Codebeispiele

Theo III

Best Practise

Ausschalten der Motoren bei Unterbrechung des Programms

Anfangs wird man sehr viel an dem Roboterprogramm testen müssen. Dabei wird das Programm dann häufig abstürzen. Damit die Motoren nicht weiter in dem Zustand laufen, in dem sie dabei geschaltet waren, kann man mit einem `try/except` arbeiten:

```
try:
  # Hier läuft das Programm
except Exception as err:
    print(err) # Nötig, um Fehlermeldungen angezeigt zu bekommen.
    r.emergency_stop() # Roboter anhalten, hier ein Beispiel mit der robotlibrary.
    print("Robot stopped") # Damit es ganz deutlich ist.
except KeyboardInterrupt:
    r.emergency_stop()
    print("Keyboard interrupt")
```

Effizienter Code

Auch wenn die Rechenleistung der Picos ausreichen sollte, ergibt es Sinn, sich über Effizienz Gedanken zu machen, da schwer zu erkennen ist, ob manche Probleme durch Überlastung des Prozessors hervorgerufen werden.

```
while True:
    robot.drive()
    if us.get_dist() > min_distance:
        # stop or turn
        robot.stop()
```

In diesem Beispiel wird in der Schleife der Befehl `drive()` mit jedem Durchlauf aufgerufen, was nicht sonderlich effizient ist, da die Motoren weiterfahren, auch wenn das Programm gerade andere Befehle ausführt. Eine bessere Variante wäre diese:

```
robot.drive()
while us.get_dist() > min_distance:
    pass
robot.turn()
```

Hier wird nur die Entfernung zum nächsten Hindernis überprüft. Sobald der Roboter zu nahe gekommen ist, wird die Schleife beendet und der Code wird weiter ausgeführt.

Fehlertoleranter Code

Die Sensoren, die wir benutzen, liefern nicht immer zuverlässige und korrekte Ergebnisse. Daher kann man sich nicht darauf verlassen, dass **eine** Messung ausreicht. Ist man auf genauere Ergebnisse angewiesen, kann es sinnvoll sein, die Ergebnisse von Sensormessungen (insbesondere des Ultraschallsensors) zu filtern. Dazu kann gehören, Extremwerte, die im vorliegenden Fall unwahrscheinlich sind, zu ignorieren oder Mittelwerte von mehreren Messungen zu bilden. Ein Beispiel für die Glättung der Entfernungswerte aus dem Ultraschallsensor:

```
us = Ultra(16)
dist_values = deque([0,0,0,0,0],5)
while True:
    d = us.get_dist()
    dist_values.append(d)
    d = sum(dist_values)/len(dist_values)
    print(f"Entfernung: {d} cm")
```

Eine andere Methode muss für das Auslesen des Infrarotsensors gefunden werden. Hier könnte man z.B. eine kurze Wartezeit einbauen und dann abfragen, ob der Sensor immer noch denselben Wert liefert wie bei Auslösung der Reaktion.

Überschreiben von Methoden aus der Bibliothek

Möchte man die Funktion einer Methode aus der Roboterbibliothek (robotlibrary) verändern, kann man die Methode überschreiben (Fachbegriff aus der objektorientierten Programmierung). Dafür wird die Klasse `Robot` vererbt, wie in dem Codebeispiel angegeben. Möchte man nun z.B. die Methode `set_speed()` verändern, dann definiert man sie einfach neu. Ist eine Methode nicht in der abgeleiteten Klasse (in diesem Fall `MyRobot` definiert, dann wird die Methode der Elternklasse (`Robot`) genommen. Probiere es aus, indem du das vorliegende Programm einmal mit und einmal ohne die Definition von `set_speed()` aufrufst.

```
from robotlibrary.robot import Robot
from time import sleep, sleep_ms


class MyRobot(Robot):
```

```
    def __init__(self):
        super().__init__(False)
        print("start")

    def set_speed(self,x):
        print(f"Child method set_speed. Value: {x}")

def main():
    try:
        robot = MyRobot()
        robot.set_speed(90)
        while True:
            sleep(1)
    except KeyboardInterrupt:
        print("The robot was stopped by the user.")
    finally:
        robot.emergency_stop()

if __name__ == "__main__":
    # execute only if run as a script
    main()
```

Den Ultraschallsensor im Hintergrund laufen lassen

In komplexeren Programmen kann es lästig werden, immer wieder die Entfernung zum nächsten Hindernis zu überpüfen. Dieses Beispiel erläutert, wie man das Problem löst, indem man die Entfernungsmessung in den Hintergrund verlegt.

