grundlegende_bewegungen

September 25, 2022

1 Grundlegende Bewegungen

In diesem Notebook werden die Grundlagen der Steuerung des Jetbots behandelt.

1.0.1 Importieren der Roboter-Klasse

Um mit der Programmierung des Jetbots zu beginnen muss die Klasse `Robot` importiert werden. Diese Klasse erlaubt es die Motoren des Roboters einfach zu steuern. Sie ist in dem Paket `jetbot` enthalten.

[ ]:
```
from jetbot import Robot
```

Nachdem die Klasse `Robot` importiert wurde kann die Klassen-*Instanz* wie folgt initialisiert werden:

[ ]:
```
robot = Robot()
```

1.0.2 Steuern des Roboters

Nun wo die `Robot`-Instanz erstellt wurde unter dem Namen "robot" kann die Instanz verwendet werden um den Roboter zu steuern. Um den Roboter mit 30% seiner maximalen Geschwindigkeit gegen den Uhrzeigersinn drehen zu lassen kann folgendes Codebeispiel aufgerufen werden:

[ ]:
```
robot.left(speed=0.3)
```

Um den Roboter wieder zu stoppen muss die `stop`-Methode aufgerufen werden.

[ ]:
```
robot.stop()
```

Falls der Roboter nur für eine bestimmte Zeit aktiv sein soll kann das Python-Paket `time` importiert und verwendet werden.

[ ]:
```
import time
```

Dieses Paket definiert die Funktion `sleep`, die die Codeausführung für die angegebene Anzahl von Sekunden blockiert bevor der nächste Befehl ausgeführt wird. Das folgende Codebeispiel läst den Roboter für eine halbe Sekunde nach links drehen.

[ ]:
```
robot.left(0.3)
time.sleep(0.5)
robot.stop()
```

1.0.3 Motoren einzeln ansteuern

Bis jetzt konnten die Motoren nur über die Befehle `left`, `right`, usw. angesteuert werden. Es besteht jedoch auch die Möglichkeit die Geschwindigkeit jedes Motors einzeln zu setzen. Grundsätzlich existieren dafür zwei Möglichkeiten.

Die erste Möglichkeit ist es die Methode `set_motors` aufzurufen. Um zum Beispiel eine Sekunde lang einen Linksbogen zu fahren könnten der linke Motor auf 30% und der rechte Motor auf 60% eingestellt werden:

[ ]:
```
robot.set_motors(0.3, 0.6)
time.sleep(1.0)
robot.stop()
```

Alternativ existiert eine zweite Möglichkeit das gleiche Ergebnis zu erzielen.

Die Klasse `Robot` hat zwei Attribute mit den Namen `left_motor` und `right_motor`, die jeden Motor einzeln repräsentieren. Diese Attribute sind Instanzen der Klasse `Motor`, die jeweils ein `value`-Attribut besitzen. Dieses `value`-Attribut ist ein traitlet das `events` erzeugt, wenn ihm ein neuer Wert zugewiesen wird. In der Motor Klasse wird eine Funktion hinzugefügt, die die Motorbefehle aktualisiert, sobald sich der Wert ändert.

Um also genau das Gleiche wie oben zu erreichen könnte folgender Code ausgeführt werden:

[ ]:
```
robot.left_motor.value = 0.3
robot.right_motor.value = 0.6
time.sleep(1.0)
robot.left_motor.value = 0.0
robot.right_motor.value = 0.0
```

You should see the robot move in the same exact way!

1.0.4 Verbinden der Motoren zu traitlets

Eine hilfreiche Fähigkeit dieser Traitlets ist, dass sie auch mit anderen Traitlets verknüpfen werden können. Das ist sehr praktisch, da Jupyter Notebooks es erlauben grafische `Widgets` zu erstellen, die besagte Traitlets verwenden. Das bedeutet, dass die Motoren mit `Widgets` verknüpft werden können, um sie vom Browser aus zu steuern oder einfach nur Daten zu Visualisieren.

Als Besipiel für diese Fähigkeit dienen zwei Schieberegler, über die die Geschwendigkeit der Motoren individuell eingestellt werden kann.

