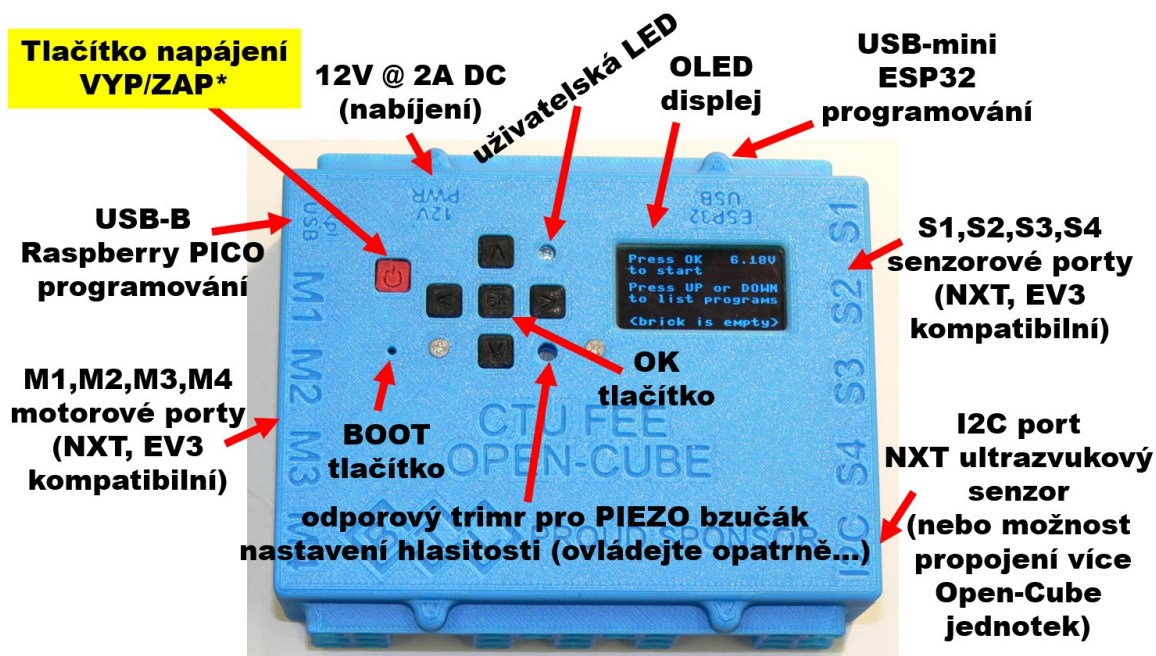


OPEN-CUBE STRUČNÝ NÁVOD

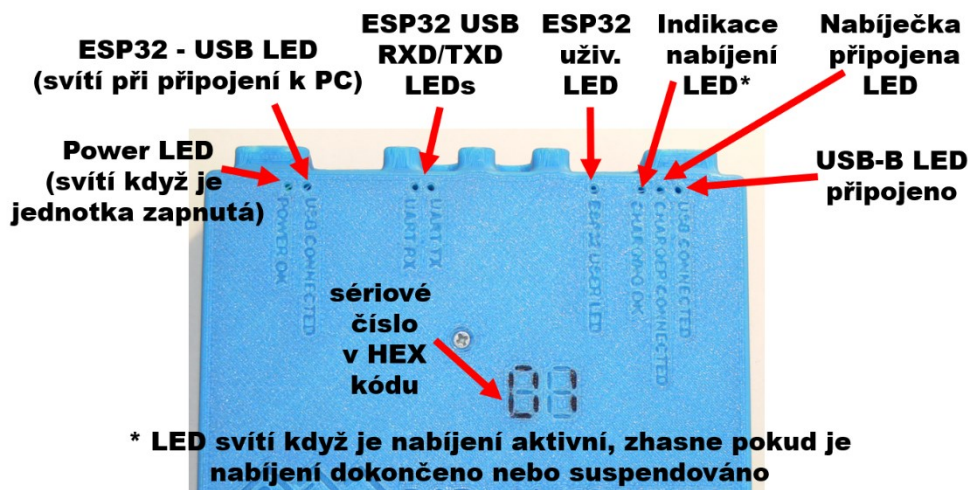
Pro více informací navštivte stránku: open-cube.fel.cvut.cz

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...). Nainstalujte si Thonny plugin pomocí plug-in manageru (<https://gitlab.fel.cvut.cz/open-cube/firmware/-/releases>).
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.
3. Alternativně spárujte Open-Cube pomocí Bluetooth serial s počítačem, použijte plug-in do programu Thonny a programy nahrávejte bezdrátově, viz sekce 3.9 v podrobnějším návodu.



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



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 (stejně 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 (snímání 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() # změř vzdálenost
```

O-C ULTRAZVUKOVÝ DÁLKOMĚR více 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 více v kapitole 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(4000,0.5) # 4000 Hz, 50 % střída (střída 0.0-1.0)
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), viz 3.7 a 3.8

BLUETOOTH sériová komunikace & **Wi-Fi** přístupový bod – více v kapitole 3.9

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 *DataPlotteru*

Open-Cube

Katedra měření ČVUT FEL

2026

Obsah

1	Instalace firmwaru	4
2	Editor kódu Thonny	4
2.1	Nastavení editoru	4
2.2	Nahrání kódu do kostky	4
2.3	Bezdrátové nahrání a spuštění programu	5
2.4	Menu kostky	6
3	Robot	7
3.1	Tlačítka	9
3.2	LED	10
3.3	Bzučák	11
3.4	Baterie	12
3.5	Displej	13
3.6	Gyroskop a akcelerometr	17
3.7	I2C master	19
3.8	I2C slave	21
3.9	ESP32 komunikace	22
3.9.1	Bluetooth propojení ESP32 s jiným zařízením	23
3.9.2	Kostka	25
3.9.3	Dálkové ovládání kostky	28
3.9.4	Serial Bluetooth Terminal	29
3.9.5	Matlab	29
3.9.6	Simulink	30
4	NXT senzory	34
4.1	Tlačítko	34
4.2	Světelný senzor	34
4.3	Ultrazvukový dálkoměr	36
4.4	Zvukový senzor	36

5	EV3 senzory	38
5.1	Tlačítko	38
5.2	Světelný senzor	38
5.3	Ultrazvukový dálkoměr	40
5.4	Infračervený senzor	40
5.5	Gyroskop	41
6	Open-Cube senzory	43
6.1	Tlačítko	43
6.2	Světelný senzor	43
6.3	Laserový dálkoměr	45
6.4	Ultrazvukový dálkoměr	46
6.5	AHRS senzor	47
7	Servomotory	49
8	Parametry a konstanty	52
8.1	Sensor	52
8.2	Port	53
8.3	Button	54
8.4	Light	54
8.5	GyroAcc	54
8.6	Color	55
8.7	Note	56
8.8	Melody	56
9	Užitečné funkce MicroPythonu	57
9.1	Výpis informací	57
9.2	Generování náhodných čísel	57
9.3	Čas	58
9.4	Manipulace s binárními daty	58
9.5	Vícejádrové programování	59
10	Použité pojmy Pythonu	60

Seznam kódů

1	Použití globální třídy robot.	9
2	Zjištění stavu tlačítek.	10
3	Použití LED.	10
4	Použití bzučáku.	12
5	Zjištění napětí baterií.	13
6	Použití displeje.	17
7	Získávání raw dat a orientace kostky z gyroskopu a akcelerometru.	18
8	Použití kostky pro mezikostkovou komunikaci jako I2C master.	20
9	Použití kostky pro mezikostkovou komunikaci jako I2C slave.	22
10	Komunikace mezi mikroprocesorem a ESP32.	27
11	Komunikace s kostkou z Matlabu.	29
12	Použití NXT tlačítka.	34
13	Použití NXT světelného senzoru.	36
14	Použití NXT ultrazvukového senzoru.	36
15	Použití NXT zvukového senzoru.	37
16	Použití EV3 tlačítka.	38
17	Použití EV3 světelného senzoru.	40
18	Použití EV3 ultrazvukového dálkoměru.	40
19	Použití EV3 infračerveného senzoru.	41
20	Použití EV3 gyroskopu.	42
21	Použití Open-Cube světelného senzoru.	45
22	Použití Open-Cube laserového dálkoměru.	46
23	Použití Open-Cube ultrazvukového dálkoměru.	46
24	Použití Open-Cube AHRS senzoru.	48
25	Použití motorů.	51
26	Výpis informací.	57
27	Použití knihovny random.	57
28	Použití knihovny time.	58
29	Použití třídy Timer.	58
30	Použití knihovny struct.	59
31	Použití knihovny _thread.	59

1 Instalace firmwaru

MicroPython firmware spolu se všemi Open-Cube knihovnami popsány v tomto dokumentu je ve výchozím nastavení nahráný v kostce. Aktuální firmware lze stáhnout z [stránky Open-Cube releases](https://gitlab.fel.cvut.cz/open-cube/firmware/-/releases) [<https://gitlab.fel.cvut.cz/open-cube/firmware/-/releases>]. Pro běžné použití stáhněte `opencube_micropython_with_pylibs.uf2`, který obsahuje nativní moduly a vestavěné Python knihovny Open-Cube.

Pro nahrání firmwaru při jeho aktualizaci, poškození či přepsání postupujte následujícími kroky:

1. Stáhněte si firmware v UF2 souboru .
2. Připojte kostku k počítači pomocí USB kabelu.
3. V menu kostky otevřete stránku About a podržte tlačítko OK přibližně jednu sekundu. Alternativně podržte tlačítko boot select a stiskněte zapínací tlačítko.
4. V počítači otevřete adresář RPI-RP2 a zkopírujte do něj stažený firmware.
5. Po nahrání firmwaru se kostka restartuje a na displeji se zobrazí menu.

2 Editor kódu Thonny

Pro úpravu, debugování a nahrávání kódu do kostky doporučujeme použít editor [Thonny](https://thonny.org/) [<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 interpret 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 stiskněte **Run current script** pro spuštění zobrazeného skriptu na kostce. Startovací kód firmwaru před spuštěním skriptu vytvoří globální objekt **robot**, proto lze script spouštět jak z menu, tak i přímo z editoru.

2.3 Bezdrátové nahrání a spuštění programu

Plugin Open-Cube pro Thonny umožňuje nahrávat a spouštět programy přes sériové spojení Bluetooth:

1. Nainstalujte plugin z PyPI příkazem `pip install thonny-open-cube`, nebo nainstalujte soubor `.whl` ze [stránky Open-Cube releases](https://gitlab.fel.cvut.cz/open-cube/firmware/-/releases) [<https://gitlab.fel.cvut.cz/open-cube/firmware/-/releases>] pomocí **Tools -> Manage plug-ins... -> Install from local file**.
2. Spárujte kostku s počítačem (viz [3.9.1](#)).
3. V nastavení **Tools -> Options -> Open-Cube** zvolte Bluetooth COM port kostky. Výchozí hodnotou je `COM3` ve Windows a `/dev/rfcomm0` v Linuxu.
4. Otevřete Python soubor a stiskněte **OC Run** nebo **Shift+F5**. Upravený soubor se před nahráním automaticky uloží.
5. Stiskněte **OC Stop** nebo **Shift+F6** pro přerušení nahrávání. Je-li zapnuté **Take control**, stejný příkaz zastaví běžící program.

Volby pluginu jsou: spuštění po nahrání, převzetí řízení běžícího programu, automatické připojení, výpis debugovacích zpráv, doplňování kódu a Bluetooth port. Spuštění po nahrání, převzetí řízení, automatické připojení a doplňování kódu jsou ve výchozím nastavení zapnuté.

Open-Cube log zobrazuje stav připojení, jméno kostky, napětí a procento baterie a informace o firmwaru. Je-li zapnuté **Take control**, stav baterie se obnovuje přibližně jednou za sekundu a během běhu programu se zobrazují zprávy odeslané metodou `robot.esp.bt_write()`. Je-li zapnuté automatické připojení, plugin se po ztrátě spojení znovu připojí.

Nástroj **Tools -> Open-Cube File Manager** vypíše programy uložené v kostce a po potvrzení může smazat vybraný soubor nebo adresář.

OC Stop nejprve požádá o šetrné přerušení programu. Pokud jej program nedokáže zpracovat přibližně do tří sekund, například protože je blokován v nativním kódu, plugin restartuje kostku. Bez **Take control** plugin běžící program nesleduje a **OC Stop** po dokončení nahrávání nemá účinek.

2.3.0.1 Přenos programu

Přenos programu používá okno 20 paketů, ochranu CRC-32 každého paketu a ověření SHA-256 celého souboru. Výchozí velikost dat je 512 bytů a přijímač přijímá data až do 1024 bytů. Soubory jsou omezeny na 100 KiB a přijmou se pouze při dostatečné volné flash paměti a RAM. Kostka přijímaná data zapisuje do dočasného souboru a až po úspěšném ověření jej přejmenuje na konečný cíl v `/programs`.

2.4 Menu kostky

Menu kostky se ovládá tlačítky LEFT, RIGHT, UP, DOWN a OK. Tlačítka LEFT a RIGHT se volí stránky Programs, Motor test, Sensor test, Cube test, BT pair, Settings a About. Tlačítka UP a DOWN se volí položka a tlačítko OK ji aktivuje.

Stránka Settings obsahuje následující možnosti:

- **Brightness** – jas displeje od 10 do 100% v krocích po 10%,
- **Invert** – normální nebo invertované barvy displeje,
- **BT name** – název Bluetooth a Wi-Fi `Open-Cube01` až `Open-Cube99`,
- **Bat calib** – násobitel měření baterie od 0,50 do 1,50,
- **OC doublecore** – použití druhého jádra pro komunikaci s periferií (uživatelský program používá první jádro), pokud je tato možnost vypnutá, druhé jádro se nepoužívá,
- **Reset** – obnovení výchozího nastavení.

Nastavení se ukládá do `/config.json` v adresáři zařízení.

Stránka About zobrazuje verzi firmwaru, commit a poslední čtyři číslice sériového čísla RP2040 mikrokontroléru. Podržetím OK přibližně jednu sekundu se kostka přepne do režimu USB bootloaderu.

3 Robot

Během startu firmwaru vytvoří `boot.py` globální objekt `robot`. Tento objekt je dostupný programům spuštěným z menu i skriptům spuštěným přes Thonny. Obsahuje funkce pro inicializaci, deinicializaci a přístup k perifériím kostky. Struktura jeho proměnných periférií je následující:

