

# Bibliotheken für die Roboter

Micropython Software-Bibliotheken für den Betrieb der Roboter.

- [Motorsteuerung](#)
- [Ultraschallsensor](#)
- [Infrarotsteuerung](#)
- [Servosteuerung](#)
- [Das komplette Modul "robotlibrary"](#)
- [Dokumentation](#)

# Motorsteuerung

Speichere diesen Code auf dem Pico unter dem Namen "motor.py".

## Motorbibliothek

```
from machine import Pin, PWM
import utime

MIN_DUTY = 0
MAX_DUTY = 60000
MAX_SPEED = 100
MIN_SPEED = 30

class Motor:
    '''This class manages the motor. Don't edit!'''
    def __init__(self, pinNo):
        self.gpio = pinNo
        self.speed=0
        self.forward=True
        self.pwm1=PWM(Pin(pinNo))
        self.pwm1.freq(50000)
        self.pwm1.duty_u16(0)
        self.pwm2=PWM(Pin(pinNo+1))
        self.pwm2.freq(50000)
        self.pwm2.duty_u16(0)
        self.speed_offset = 0

    def set_speed(self,s):
        '''Sets the speed of the motor. Checks for sensible input.'''
        if s + self.speed_offset <= MIN_SPEED:
            s = 0
            self.reset_offset()
        elif s + self.speed_offset >= MAX_SPEED:
            s = MAX_SPEED
        self.pwm1.duty_u16(int(MAX_DUTY*(s+self.speed_offset)/100))
        self.speed=s
```

```

def change_speed(self,sc):
    '''This defines an offset to the speed in motor. It is used with the remote control to
    turn the robot.'''
    if self.speed + sc > MIN_SPEED and self.speed + sc < MAX_SPEED:
        self.speed_offset += sc
        self.set_speed(self.speed)

def reset_offset(self):
    self.speed_offset = 0

def off(self):
    self.pwm1.duty_u16(0)
    self.speed = 0

def set_forward(self,forward):
    '''Sets the motor to forward or backward without changing the speed. '''
    if self.forward==forward:
        return
    self.pwm1.duty_u16(0)
    self.pwm1,self.pwm2=self.pwm2,self.pwm1
    self.forward=forward
    self.set_speed(self.speed)

```

# Beispiel für die Anwendung dieser Bibliothek

Kopiere diesen Code in eine andere Datei auf dem Pico, z. B. „motortest.py“.

```

from motor import Motor
from utime import sleep, sleep_ms

motor = Motor(12)

motor.set_speed(70)
motor.set_forward(True)

```

```
sleep(1)
```

```
motor.off()
```

# Ultraschallsensor

## Ultraschallbibliothek

Speichere diesen Code auf dem Pico unter dem Namen "ultrasonic.py".

```
from machine import Pin
from time import sleep
import utime

class Ultra:
    '''This class manages the ultrasonic sensor. It returns the distance to an obstacle in cm.
    ...
    def __init__(self, pinNo):
        self.trigger = Pin(pinNo, Pin.OUT) # to trigger a sound impulse
        self.echo = Pin(pinNo+1, Pin.IN) # records the echo of the trigger pulse

    def get_dist(self):
        '''This returns the measured distance in cm. (float)'''
        timepassed = 0
        signalon = 0
        signaloff = 0
        self.trigger.low()
        utime.sleep_us(2)
        self.trigger.high()
        utime.sleep_us(5)
        self.trigger.low()
        while self.echo.value() == 0:
            signaloff = utime.ticks_us()
        while self.echo.value() == 1:
            signalon = utime.ticks_us()
        timepassed = signalon - signaloff
        distance = round((timepassed * 0.0343) / 2, 2)
        # print("The distance from object is ", distance, "cm.") # for debugging purposes
        uncomment the line.
        utime.sleep_ms(10) # Wait necessary or program halts
```

return distance

# Beispiel für die Anwendung dieser Bibliothek

Kopiere diesen Code in eine andere Datei auf dem Pico, z. B. „ultratest.py“.

```
from ultrasonic import Ultra
from utime import sleep, sleep_ms
us = Ultra(16)

while True:
    print(f"gemessene Entfernung: {us.get_dist()} cm.")
    sleep(1)
```

Zum Fahren siehe [Motorsteuerung](#).