Dieses Beispiel vereint alle vorgestellten Techniken und sollte als Standard-Startdatei genutzt werden.

```
from time import sleep, sleep_ms
import uasyncio as asyncio
from robotlibrary.robot import Robot
################################## Your class definition
class MyRobot(Robot):
    def __init__(self):
        super().__init__(False) # Call the original constructor.
        print("Start MyRobot")
```

```
    # With this method defined here, the robot will not drive as the speed is not set in this
function.
    # This is to illustrate how overwriting works.
    def set_speed(self,x):
        print(f"Child method set_speed. Value: {x}")

################################## End of class definition
# Define functions for your program
async def monitor_dist():
    '''This checks the distance from the ultrasonic sensor continually.
    If the given distance is longer than the measured one, react_to_obstacle() will be called.
    '''
    global distance
    while True:
        if robot.get_dist() < distance:
            react_to_obstacle()
        await asyncio.sleep_ms(100)

def react_to_obstacle():
    '''Do whatever you want to do when an obstacle is detected.
    '''
    global distance
    robot.random_spin(300)
    robot.set_forward(True)
    robot.set_speed(80)

async def driving_program():
    robot.set_speed(90)
    while True:
        print("Driving program running.")
        await asyncio.sleep_ms(3000)

async def main():
    asyncio.create_task(monitor_dist())
    await driving_program()



################################## Initialize the robot and start the program.
robot = MyRobot()
distance = 20 # Define the distance you want to have.
```

```
if __name__ == "__main__":
    # execute only if run as a script
    try:
        asyncio.run(main())
    except KeyboardInterrupt:
        print("The robot was stopped by the user.")
    finally:
        robot.emergency_stop()
```

Beschleunigung mit Entfernungsmessung

Diese Methode ist nur nötig, wenn man nicht asyncio benutzt. Beschleunigt man den Roboter langsam mit der Methode `set_speed`, dann kann er in der Zeit bis zum Erreichen der Geschwindigkeit keine Entfernungsmessung durchführen. Dies ist ein Beispiel, wie man beides erreichen kann:

```
obstacle_detected = False
new_speed = 100
speed_now = 0
min_distance = 15
while speed_now <= new_speed and not obstacle_detected:
    #Set the speed for the motors, f. ex. motor.set_speed(speed_now)
    utime.sleep_ms(10+int(speed_now/2))
    speed_now += 1
    if us.get_dist() < min_distance: # Adjust the code to your needs.
        obstacle_detected = True
if obstacle_detected:
    # Stop or turn or whatever
    obstacle_detected = False
else:
    # keep going
    pass
```

Mit einem Timer arbeiten

Diese Programmiertechnik könnte man z. B. nutzen, um nach einer bestimmten Zeit in das Programm einzuschalten, falls der Roboter zu lange dieselbe Funktion, z. B. vorwärts fahren, ausführt. Dann würde der Zähler zu gegebener Zeit eine Änderung herbeiführen können.

```
from machine import Pin,Timer
import time

def do_something(t):
    print("do something!")

def main():
    timer = Timer()
    timer.init(period=1500, mode=Timer.PERIODIC, callback=do_something)
    while True:
        print("Running...")
        time.sleep_ms(700)

if __name__ == "__main__":
    # execute only if run as a script
    main()
```

Revision #20
Created 17 December 2024 11:25:11 by Marcus Jacobs
Updated 12 November 2025 09:44:26 by Marcus Jacobs