```
robot
├── battery
├── buttons
├── buzzer
├── display
├── esp
├── led
├── i2c_master
├── i2c_slave
├── 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 periférií kostky. Automaticky inicializuje tlačítka, LED, bzučák, měření napětí baterie, displej a komunikaci ESP32. Uživatelem lze inicializovat senzory, motory a volitelnou mezikostkovou komunikaci I2C.

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. Nepoužívá se pro [Sensor.GYRO_ACC](#) a [Sensor.NXT_ULTRASONIC](#); všechny ostatní typy senzorů vyžadují port.

```
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.

init_i2c_master()

Inicializace kostky jako I2C master pro mezikostkovou komunikaci.

deinit_i2c_master()

Deinicializace I2C master.

init_i2c_slave(*address*)

Inicializace kostky jako I2C slave pro mezikostkovou komunikaci.

Parametry: **address** (8-119) – Adresa I2C zařízení.

deinit_i2c_slave()

Deinicializace I2C slave.

deinit_all()

Vypne LED a bzučák a deinicializuje uživatelské I2C, motory a senzory.

reset()

Restartuje kostku.

bootloader()

Otevře USB bootloader RP2040.

turn_off()

Vypne kostku.

fw_version()

Vrátí datum sestavení firmwaru.

storage()

Vypíše celkovou a volnou kapacitu souborového systému.

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.

```
1 # Importování funkce pro uspání programu
2 from time import sleep
3 # Importování konstant tlačítek na kostce
4 from lib.robot_consts import Button
5
6 # Smyčka čekající na ukončení programu
7 while True:
8     # Změna stavu LED
9     robot.led.toggle()
10
11     # Získání stavu tlačítek
12     buttons = robot.buttons.pressed_since()
13     # Ukončení programu, pokud je levé tlačítko stisknuté
14     if buttons[Button.LEFT]:
15         break
16
17     # Uspání programu na 1 sekundu
18     sleep(1)
```

Kód 1: 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)

pressed_since()

Navrátí tlačítka stisknutá od předchozího volání.

Vrací: Tuple stavů tlačítek (power, left, right, ok, up, down).

Typ: (bool, bool, bool, bool, bool, bool)

```

1 # Importování funkce pro uspání programu
2 from time import sleep
3 # Importování konstant tlačítek na kostce
4 from lib.robot_consts import Button
5
6 while True:
7     # Získání stavu tlačítek
8     buttons = robot.buttons.pressed_since()
9
10    # Zobrazení stavu tlačítek v terminálu
11    print("Tlačítko stisknuté:",
12          "zapnout:", buttons[Button.POWER],
13          "doleva:", buttons[Button.LEFT],
14          "doprava:", buttons[Button.RIGHT],
15          "OK:", buttons[Button.OK],
16          "nahoru:", buttons[Button.UP],
17          "dolů:", buttons[Button.DOWN])
18    # Ukončení programu, pokud je levé tlačítko stisknuté
19    if buttons[Button.LEFT]:
20        break
21    # Uspání programu na 1 sekundu
22    sleep(1)

```

Kód 2: 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.

```

1 # Zapnutí LED
2 robot.led.on()
3 # Vypnutí LED
4 robot.led.off()

```

Kód 3: 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 střidu.

Parametry:

- **freq** (Hz) – Frekvence PWM.
- **duty** (float) – Střída PWM od 0,0 do 1,0.

```
off()
```

Ukončí aktuální notu, melodii nebo WAV soubor a vypne bzučák.

```
play_note(frequency, duration_ms, duty=0.5)
```

Spustí notu na zadanou dobu a ihned se vrátí.

Parametry:

- **frequency** (int: Hz) – Frekvence noty; nula vytvoří pauzu.
- **duration_ms** (int: ms) – Doba noty.
- **duty** (float) – Střída od 0,0 do 1,0.

```
play_sequence(notes)
```

Přehraje posloupnost dvojic (**frequency**, **duration_ms**) bez blokování programu.

```
play_melody(melody)
```

Přehraje konstantu [Melody](#), posloupnost not nebo název melodie bez rozlišení velikosti písmen.

```
play_wav(path, loop=False)
```

Spustí neblokující přehrávání WAV souboru. Podporovaný formát je mono PCM bez znaménka s 8 bity a vzorkovací frekvencí 4–32 kHz.

Parametry:

- **path** (str) – Cesta k WAV souboru.
- **loop** (bool) – Opakuje soubor až do volání `off()`.

```
is_playing_wav()
```

Vrací True, dokud probíhá přehrávání WAV.