# Infrarotsteuerung

Ein Infrarotsensor kann zum Erkennen von Hindernissen oder der Verfolgung einer schwarzen Linie genutzt werden. Die folgende Klasse steuert den Sensor.

```
from machine import Pin,Timer

import micropython

micropython.alloc_emergency_exception_buf(100)

class IR:
    '''This class manages the IR-sensor. Write your code in Robot.ir_detected()'''
    def __init__(self, pinNo,callback_func):
        self.out = pinNo
        self.ir_detected = callback_func
        self.ir = Pin(pinNo, Pin.IN, Pin.PULL_UP)
        self.ir.irq(trigger=Pin.IRQ_FALLING | Pin.IRQ_RISING, handler=self.obstacle)
        self.detected=False
        self.timer = Timer()

    def reset_detected(self,t):
        self.detected = False

    def obstacle(self, pin):
        '''This is called on any change in the IR-sensor. '''
        if not self.detected:
            self.ir_detected(pin,self.out)
            self.detected = True
            self.timer.init(mode=Timer.ONE_SHOT, period=100, callback=self.reset_detected)
```

Der Code muss unter dem Dateinamen „infrared.py“ auf dem Pico gespeichert werden.

Der Infrarotsensor wird folgendermaßen im eigenen Programm eingebunden. Das Beispiel ist für zwei Infrarotsensoren.

```
from infrared import IR

IR_PIN_LEFT=0
```

```
IR_PIN_RIGHT=1
```

```
def ir_detected(pin, pinno):  
    print(f"Pin: {pin}, pin number: {pinno}")  
    if pinno == IR_PIN_LEFT:  
        print("links")  
    elif pinno == IR_PIN_RIGHT:  
        print("right")  
  
ir_left = IR(0, ir_detected)  
ir_right = IR(1, ir_detected)
```

# Servosteuerung

Der Ultraschallsensor kann auch mit einem Servomotor drehbar gemacht werden. Die folgende Klasse steuert den Servomotor:

```
from machine import Pin, PWM
import utime

class Servo:
    '''This class manages the servo motor that turns the ultrasonic sensor. You need a servo
motor installed to get use out of this.
    Don't use directly or edit.'''
    def __init__(self, pin):
        self.pin = PWM(Pin(pin))
        self.pin.freq(50)
        self.min = 1350
        self.max = 8100
        self.angle = 0

    def set_angle(self, a):
        '''If installed, the server motor will set the angle of the ultrasonic sensor. 90° ist
straight ahead.'''
        if a > self.angle:
            for i in range(self._get_duty(self.angle), self._get_duty(a)):
                self.pin.duty_u16(i)

        elif a < self.angle:
            for i in range(self._get_duty(self.angle), self._get_duty(a), -1):
                self.pin.duty_u16(i)
        self.angle = a
        utime.sleep_ms(4)

    def _get_duty(self, angle):
        '''Internal function. Calculates the PWM duty for the given angle.'''
        return round((self.max - self.min) / 180 * angle + self.min)
```

Dieser Code muss unter dem Dateinamen „servo.py“ auf dem Pico gespeichert werden.

# Das komplette Modul "robotlibrary"

Dieses Modul, das von [Github](#) heruntergeladen werden kann, steuert die Roboter mit allen Peripheriegeräten (Motoren, Sensoren). Dazu muss das Paket heruntergeladen und entpackt werden. Das Verzeichnis „robotlibrary“ muss dann auf den Pico hochgeladen werden.

## SMARS Roboter "Theo III"

Um das Modul zu benutzen, muss nur folgender Import gemacht werden: `from robotlibrary.robot import Robot`.

Ein kurzes Codebeispiel, wie der Roboter funktioniert, ist in der Quelldatei zu finden.

Oder hier ein Beispiel für die Benutzung des Servomotors.

```
from robotlibrary.robot import Robot
from time import sleep, sleep_ms
try:
    r = Robot(False)
    r.set_angle(0)
    sleep_ms(500)
    r.set_angle(180)
    sleep_ms(500)
    r.set_angle(90)
    r.set_speed(80)
    while True:
        while r.get_dist() > 15:
            pass
        r.emergency_stop()
        sleep_ms(400)
        r.set_speed_instantly(80)
        r.spin_before_obstacle(20)
        r.set_forward(True)
        r.set_speed(80)
```

```
except:  
    r.emergency_stop()  
    print("Robot stopped")
```

## Crawly

Crawly ist ein Roboter, der, ähnlich wie eine Schildkröte auf vier Beinen kriechen kann. Um Crawly zu steuern, muss nur folgender Import gemacht werden: `from robotlibrary.crawly import Crawly`

Ein kurzes Codebeispiel, wie der Roboter funktioniert, ist in der Quelldatei zu finden.

