(f"{y_string:s} = {y:.2f}", 0, 10, 1)
robot.display.show()

# Výpis bez odřádkování
for i in range(10)
  print(i, end="")
print() # odřádkování na konci
# 123456789
```

Kód 25: Výpis informací.

9.2 Generování náhodných čísel

Ukázka generování náhodných čísel.

- MicroPython dokumentace knihovny random [https://docs.micropython.org/en/latest/library/random.html]

```
# Importování funkcí knihovny random
from random import random, randint

random() # Náhodné číslo typu float v intervalu [0.0, 1.0)

a, b = 0, 100
randint(a, b) # Náhodné číslo typu int (celé číslo) v intervalu [a, b]
```

Kód 26: Použití knihovny random.

9.3 Čas

Ukázka funkcí pro uspání programu, měření času a periodického volání funkce pomocí timeru.

- MicroPython dokumentace knihovny time [https://docs.micropython.org/en/latest/library/time.html]
- MicroPython dokumentace třídy Timer [https://docs.micropython.org/en/latest/library/machine.Timer.html]

```
# Importování funkcí knihovny sleep
from time import sleep, sleep_ms, sleep_us, ticks_us, ticks_diff

start_time = ticks_us() # Zjištění počátečního času v mikrosekundách

sleep(1)    # Uspání programu na 1 sekundu
sleep_ms(2) # Uspání programu na 2 milisekundy
sleep_us(3) # Uspání programu na 3 mikrosekundy

end_time = ticks_us() # Zjištění koncového času v mikrosekundách
print(ticks_diff(end_time, start_time), "us") # 1002003 us
```

Kód 27: Použití knihovny time.

```
# Importování třídy Timer
from machine import Timer

counter = 0

# Definice callback funkce pro timery
def my_callback(t):
  global counter
  counter += 1
  print(counter)

# Inicializace timeru, který volá callback funkci periodicky s frekvencí 100 Hz
# -1 značí virtuální timer (mikrokontrolér rp2040 neumožňuje použití hardwarových timerů)
timer1 = Timer(-1)
timer1.init(mode=Timer.PERIODIC, freq=100,  callback=my_callback)

# Inicializace timeru, který zavolá callback funkci pouze jednou po uplynutí 1000 ms
timer2 = Timer(-1)
timer2.init(mode=Timer.ONE_SHOT, period=1000, callback=my_callback)

# Vypnutí periodického timeru
timer1.deinit()
```

Kód 28: Použití třídy Timer.

9.4 Manipulace s binárními daty

Tyto funkce jsou užitečné při práci s binárními daty např. při bezdrátové komunikaci přes ESP32.

- Micropython dokumentace knihovny struct [https://docs.micropython.org/en/latest/library/struct.html]

```
# Importování knihovny struct
import struct

# Vytvoření binárních dat ze tří čísel. Formát čísel je určen stringem prvního argumentu. Formát stringu je podrobně popsán v dokumentaci knihovny struct.
# V tomto případě je první číslo uloženo jako typ int (4 byty), druhé jako float (4 byty) a třetí jako double (8 bytů)
binary_data = struct.pack("@ifd", 12, 1.234, 5)

# Konverze binárních dat
values = struct.unpack("@ifd", binary_data)

# Spočítá velikost paměti v bytech potřebnou k uložení binárních dat
byte_size = struct.calcsize("@ifd")
print(byte_size) # 16
```

Kód 29: Použití knihovny struct.

9.5 Vícejádrové programování

Mikrokontrolér rp2040 má 2 jádra, defaultně se používá pouze jedno jádro. Použití druhého jádra je možné, ale aktuálně je v MicroPythonu pouze experimentální.

- Python dokumentace knihovny _thread [https://docs.python.org/3.5/library/_thread.html]

```
# Importování knihovny _thread a funkce sleep
import _thread
from time import sleep

lock = _thread.allocate_lock()
counter = 0

# Definice funkce druhého jádra
def core2():
  global counter
  lock.acquire() # Uzamčení pro práci se společnými daty
  counter += 1
  lock.release() # Uvolnění
  _thread.exit() # Ukončení běhu jádra

# Aktivace druhého jádra
_thread.start_new_thread(core2, ())

lock.acquire() # Uzamčení pro práci se společnými daty
print(counter)
lock.release() # Uvolnění
```

Kód 30: Použití knihovny _thread.

10 Použité pojmy Pythonu

Datové typy:

- **int** (integer) – celé číslo (12, -51),
- **float** – desetinné číslo (120.32, -0.1248),
- **bool** – pravdivostní hodnota (True, False),
- **str** (string) – textový řetězec ("abc", "Open-Cube").

Datové struktury:

- **list** – posloupnost dat, která může obsahovat prvky různých datových typů. Jednotlivé prvky lze měnit, přidávat a odebírat. Zapisuje se do hranatých závorek a prvky jsou odděleny čárkou. [1, "Open-Cube", True, 0.23]
- **tuple** – posloupnost dat, která může obsahovat prvky různých datových typů. Jednotlivé prvky nelze měnit ani přidávat a odebírat. Zapisuje se do kulatých závorek a prvky jsou odděleny čárkou. (1, "Open-Cube", True, 0.23)

Klíčová slova:

- **None** – žádná hodnota (senzor nefunguje, nedokázal změřit hodnotu atd.).