```

1 # Zapnutí bzučáku na frekvenci 4000 Hz se střídou 50 %
2 robot.buzzer.set_freq_duty(4000, 0.5)
3 # Změna frekvence bzučáku na 1000 Hz
4 robot.buzzer.set_freq_duty(1000, 0.5)
5 # Vypnutí bzučáku
6 robot.buzzer.off()

```

Kód 4: Použití bzučáku.

Stránka Cube test obsahuje program Melodies pro předdefinované posloupnosti [Melody](#) a program Music (WAV) pro soubory uložené v `/music`. Program Music tento adresář v případě potřeby vytvoří, vypíše jeho soubory `.wav` a tlačítkem OK volí běžné přehrávání, opakování a zastavení.

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. Firmware kostku vypne, pokud naměřené napětí klesne pod 6,3 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

`set_multiplier(multiplier)`

Nastaví násobitel kalibrace baterie a ihned přepočítá uložené měření.

Parametry: **multiplier** (float) – Násobitel kalibrace baterie.

`deinit()`

Vypne periodické měření napětí.

```

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

```

Kód 5: 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(full_update=False)
```

Odešle změněné stránky frame bufferu na displej. Přenos přes I2C probíhá asynchronně a čekající změny se automaticky sloučí. Nastavením **full_update** na **True** odešlete všechny stránky.

```
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=0b1111*)

Nakreslí elipsu do frame bufferu.

- Parametry:**
- **x, y** – Souřadnice středu elipsy.
 - **xr, yr** – Vodorovný a svislý poloměr v pixelech.
 - **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_triangle(*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, c=1*)

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

- Parametry:**
- **s** (str) – Text.
 - **y** – Souřadnice horní hranice textu.
 - **c** (0/1) – Barva – 0 černá, 1 bílá.

```
draw_bar_chart_v(value, x, y, w, h, low_lim=0, high_lim=100,  
no_of_ticks=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_ticks** (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_ticks=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_ticks** (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, sa, 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.
- **sa** (int) – Rozsah číselníku ve stupních.
- **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.

contrast(*contrast*)

Nastaví kontrast displeje od 0 do 255.

invert(*invert*)

Zapne nebo vypne invertované barvy displeje.

flip(*flag=None, update=True*)

Zvolí normální orientaci displeje nebo otočení o 180 stupňů.

draw_loading_bar(*progress, y=0, bar_width=80, bar_height=10, label=None*)

Nakreslí do frame bufferu ukazatel průběhu.

draw_battery_icon(*battery_level, y=0, x=0*)

Nakreslí do frame bufferu ikonu baterie.

```

1 # Vymazání displeje
2 robot.display.fill(0)
3 # Zapsání textu do frame bufferu do levého horního rohu
4 robot.display.text("test message", 0, 0, 1)
5
6 # Zapsání čísel s přesností na 2 desetinná místa do frame bufferu na
   další řádek
7 x, y = 1.23, 56.789
8 robot.display.text(f"x={x:.2f}, y: {y:.2f}", 0, 10, 1)
9 # Zobrazení frame bufferu na displeji
10 robot.display.show()
11
12 # Funkce, které vrací hodnotu pro snížení problikávání grafu by měly
   být ve smyčce volány následovně
13 redraw = True
14 while(True):
15     redraw = robot.display.draw_dial(50,128/2,40,20,0,100,10,270,
        "Speedometer",redraw)

```

Kód 6: Použití displeje.

3.6 Gyroskop a akcelerometr

class ICM42688()

Integrovaný gyroskop a akcelerometr.

Měření zrychlení a úhlové rychlosti ve třech osách. Fúze dat ze senzorů pro výpočet Eulerových úhlů a quaternionů orientace kostky.

Inicializace: robot.init_sensor(sensor_type=Sensor.GYRO_ACC)

Deinicializace: robot.deinit_sensor(sensor_type=Sensor.GYRO_ACC)

Přístup: robot.sensors.gyro_acc

raw()

Navrátí poslední 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: g, a_y: g, a_z: g, g_x: °/s, g_y: °/s, g_z: °/s)

euler_angles()

Navrátí orientaci kostky v Eulerových úhlech. Algoritmus fúze musí být zapnutý.

Vrací: Tuple Eulerových úhlů yaw, pitch roll.

Typ: (yaw: °, pitch: °, roll: °)

quaternions()

Navrátí orientaci kostky v quaternionech. Algoritmus fúze musí být zapnutý.

Vrací: Tuple quaternionů.

Typ: (w, x, y, z)

`start_fusion()`

Zapne algoritmus pro fúzi dat z akcelerometru a gyroskopu pro výpočet Eulerových úhlů a quaternionů.

`stop_fusion()`

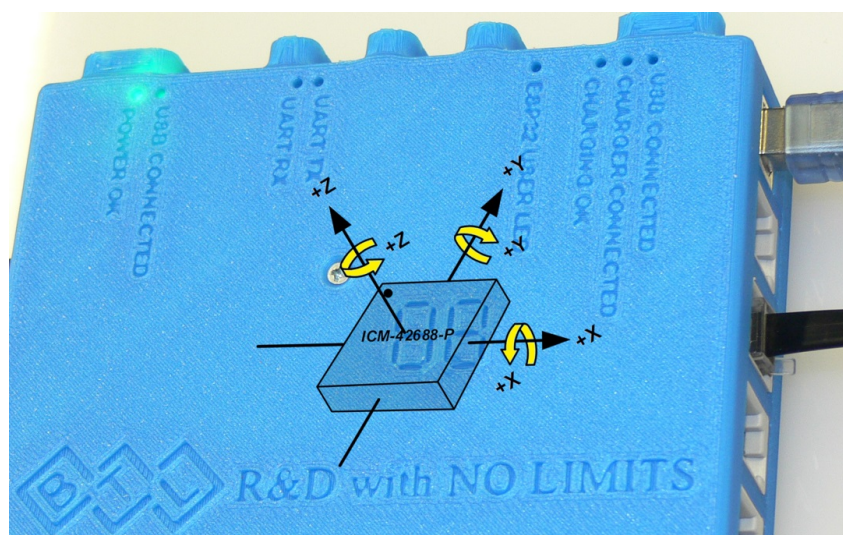
Vypne algoritmus pro fúzi dat z akcelerometru a gyroskopu.

`reset_fusion()`

Vynuluje yaw úhel a nastaví novou orientaci souřadnicovou soustavy podle směru gravitačního vektoru.

`calibrate_gyro()`

Spustí kalibraci offsetu gyroskopu. Kostku ponechte v klidu a metodu volejte, dokud nevrátí True.



Obrázek 1: Orientace os integrovaného gyroskopu a akcelerometru.

Příklad získávání raw dat a orientace kostky z gyroskopu a akcelerometru:

```
1 # Importování knihoven
2 from time import sleep
3 # Importování konstant
4 from lib.robot_consts import Button, Sensor, GyroAcc
5
6 # Inicializace senzoru a algoritmu pro fúzi dat
7 robot.init_sensor(sensor_type=Sensor.GYRO_ACC)
8 robot.sensors.gyro_acc.start_fusion()
9
10 # Smyčka čekající na ukončení programu
11 while True:
12     # Zobrazení posledních hodnot zrychlení a úhlové rychlosti
13     a_g_data = robot.sensors.gyro_acc.raw()
```

```

14 print("Acc:",
15       a_g_data[GyroAcc.AX],
16       a_g_data[GyroAcc.AY],
17       a_g_data[GyroAcc.AZ],
18       "Gyro:",
19       a_g_data[GyroAcc.GX],
20       a_g_data[GyroAcc.GY],
21       a_g_data[GyroAcc.GZ])
22
23 # Zobrazení orientace kostky
24 euler_angles = robot.sensors.gyro_acc.euler_angles()
25 print("Yaw:", euler_angles[0],
26       "Pitch:", euler_angles[1],
27       "Roll:", euler_angles[2])
28
29 # Získání stavu tlačítek
30 buttons = robot.buttons.pressed()
31 # Ukončení programu, pokud je levé tlačítko stisknuté
32 if buttons[Button.LEFT]:
33     break
34 # Uspání programu na 1 sekundu
35 sleep(1)
36
37 # Deinicializace senzoru
38 robot.deinit_sensor(sensor_type=Sensor.GYRO_ACC)

```

Kód 7: Získávání raw dat a orientace kostky z gyroskopu a akcelerometru.

3.7 I2C master

```
class I2C_master()
```

Funkce pro nastavení kostky jako I2C master zařízení pro mezikostkovou komunikaci. Propojení Open-Cube kostek je možné pomocí I2C portu a speciálně upraveného kabelu. Pro správné fungování jedna kostka musí být nastavena jako I2C master a ostatní jako I2C slave. I2C master čte nebo zapisuje data do 252 registrů I2C slave.

Inicializace: robot.init_i2c_master()

Deinicializace: robot.deinit_i2c_master()

Přístup: robot.i2c_master

```
read(slave_add, len=1, mem_add=None)
```

Přečte data z I2C slave zařízení.

Parametry: **slave_add** (8-119)) – Adresa I2C slave zařízení.

len (int) – Počet registrů (bytů) k přečtení.

mem_add (0-251) – Adresa registru k přečtení. Pokud je None, čte se od poslední čtené nebo zapisované adresy.

Vrací: Data z I2C slave zařízení.

Typ: bytes

```
write(slave_add, message, mem_add=None)
```

Zapíše data do I2C slave zařízení.

Parametry: **slave_add** (8-119) – Adresa I2C slave zařízení.
message (bytes) – Data k zapsání.
mem_add (0-251) – Adresa registru k zápisu. Pokud je None, zapisuje se od poslední čtené nebo zapisované adresy.
Vrací: True, pokud je zapisování úspěšné, False, pokud není.
Typ: bool

trigger(*time_us*, *slave_add*, *callback=None*)

Nastaví probuzení I2C slave zařízení. I2C slave musí mít předem nastavenou callback funkci.