## Under construction:

## Walky

Walky ist ein Roboter, der, ähnlich wie ein Hund, auf vier Beinen laufen kann. Um Walky zu steuern, muss nur folgender Import gemacht werden: `from robotlibrary.walky import Walky`

Ein kurzes Codebeispiel, wie der Roboter funktioniert, ist in der Quelldatei zu finden.

Die Dokumentation für die Bibliothek ist unter `docs` zu finden oder kann hier heruntergeladen werden: [robotlibrary.pdf](#)

# Dokumentation

## Documentation for robotlibrary/robot.py

### Robot

This is the central class which manages and uses all the other components of the robot. The parameters are defined in config.py

### `_drive`

This abstracted driving function is only called locally by the other functions with better names. It accelerates and decelerates to make driving more natural. Do not call directly!

### `_drive_instantly`

This abstracted driving function is only called locally by the other functions with better names. It sets the speed immediately. Do not call directly!

### `set_speed_instantly`

Sets the new speed immediately. Doesn't change the driving mode of the robot. :param s: the speed you want to set.

### `set_speed`

Sets the new speed and accelerates and decelerates. Doesn't change the driving mode of the robot. :param s: the speed you want to set.

# set\_forward

Sets the direction of the robot. True means forward. :param f: True for forwards and False for backwards.

# spin\_right

Spin right indefinitely.

# spin\_left

Spin left indefinitely.

# turn\_right

This turns the robot to the right without it spinning on the spot. Each call makes the turn steeper.

# turn\_left

This turns the robot to the right without it spinning on the spot. Each call makes the turn steeper.

# go\_left

With Meccanum wheels the robot goes sideways to the left.

# go\_right

With Meccanum wheels the robot goes sideways to the right.

# turn

This turns the robot right or left. Is mostly used by the remote control. :param turn: positive or negative value. Higher values mean steeper turn.

# go\_straight

Lets the robot go straight on. Usually called when a turn shall end.

# spin\_before\_obstacle

This spins until the distance to an obstacle is greater than the given parameter *distance*. :param distance: The distance

# toggle\_spin

Toggle turn for the given duration. With each call the opposite direction(clockwise / anti-clockwise) is used. :param d: The duration for the turn in milliseconds.

# random\_spin

Randomly turn for the given duration. :param d: The duration for the turn in milliseconds.

# stop

Stop the robot slowly by deceleration.

# emergency\_stop

Stop the robot immediately.

# ir\_detected

If implemented this method is called when the IR-sensor has detected a change. Fill in your code accordingly.

# get\_dist

Get the distance from the ultrasonic sensor.

## set\_angle

If implemented, turn the servo motor with the ultrasonic sensor to the given angle. :param a: The angle that is to be set.

## get\_smallest\_distance

This returns the angle of the ultrasonic sensor where it measured the smallest distance

## get\_longest\_distance

This returns the angle of the ultrasonic sensor where it measured the longest distance

# Documentation for robotlibrary/config.py

## Module

This defines the parameters for the motors.

MAX\_DUTY: Set to lower than the maximum not to overload the motors. Absolute maximum is 65535. MIN\_DUTY: Set this to the minimum duty cycle that the motor needs to start moving. MIN\_SPEED: Only 0 is making sense here but if you want you can change that. Must be above 0 though. MAX\_SPEED: If you want another scale than 0-100, set the maximum here.

DEBOUNCE\_WAIT: This defines the waiting time for the debouncing of the buttons. Leave as it is if you don't know what it means.

WHITE\_DETECTED: Use these constants to check for white or black with the IR-sensor. Don't change! BLACK\_DETECTED: Use these constants to check for white or black with the IR-sensor. Don't change!

Motors and ultrasonic sensor must use consecutive pins. so, f. ex. the left motor uses pins 12 and 13. Use >None< if you don't use the device. MLF and LRF are for four wheel drive. ML: pin number

for left motor (or left rear motor for four wheel drive). MR: pin number for right motor (or right rear motor for four wheel drive). MLF: pin number for left motor (or left front motor for four wheel drive). Use None if not used. MRF: pin number for right motor (or right front motor for four wheel drive). Use None if not used. US: pin number for the ultrasonic sensor. Use None if not used. IR: pin number for the infrared sensor. Use None if not used. SERVO: pin number for the servo motor. Use None if not used.

JS\_X\_MEDIAN JS\_Y\_MEDIAN JS\_MAX\_DUTY JS\_MIN\_DUTY: These define the parameters for the joystick. You need to calibrate the numbers. Look at joystick.py for details.

ROBOT\_NAME: You need to set a custom name if you use a remote control.

SERVO\_MIN\_DUTY: Only change if the servo doesn't move the required 180°. SERVO\_MAX\_DUTY: Only change if the servo doesn't move the required 180°.