Parametry: **time_us** (int) – Čas (us), za jak dlouho bude zavolána nastavená callback funkce.
slave_add (8-119) – Adresa I2C slave zařízení.
callback (callable) – Callback funkce, která je zavolána v I2C master zařízení po uplynutí zadaného času.
Vrací: True, pokud je nastavení úspěšné, False, pokud není.
Typ: bool

trigger_all(*time_us*, *slave_adds=None*, *callback=None*)

Nastaví probuzení několika I2C slave zařízení. I2C slave musí mít předem nastaveny callback funkce.

Parametry: **time_us** (int) – Čas (us), za jak dlouho bude zavolána nastavená callback funkce.
slave_adds (list) – Seznam adres I2C slave zařízení. Pokud je None, provede sken I2C sběrnice a trigger nastaví pro všechny nalezená zařízení.
callback (callable) – Callback funkce, která je zavolána v I2C master zařízení po uplynutí zadaného času.
Vrací: True, pokud je nastavení úspěšné, False, pokud není (například pokud je zvolený čas pro zavolání callback funkce příliš krátký).
Typ: bool

scan()

Provede sken I2C sběrnice.

Vrací: Seznam adres I2C slave zařízení dostupných na I2C sběrnici.
Typ: list

```
1 # Inicializace kostky jako I2C master
2 robot.init_i2c_master()
3
4 # Scan I2C sběrnice pro získání všech adres I2C slave zařízení
5 all_adds = robot.i2c_master.scan()
6
7 # Zapsání a přečtení znaku "a" (97) do I2C slave zařízení s adresou 10
  do registru 1
8 robot.i2c_master.write(10, (97).to_bytes(1, 'little'), mem_add=1)
```

```

9 data_read = robot.i2c_master.read(10, 1, mem_add=1)
10 print(chr(int.from_bytes(data_read, 'little'))) # "a"
11
12 # Callback funkce pro synchronizaci mezi kostkami
13 def master_trigger_callback(timer):
14     robot.led.on()
15
16 # Nastavení triggeru ve všech nalezených I2C slave zařízeních za 1 s
17     (1000000 us)
18 robot.i2c_master.trigger_all(1000000, slave_adds=all_adds, callback=
19     master_trigger_callback)
20
21 # Deinicializace I2C master
22 robot.deinit_i2c_master()

```

Kód 8: Použití kostky pro mezikostkovou komunikaci jako I2C master.

3.8 I2C slave

```
class I2C_slave()
```

Funkce pro nastavení kostky jako I2C slave zařízení pro mezikostkovou komunikaci. Propojení Open-Cube kostek je možné pomocí I2C portu a speciálně upraveného kabelu. Pro správné fungování jedna kostka musí být nastavena jako I2C master a ostatní jako I2C slave. I2C master čte nebo zapisuje data do 252 registrů I2C slave.

Adresa I2C slave zařízení může být zvolena v rozsahu 8–119. Rezervované adresy I2C jsou odmítnuty.

Inicializace: robot.init_i2c_slave(address)

Deinicializace: robot.deinit_i2c_slave()

Přístup: robot.i2c_slave

```
read(mem_add)
```

Přečte registr.

Parametry: **mem_add** (0-251) – Adresa registru k přečtení.

Vrací: Data z registru.

Typ: int

```
read_all()
```

Přečte adresy 0–254. Adresy 0–251 jsou uživatelské registry a adresy 252–254 patří oblasti časových triggerů.

Vrací: Data ze všech dostupných registrů.

Typ: bytes

```
write(mem_add, data)
```

Zapíše do registru.

Parametry: **mem_add** (0-251) – Adresa registru k zapsání.

data (0-255) – Data k zapsání.

`read_flags()`

Vrátí čtyři bitová pole udávající, které registry četl I2C master.

`written_flags()`

Vrátí čtyři bitová pole udávající, do kterých registrů zapisoval I2C master.

`irq(handler=None, trigger=0)`

Nastaví interrupt handler funkci.

Parametry: **handler** (callable) – Volaná funkce, musí mít jeden vstupní argument pro I2C_slave instanci.

trigger (0-7) – Maska události, při které je volaná handler funkce.

`robot.i2c_slave.RECEIVED` – I2C master zapsal data do registru.

`robot.i2c_slave.READ` – I2C master přečetl data z registru.

`robot.i2c_slave.TRIGGER` – I2C master trigger.

```
1 # Inicializace kostky jako I2C slave s adresou 10
2 robot.init_i2c_slave(10)
3
4 # Callback funkce pro reakci na události
5 def slave_callback(i2c_slave):
6     flags = i2c_slave.irq().flags()
7     if (flags & i2c_slave.RECEIVED):
8         print("new data received")
9     if (flags & i2c_slave.READ):
10        print("master read data")
11    if (flags & i2c_slave.TRIGGER):
12        robot.led.on()
13
14 # Nastavení interrupt handleru pro všechny tři možné události
15 flags = robot.i2c_slave.RECEIVED | robot.i2c_slave.READ | robot.
16        i2c_slave.TRIGGER
17 robot.i2c_slave.irq(handler=slave_callback, trigger=flags)
18
19 # Zapsání a přečtení znaku "b" (98) do registru 2
20 robot.i2c_slave.write(2, 98)
21 data_received = robot.i2c_slave.read(2)
22 print(chr(data_received)) # "b"
23
24 # Deinicializace I2C slave
25 robot.deinit_i2c_slave()
```

Kód 9: Použití kostky pro mezikostkovou komunikaci jako I2C slave.

3.9 ESP32 komunikace

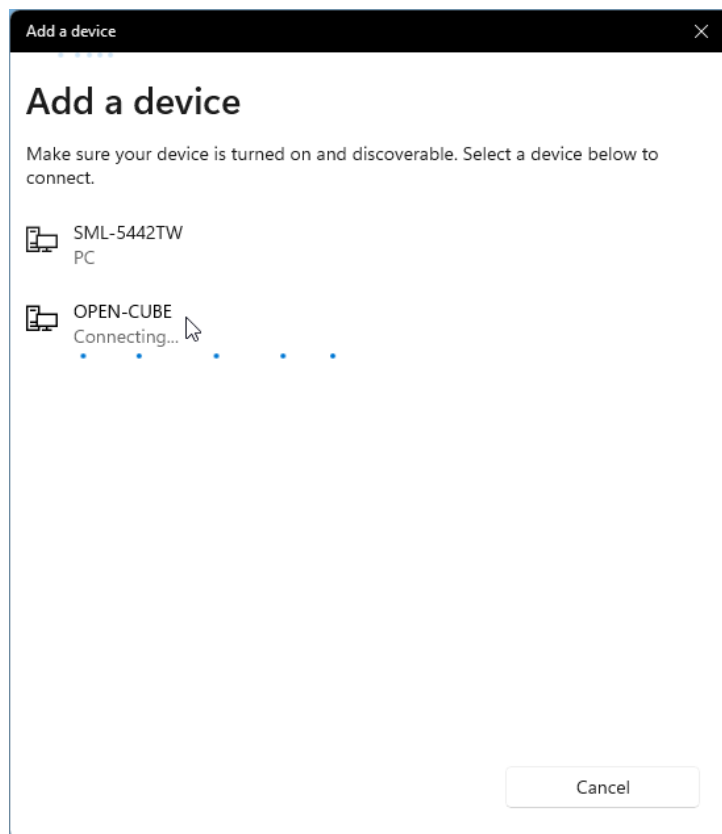
ESP32 komunikuje s RP2040 v kostce interní sériovou linkou. Pracuje v jednom ze dvou vzájemně se vylučujících režimů: režim Bluetooth SPP poskytuje virtuální sériové spojení

s počítačem nebo mobilem, zatímco režim Wi-Fi access point poskytuje webové rozhraní s tlačítky, přepínači, indikátory a číselnými hodnotami. Přepnutí do Wi-Fi ukončí Bluetooth SPP; návrat do Bluetooth ukončí webový server a odpojí access point. ESP32 lze programovat přes USB konektor v horní části kostky. Výchozí firmware je dostupný ze stránky [Open-Cube releases](https://gitlab.fel.cvut.cz/open-cube/firmware/-/releases) [<https://gitlab.fel.cvut.cz/open-cube/firmware/-/releases>].

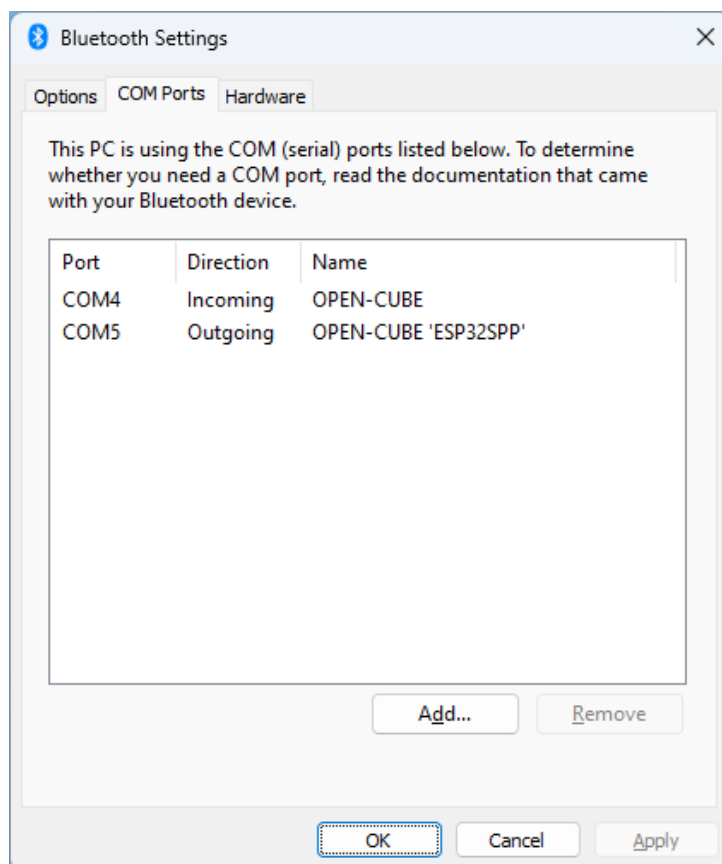
3.9.1 Bluetooth propojení ESP32 s jiným zařízením

Komunikace probíhá sériově přes Bluetooth virtuální COM port a je otestována pro Windows, Ubuntu a Android. Pro spárování musí být kostka zapnutá v menu BT pair. Na displeji se zobrazí název zařízení.

- Windows:
 1. V Bluetooth nastavení zvolte Přidat nové zařízení, dále Bluetooth a spárujte počítač s kostkou. Viz obrázek 2.
 2. Potvrďte pin na kostce a na počítači.
 3. 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.
- Ubuntu:
 1. V Bluetooth nastavení zvolte zařízení Open-Cube a spárujte počítač s kostkou.
 2. Potvrďte pin na kostce a na počítači.
 3. Zobrazte podrobnosti o zařízení a zkopírujte si jeho MAC adresu.
 4. Vytvořte serial port link: `sudo rfcomm bind /dev/rfcomm0 XX:XX:XX:XX:XX:XX`.
 5. Sériový port otevřete například příkazem `screen /dev/rfcomm0`
- Android:
 1. V Bluetooth nastavení zvolte zařízení Open-Cube a spárujte mobil s kostkou.
 2. Potvrďte pin na kostce a na mobilu.



Obrázek 2: Spárování počítače s kostkou.



Obrázek 3: Zjištění čísla COM portu.

3.9.2 Kostka

```
class ESP()
```

Oboustranné posílání dat mezi mikrokontrolérem a ESP32 pomocí rozhraní UART. 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(timeout=2000)
```

Resetuje ESP, nastaví režim Bluetooth a po potvrzení vrátí True, při vypršení časového limitu False.

```
bt()
```

Přepne z režimu Wi-Fi access point do režimu Bluetooth SPP. Je-li Bluetooth již aktivní, metoda nic nezmění.

```
wifi()
```

Nastaví režim Wi-Fi access point. Výchozí adresa ESP je 192.168.4.1. Indexace nastavitelných prvků webového serveru je na obrázku 4.

```
set_name(name)
```

Nastaví jméno pro Bluetooth a Wi-Fi SSID. Běžný start kostky použije jméno zvolené v Settings, zpočátku "Open-Cube01".

Parametry: `name` (str) – Jméno ESP.

```
set_password(password)
```

Nastaví Wi-Fi heslo ESP. Použijte 8–63 znaků ASCII. 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(*message*)

Odešle zprávu v Bluetooth režimu.

Parametry: **message** (str/bytes) – Textová nebo binární data k odeslání.

bt_message_ready()

Vrátí True, pokud je dostupná úplná zpráva.

bt_set_binary(*header=None, data_size=0*)

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.

bt_set_callback(*callback=None*)

Nastaví funkci volanou jako **callback(esp)** při přijetí úplné zprávy.

Parametry: **callback** (callable) – Callback funkce. Pokud je None, callback se zruší.

bt_reset()

Vymaže přijatá data a vrátí režim ASCII.

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 popisů 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 popisů 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 popisů 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 popisů čísel.

Příklad použití komunikace mezi mikroprocesorem a ESP32:

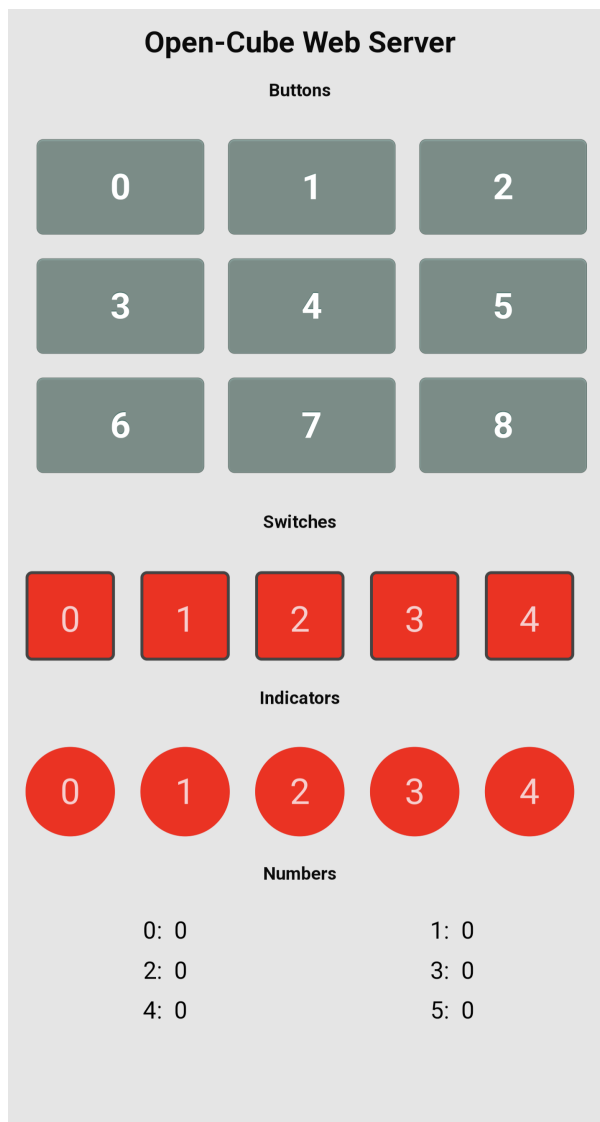
```
1 import struct
2 from time import sleep_ms
3
4 # Odeslání textu externímu zařízení Bluetooth
5 robot.esp.bt_write("test message\n")
6
7 # Čekání na odpověď externího zařízení
8 while not robot.esp.bt_message_ready():
9     sleep_ms(10)
10 received_message = robot.esp.bt_read()
11
12 # Odeslání a přijetí zprávy o velikosti 8 bytů v binárním režimu
13 robot.esp.bt_set_binary((112, 241, 3, 62), 8)
14 # Vytvoření a odeslání zprávy typu bytes se dvěma čísly typu single
15 write_message = struct.pack("<ff", 521.9, -11.821)
16 robot.esp.bt_write(write_message)
17 # Čekání na binární odpověď externího zařízení
18 while not robot.esp.bt_message_ready():
19     sleep_ms(10)
20 received_message2 = robot.esp.bt_read()
21 value1, value2 = struct.unpack("<ff", received_message2)
22
23 # Přepne ESP do Wi-Fi access point režimu
24 robot.esp.wifi()
```

```

25 # Nastaví popisky indikátorů
26 robot.esp.wifi_set_indicators_labels(("1", "2", "3", "4", "5"))
27 # Zjistí stav přepínačů
28 switches = robot.esp.wifi_get_switches()

```

Kód 10: Komunikace mezi mikroprocesorem a ESP32.



Obrázek 4: Indexace prvků ESP web serveru.

3.9.3 Dálkové ovládání kostky

Pokud v režimu Bluetooth běží framework kostky, lze menu ovládat, zjišťovat stav kostky a spravovat programy řádkovými příkazy ukončenými LF nebo CR.

Ovládací zprávy:

- `oc u`, `oc d`, `oc l`, `oc r`, `oc k` – emulují tlačítka UP, DOWN, LEFT, RIGHT a OK,
- `oc ko` – emuluje tlačítko napájení a vypne kostku,
- `oc ping` – odpoví `oc pong`,

- `oc fw` – otevře USB bootloader RP2040,
- `oc rs` – restartuje RP2040,
- `oc vfw` – vrátí datum a commit firmwaru,
- `oc bat` – vrátí napětí a procento baterie,
- `oc nam` – vrátí nastavené jméno kostky,
- `oc mst` – vrátí aktuální stránku menu, vybranou položku a stav programu,
- `oc ls` – vrátí seznam programů v `/programs` oddělený čárkami,
- `oc del <name>` – smaže soubor nebo adresář programu a vrátí `oc del ok` nebo `oc del ko`,
- `oc run <name>` – spustí program bez sledování výstupu,
- `oc rtc <name>` – spustí program v režimu převzetí řízení, který sleduje jeho výstup a ukončení,
- `oc stp` – zastaví řízený program vyvoláním `KeyboardInterrupt`,
- `oc prg <filename>` – spustí binární protokol nahrávání používaný pluginem Thonny a nástrojem PC.

Kostka odešle `oc menu`, když začne ovládání menu přes Bluetooth nebo se řízený program vrátí do menu. Názvy programů jsou omezeny na konečný název souboru nebo adresáře v `/programs`; prázdné názvy, `.`, `..` a názvy obsahující dvojtečku jsou odmítnuty.

3.9.4 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) [https://play.google.com/store/apps/details?id=de.kai_morich.serial_bluetooth_terminal]. Aplikace umožňuje odesílat a přijímat zprávy kostky v režimu Bluetooth ASCII. Zprávy a příkazy musí končit znakem LF nebo CR.

3.9.5 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/ref/bluetooth.html) [<https://www.mathworks.com/help/matlab/ref/bluetooth.html>].

```
1 % Připojení kostky k Matlabu
2 cube = bluetooth('Open-Cube01')
3 % Nastavení terminátoru LF pro odesílání a příjem ASCII dat
4 configureTerminator(cube, 'LF')
```

```

5 %%
6 % Poslání testovací zprávy v ASCII režimu Bluetooth; zpráva
  se automaticky ukončí nastaveným terminátorem.
7 writeline(cube, 'test message')
8 % Příjem zprávy; funkce čeká na nastavený terminátor.
9 cube_message = readline(cube)
10 %%
11 % Kostka používá stejnou čtyřbytovou hlavičku a osm bytů dat.
12 header = uint8([112 241 3 62]);
13 payload = typecast(single([521.9 -11.821]), 'uint8');
14 write(cube, [header payload], 'uint8')
15
16 % Přečte odpověď a převede její data na dvě hodnoty single.
17 frame = read(cube, 12, 'uint8');
18 if isequal(frame(1:4), header)
19     cube_values = typecast(uint8(frame(5:12)), 'single');
20 end

```

Kód 11: Komunikace s kostkou z Matlabu.

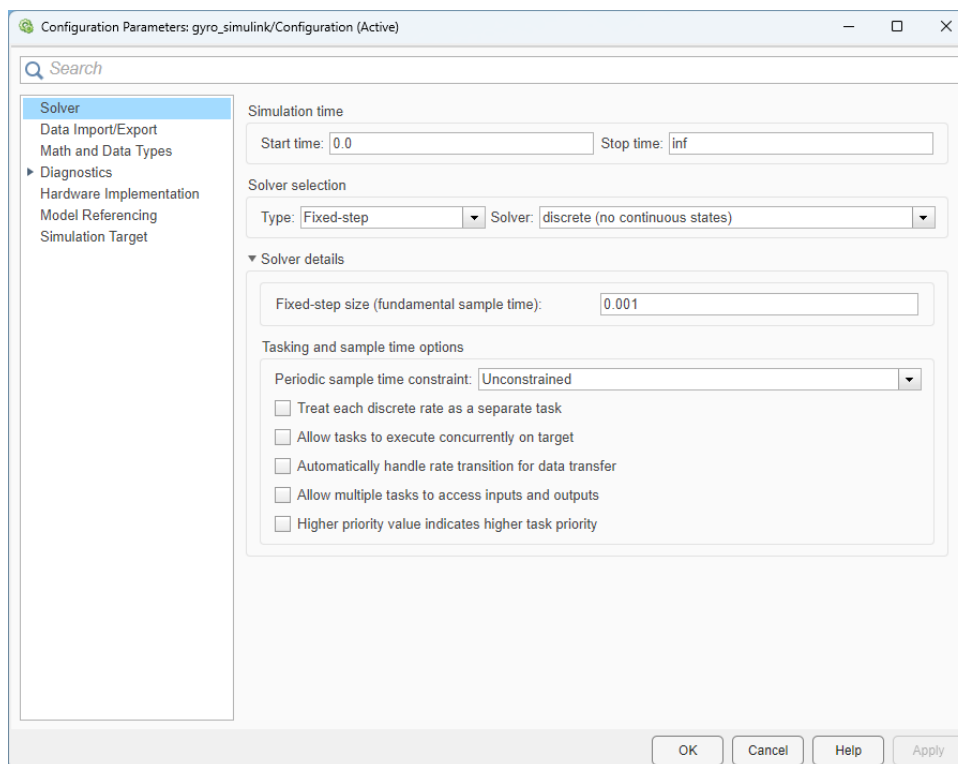
3.9.6 Simulink

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 solveru simulace a nastavit parametry sériové komunikace:

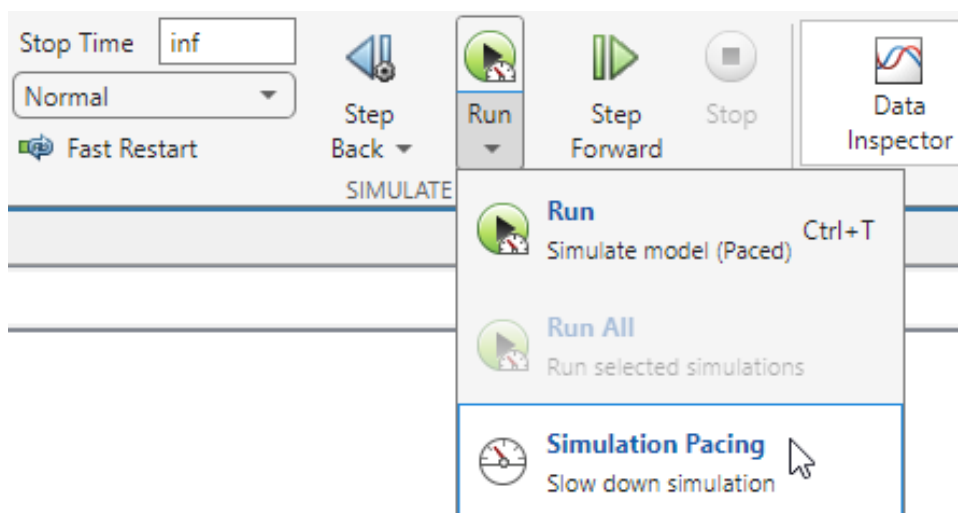
1. Kliknutím na tlačítko v pravém dolním rohu nastavit solver na Fixed-Step, discrete a Fixed-step size na hodnotu podle frekvence odesílání a přijímání dat (např. 0,001). Viz obrázek 5
2. Kliknutím na šipku pod tlačítkem Run zapnout Simulation Pacing, viz obrázky 6 a 7.
3. V modelu vytvořit blok Serial Configuration, ve kterém nastavit COM port, zjištěný v kapitole 3.9.1 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) [<https://www.mathworks.com/help/instrument/direct-interface-communication-in-simulink.html>].

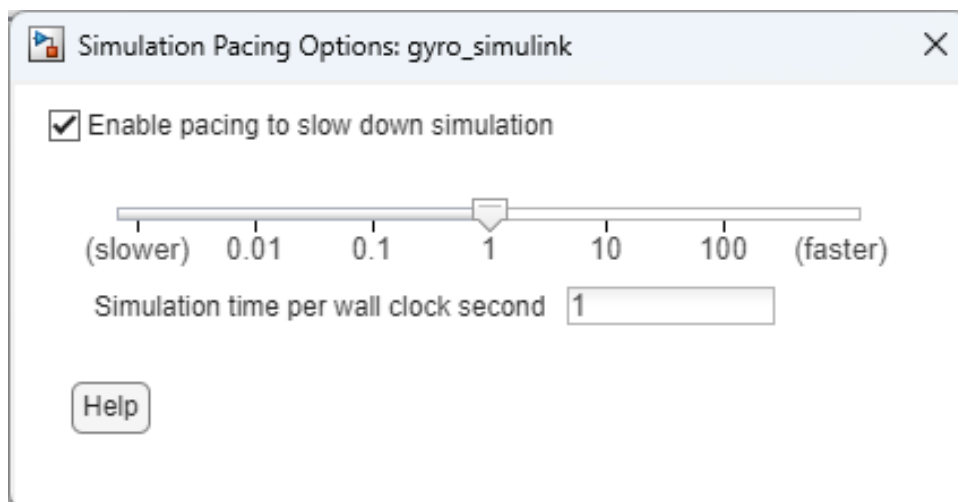
Příklad použití sériové komunikace pro nastavení konstant PID regulátoru v kostce a zobrazení aktuálních složek PID je na obrázku 9. V tomto příkladu se hodnoty odesílají a přijímají v binárním formátu, je nastavena hlavička pro odesílání a přijímání a data jsou typu single (4 byty). Data se přijímají 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 solveru by měl být menší než Block sample time.



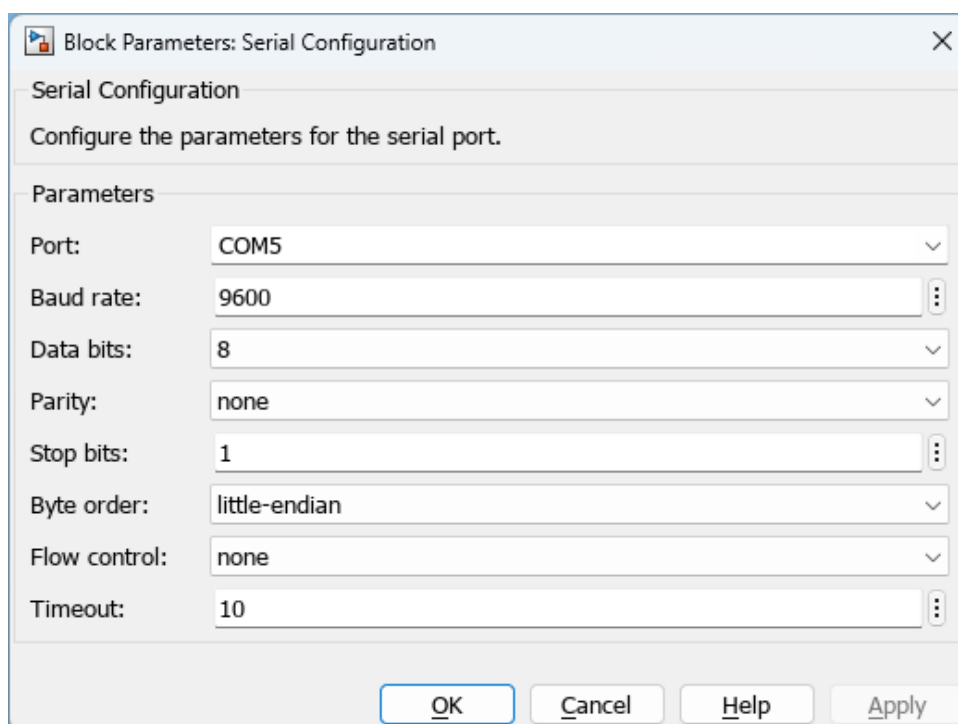
Obrázek 5: Nastavení solveru Simulinku.



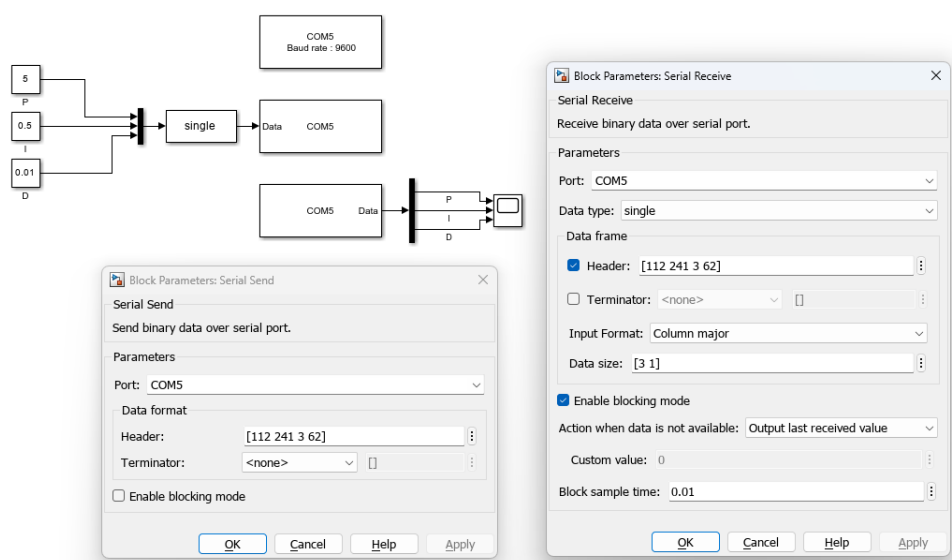
Obrázek 6: Umístění nastavení Simulation pacing v Simulinku.



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

```
1 # Importování konstant typů senzorů a portů
2 from lib.robot_consts import Sensor, Port
3
4 # Inicializace NXT tlačítka na senzorovém portu 1
5 robot.init_sensor(sensor_type=Sensor.NXT_TOUCH, port=Port.S1)
6
7 # Zjištění stavu stisknutí tlačítka
8 touch_pressed = robot.sensors.touch[Port.S1].pressed()
```

Kód 12: 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 neblíkající a blikající. Při nastavení neblíkajícího 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.

toggle()

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 navracené změřené intenzity dopadajícího světla. V manuálním módu a v kontinuálním módu s neblíkajícím 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 neblíkajícího režimu.

```

1 # Importování konstant typů senzorů a portů
2 from lib.robot_consts import Sensor, Port
3
4 # Inicializace NXT světelného senzoru na senzorovém portu 1
5 robot.init_sensor(sensor_type=Sensor.LIGHT, port=Port.S1)
6
7 # Zapnutí LED senzoru
8 robot.sensors.light[Port.S1].on()
9 # Změření intenzity světla
10 light_intensity = robot.sensors.light[Port.S1].intensity()

```

Kód 13: 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. Tento port současně nelze použít pro mezikostkovou komunikaci I2C master nebo slave.

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)

```

1 # Importování konstant typů senzorů
2 from lib.robot_consts import Sensor
3
4 # Inicializace NXT ultrazvukového senzoru
5 robot.init_sensor(sensor_type=Sensor.NXT_ULTRASONIC)
6
7 # Změření vzdálenosti
8 distance = robot.sensors.ultra_nxt.distance()

```

Kód 14: 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)

```
1 # Importování konstant typů senzorů a portů
2 from lib.robot_consts import Sensor, Port
3
4 # Inicializace NXT zvukového senzoru na senzorovém portu 1
5 robot.init_sensor(sensor_type=Sensor.NXT_SOUND, port=Port.S1)
6
7 # Změření hladiny zvuku
8 sound_intensity = robot.sensors.sound[Port.S1].intensity()
```

Kód 15: Použití NXT zvukového senzoru.

5 EV3 senzory

EV3 UART senzory jsou během otevřeného menu kostky detekovány na pozadí. Inicializace odpovídajícího senzoru použije detekované připojení a nemusí opakovat celé navázání komunikace.

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

```
1 # Importování konstant typů senzorů a portů
2 from lib.robot_consts import Sensor, Port
3
4 # Inicializace EV3 tlačítka na senzorovém portu 1
5 robot.init_sensor(sensor_type=Sensor.EV3_TOUCH, port=Port.S1)
6
7 # Zjištění stavu stisknutí tlačítka
8 touch_pressed = robot.sensors.touch[Port.S1].pressed()
```

Kód 16: 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, port=Port)`
Deinicializace: `robot.deinit_sensor(sensor_type=Sensor.EV3_COLOR, port=Port)`
Přístup: `robot.sensors.light[Port]`

```
reflection()
```

Navrátí poslední změřenou odraženou intenzitu světla. Červená LED se střídavě zapíná a vypíná. Hodnota se zapnutou LED se odečte od hodnoty s 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. Hodnota se zapnutou LED se odečte od hodnoty s vypnutou LED a výsledek se omezí na nulu nebo větší hodnotu.

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 nebo None, pokud není rozpoznána žádná barva.

Typ: int (1-7 – black, blue, green, yellow, red, white, brown)

```

1 # Importování konstant typů senzorů
2 from lib.robot_consts import Sensor, Port
3
4 # Inicializace EV3 světelného senzoru na sensorovém portu 1
5 robot.init_sensor(sensor_type=Sensor.EV3_COLOR, port=Port.S1)
6 # Zjistění intenzity odraženého světla
7 reflection = robot.sensors.light[Port.S1].reflection()

```

Kód 17: 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, port=Port)`
Deinicializace: `robot.deinit_sensor(sensor_type=Sensor.EV3_ULTRASONIC, port=Port)`
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

```

1 # Importování konstant typů senzorů
2 from lib.robot_consts import Sensor, Port
3
4 # Inicializace EV3 ultrazvukového senzoru na sensorovém portu 1
5 robot.init_sensor(sensor_type=Sensor.EV3_ULTRASONIC, port=Port.S1)
6
7 # Změření vzdálenosti
8 distance = robot.sensors.ultrasonic[Port.S1].distance()

```

Kód 18: 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, port=Port)`
Deinicializace: `robot.deinit_sensor(sensor_type=Sensor.EV3_INFRARED, port=Port)`
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: 1–4)` – Analyzovaný kanál ovladače.
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 %)

```
1 # Importování konstant typů senzorů
2 from lib.robot_consts import Sensor, Port
3
4 # Inicializace EV3 ultrazvukového senzoru na senzorovém portu 1
5 robot.init_sensor(sensor_type=Sensor.EV3_INFRARED, port=Port.S1)
6
7 # Změření vzdálenosti
8 distance = robot.sensors.infrared[Port.S1].distance()
```

Kód 19: 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, port=Port)`
Deinicializace: `robot.deinit_sensor(sensor_type=Sensor.EV3_GYRO, port=Port)`
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. Hodnota 0 úhel vynuluje.

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)

tilt_speed()

Navrátí úhlovou rychlost kolem vedlejší osy senzoru ve °/s.

tilt_angle()

Navrátí natočení kolem vedlejší osy senzoru ve stupních. Měření vedlejší osy nepodporují všechny verze EV3 gyroskopu.

```
1 # Importování konstant typů senzorů
2 from lib.robot_consts import Sensor, Port
3
4 # Inicializace EV3 gyroskopu na senzorovém portu 1
5 robot.init_sensor(sensor_type=Sensor.EV3_GYRO, port=Port.S1)
6
7 # Vynulování úhlu natočení
8 robot.sensors.gyro[Port.S1].reset_angle(0)
9
10 # Změření úhlu natočení a úhlové rychlosti
11 (angle, speed) = robot.sensors.gyro[Port.S1].angle_and_speed()
```

Kód 20: Použití EV3 gyroskopu.

6 Open-Cube senzory

Open-Cube barevný, laserový, ultrazvukový a AHRS senzor používají stejné detekování UART na pozadí a opětovné použití připojení jako EV3 senzory. Inicializace senzoru použije detekované připojení a nemusí opakovat celé navázání komunikace.

6.1 Tlačítko

```
class TouchSensor(port)
```

Open-Cube Touch Sensor používá stejné rozhraní tlačítka jako EV3 Touch Sensor.

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

```
pressed()
```

Vrátí True, pokud je senzor stisknutý, jinak False.

6.2 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,port=Port)`
Deinicializace: `robot.deinit_sensor(sensor_type=Sensor.OC_COLOR,port=Port)`
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 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)

```
1 # Importování konstant typů senzorů
2 from lib.robot_consts import Sensor, Port
3
4 # Inicializace Open-Cube světelného senzoru na senzorovém portu 1
5 robot.init_sensor(sensor_type=Sensor.OC_COLOR, port=Port.S1)
6
7 # Zjistění intenzity odraženého světla
8 reflection = robot.sensors.light[Port.S1].reflection()
```

Kód 21: Použití Open-Cube světelného senzoru.

6.3 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, port=Port)`

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

Přístup: `robot.sensors.laser[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.

```
1 # Importování konstant typů senzorů
2 from lib.robot_consts import Sensor, Port
3
4 # Inicializace EV3 ultrazvukového senzoru na sensorovém portu 1
5 robot.init_sensor(sensor_type=Sensor.OC_LASER, port=Port.S1)
6
7 # Změření vzdálenosti
8 distance = robot.sensors.laser[Port.S1].distance()
```

Kód 22: Použití Open-Cube laserového dálkoměru.

6.4 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**, port=**Port**)

Deinicializace: robot.deinit_sensor(sensor_type=**Sensor.OC_ULTRASONIC**, port=**Port**)

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.

```
1 # Importování konstant typů senzorů
2 from lib.robot_consts import Sensor, Port
3
4 # Inicializace Open-Cube ultrazvukový senzor na sensorovém portu 1
5 robot.init_sensor(sensor_type=Sensor.OC_ULTRASONIC, port=Port.S1)
6
7 # Změření vzdálenosti
8 distance = robot.sensors.ultrasonic[Port.S1].distance()
```

Kód 23: Použití Open-Cube ultrazvukového dálkoměru.

6.5 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, port=Port)`

Deinicializace: `robot.deinit_sensor(sensor_type=Sensor.OC\_GYRO, port=Port)`

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: -180–180°, roll: -90–90°)

euler_angles_mag()

Navrátí aktuální orientaci senzoru v Eulerových úhlech vypočtenou z gyroskopu, akcelerometru a magnetometru. Pokud volitelný magnetometr není dostupný, senzor zůstane použitelný a tento režim použije fúzi gyroskopu a akcelerometru.

Vrací: Tuple Eulerových úhlů yaw, pitch, roll.

Typ: (yaw: -180–180°, pitch: -180–180°, roll: -90–90°)

quaternions()

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

Vrací: Tuple kvaternionů.

Typ: (0–1, 0–1, 0–1, 0–1)

reset_angles()

Nastaví aktuální orientaci jako nulovou referenci. Nejprve zavolejte `euler_angles()`, `euler_angles_mag()` nebo `quaternions()` pro výběr režimu orientace. Příkaz se ignoruje v raw a kalibračních režimech a sám režim nemění.

raw()

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

Vrací: Tuple obsahující tuple gyroskopu, akcelerometru a magnetometru.

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 ve všech orientacích a tuto metodu volejte, dokud nevrátí 1. Dokončení sleduje orientace akcelerometru, nikoli stav magnetometru; dokončená kalibrace neověřuje platnost magnetických měření.

Vrací: 1 pokud je kalibrace dokončena.

Typ: int (0/1)

```
1 # Importování konstant typů senzorů
2 from lib.robot_consts import Sensor, Port
3
4 # Inicializace OC AHRS senzoru na senzorovém portu 1
5 robot.init_sensor(sensor_type=Sensor.OC_GYRO, port=Port.S1)
6
7 # Kalibrace gyroskopu
8 calibrated = 0
9 while not calibrated:
10     calibrated = robot.sensors.gyro[Port.S1].calibrate_gyro()
11
12 # Zjištění orientace senzoru
13 euler_angles = robot.sensors.gyro[Port.S1].euler_angles()
14 yaw, pitch, roll = euler_angles[0], euler_angles[1], euler_angles[2]
```

Kód 24: Použití Open-Cube AHRS senzoru.

7 Servomotory

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.motors[*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.

```
reset_encoder()
```

Vynuluje polohu enkodéru zvoleného 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 režimu 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 rychlosti.

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 %)

regulator_speed()

Navrátí filtrovanou rychlost používanou regulátorem rychlosti ve °/s.

```
1 # Importování konstant portů
2 from lib.robot_consts import Port
3
4 # Inicializace motorů na motorových portech 1 a 2
5 robot.init_motor(Port.M1)
6 robot.init_motor(Port.M2)
7
8 # Nastavení výkonu motorů na 50 % a 20 % v opačném směru otáčení
9 robot.motors[Port.M1].set_power(50)
10 robot.motors[Port.M2].set_power(-20)
11
12 # Inicializace enkodérů pro měření pozice a rychlosti motorů
13 robot.motors[Port.M1].init_encoder()
14 robot.motors[Port.M2].init_encoder()
15
16 # Zjištění aktuální pozice a rychlosti motorů
17 pos1 = robot.motors[Port.M1].position()
18 pos2 = robot.motors[Port.M2].position()
19 speed1 = robot.motors[Port.M1].speed()
20 speed2 = robot.motors[Port.M2].speed()
```

Kód 25: 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_ULTRASONIC

NXT ultrazvukový dálkoměr.

NXT_TOUCH

NXT tlačítko.

NXT_SOUND

NXT zvukový senzor.

GYRO_ACC

Vestavěný gyroskop a akcelerometr ICM42688.

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.

OC_TOUCH

Open-Cube tlačítko.

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.

M_COUNT

Počet portů motorů.

S1

Port senzoru 1.

S2

Port senzoru 2.

S3

Port senzoru 3.

S4

Port senzoru 4.

S_COUNT

Počet portů senzorů.

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 mikrokontroléru, ke kterému je připojen ICM42688.

8.6 Color

class Color

Barvy detekované EV3 a Open-Cube barevnými senzory. Open-Cube Color může vrátit `NO_COLOR`; metoda EV3 Color na vysoké úrovni vrátí `None`, pokud není rozpoznána žádná barva.

NO_COLOR

Nebyla rozpoznána žádná barva.

BLACK

BLUE

GREEN

YELLOW

RED

WHITE

BROWN

8.7 Note

class Note

Frekvence používané pro melodie bzučáku. Konstanty C4–B4, C5–B5 a C6–B6 obsahují frekvence not v Hz. REST má frekvenci nula.

8.8 Melody

class Melody

Předdefinované sekvence přijímané metodou `robot.buzzer.play_melody()`.

ASCENDING

DESCENDING

TWINKLE

SUCCESS

FAILURE

ALERT

STARTUP

SHUTDOWN

HAPPY_BIRTHDAY

SUPER_MARIO

ODE_TO_JOY

IMPERIAL_MARCH

9 Užitečné funkce MicroPythonu

9.1 Výpis informací

```
1 x, y = 64, 128.4096
2 y_string = "y"
3
4 # 1. varianta - oddělení čárkami
5 # Tato varianta je vhodná na jednoduché a rychlé zobrazení textu
6 # Jednotlivými prvky jsou odděleny mezerou
7 print("x =", x, ",", y_string, "=", y)
8 # x = 64, y = 128.4096
9
10 # 2. varianta - použití f-stringu
11 # Tato varianta umožňuje formátovat float čísla
12 # d označuje typ integer, s typ string a f typ float
13 # .2 určuje počet zobrazených desetinných míst
14 print(f"x = {x:d}, {y_string:s} = {y:.2f}")
15 # x = 64, y = 128.41
16
17 # Použití f-stringu je vhodné i pro zobrazení na displeji kostky
18 robot.display.fill(0)
19 # první řádek displeje
20 robot.display.text(f"x = {x:d}", 0, 0, 1)
21 # druhý řádek displeje
22 robot.display.text(f"{y_string:s} = {y:.2f}", 0, 10, 1)
23 robot.display.show()
24
25 # Výpis bez odřádkování
26 for i in range(10)
27     print(i, end="")
28 print() # odřádkování na konci
29 # 123456789
```

Kód 26: Výpis informací.

9.2 Generování náhodných čísel

Ukázka generování náhodných čísel.

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

```
1 # Importování funkcí knihovny random
2 from random import random, randint
3
4 random() # Náhodné číslo typu float v intervalu [0.0, 1.0)
5
6 a, b = 0, 100
7 randint(a, b) # Náhodné číslo typu int (celé číslo) v intervalu [a, b]
```

Kód 27: 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>]

```
1 # Importování funkcí knihovny sleep
2 from time import sleep, sleep_ms, sleep_us, ticks_us, ticks_diff
3
4 start_time = ticks_us() # Zjištění počátečního času v mikrosekundách
5
6 sleep(1) # Uspání programu na 1 sekundu
7 sleep_ms(2) # Uspání programu na 2 milisekundy
8 sleep_us(3) # Uspání programu na 3 mikrosekundy
9
10 end_time = ticks_us() # Zjištění koncového času v mikrosekundách
11 print(ticks_diff(end_time, start_time), "us") # 1002003 us
```

Kód 28: Použití knihovny time.

```
1 # Importování třídy Timer
2 from machine import Timer
3
4 counter = 0
5
6 # Definice callback funkce pro timery
7 def my_callback(t):
8     global counter
9     counter += 1
10    print(counter)
11
12 # Inicializace timeru, který volá callback funkci periodicky
13   s frekvencí 100 Hz
14 # -1 značí virtuální timer (mikrokontrolér rp2040 neumožňuje použití
15   hardwarových timerů)
16 timer1 = Timer(-1)
17 timer1.init(mode=Timer.PERIODIC, freq=100, callback=my_callback)
18
19 # Inicializace timeru, který zavolá callback funkci pouze jednou po
20   uplynutí 1000 ms
21 timer2 = Timer(-1)
22 timer2.init(mode=Timer.ONE_SHOT, period=1000, callback=my_callback)
23
24 # Vypnutí periodického timeru
25 timer1.deinit()
```

Kód 29: 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>]

```

1 # Importování knihovny struct
2 import struct
3
4 # Vytvoření binárních dat ze tří čísel. Formát čísel je určen stringem
  prvního argumentu. Formát stringu je podrobně popsán v dokumentaci
  knihovny struct.
5 # V tomto případě je první číslo uloženo jako typ int (4 byty), druhé
  jako float (4 byty) a třetí jako double (8 bytů)
6 binary_data = struct.pack("@ifd", 12, 1.234, 5)
7
8 # Konverze binárních dat
9 values = struct.unpack("@ifd", binary_data)
10
11 # Spočítá velikost paměti v bytech potřebnou k uložení binárních dat
12 byte_size = struct.calcsize("@ifd")
13 print(byte_size) # 16

```

Kód 30: 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/library/_thread.html]

```

1 # Importování knihovny _thread a funkce sleep
2 import _thread
3 from time import sleep
4
5 lock = _thread.allocate_lock()
6 counter = 0
7
8 # Definice funkce druhého jádra
9 def core2():
10     global counter
11     lock.acquire() # Uzamčení pro práci se společnými daty
12     counter += 1
13     lock.release() # Uvolnění
14     _thread.exit() # Ukončení běhu jádra
15
16 # Aktivace druhého jádra
17 _thread.start_new_thread(core2, ())
18
19 lock.acquire() # Uzamčení pro práci se společnými daty
20 print(counter)
21 lock.release() # Uvolnění

```

Kód 31: 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.).