[ ]:
```
import ipywidgets.widgets as widgets
from IPython.display import display

# create two sliders with range [-1.0, 1.0]
left_slider = widgets.FloatSlider(description='left', min=-1.0, max=1.0, step=0.
 ↪01, orientation='vertical')
right_slider = widgets.FloatSlider(description='right', min=-1.0, max=1.0,␣
 ↪step=0.01, orientation='vertical')
```

```
# create a horizontal box container to place the sliders next to each other
slider_container = widgets.HBox([left_slider, right_slider])

# display the container in this cell's output
display(slider_container)
```

Bis jetzt haben die Schieberegler noch keinen Effekt auf den Jetbot. Diese müssen zunächst mit den Motoren verbunden werden. Dies geschieht mit der Funktion `link` aus dem `traitlets`-Paket.

[ ]:
```
import traitlets

left_link = traitlets.link((left_slider, 'value'), (robot.left_motor, 'value'))
right_link = traitlets.link((right_slider, 'value'), (robot.right_motor,␣
 ↪'value'))
```

Werden die Regler nun vorsichtig bewegt ist zu erkennen, dass der zugehörige Motor sich entsprechend der Position des Reglers dreht.

Die erstellte `link`-Funktion ist eine bidirektionale Verbindung. Das bedeutet, dass wenn die Motorwerte an einer anderen Stelle gesetzt werden, sich die Schieberegler entsprechend auch anpassen. Dafür kann folgendes Codebeispiel ausgeführt werden:

[ ]:
```
robot.forward(0.3)
time.sleep(1.0)
robot.stop()
```

Es ist zu erkennen, dass sich die Regler ohne Einwirken des Nutzers bewegen. Die Verbindung kann über die `unlink`-Methode wieder entfernt werden.

[ ]:
```
left_link.unlink()
right_link.unlink()
```

Sollte keine *bidirektionale* Verbindung erwünscht sein, um z.B. die Motorwerte nur zu visualisieren, kann die `dlink`-Funktion genutzt werden. Die linke Eingabe ist die `Quelle` und der rechte Eingang das `Target`.

[ ]:
```
left_link = traitlets.dlink((robot.left_motor, 'value'), (left_slider, 'value'))
right_link = traitlets.dlink((robot.right_motor, 'value'), (right_slider,␣
 ↪'value'))
```

Werden nun die Schieberegler bewegt, ändern sich die Werte der Motoren nicht. Werden die Werte der Motoren jedoch an einer anderen Stelle angepasst, bewegen sich die Schieberegler entsprechend.

1.0.5 Funktionen an Ereignisse binden

Ein anderer Weg traitlets zu benutzen ist es Funktionen (wie `forward`) an Ereignisse zu binden. Diese Funktionen werden aufgerufen, sobald sich der Wert des Objekts ändert und erhalten Informationen über die Änderung wie den `old`- und `new`-Wert.

Beispielhaft werden einige Buttons erstellt, die den Roboter steuern sollen.

[ ]:
```
# create buttons
button_layout = widgets.Layout(width='100px', height='80px',␣
 ↪align_self='center')
stop_button = widgets.Button(description='stop', button_style='danger',␣
 ↪layout=button_layout)
forward_button = widgets.Button(description='forward', layout=button_layout)
backward_button = widgets.Button(description='backward', layout=button_layout)
left_button = widgets.Button(description='left', layout=button_layout)
right_button = widgets.Button(description='right', layout=button_layout)

# display buttons
middle_box = widgets.HBox([left_button, stop_button, right_button],␣
 ↪layout=widgets.Layout(align_self='center'))
controls_box = widgets.VBox([forward_button, middle_box, backward_button])
display(controls_box)
```

Auch hier müssen die Motoren zunächst verbunden werden. Dazu werden Funktionen erstellt, die mit den `on_click`-Ereignissen der Buttons verknüpft werden.

[ ]:
```
def stop(change):
    robot.stop()

def step_forward(change):
    robot.forward(0.4)
    time.sleep(0.5)
    robot.stop()

def step_backward(change):
    robot.backward(0.4)
    time.sleep(0.5)
    robot.stop()

def step_left(change):
    robot.left(0.3)
    time.sleep(0.5)
    robot.stop()

def step_right(change):
    robot.right(0.3)
    time.sleep(0.5)
    robot.stop()
```

Nachdem die Funktionen definiert wurden, können diese nun mit den `on_click`-Ereignissen der Buttons verknüpft werden.

[ ]:
```
# link buttons to actions
stop_button.on_click(stop)
```

```
forward_button.on_click(step_forward)
backward_button.on_click(step_backward)
left_button.on_click(step_left)
right_button.on_click(step_right)
```

Nun kann der Jetbot per Button-Klick gesteuert werden!

1.0.6 Heartbeat Killswitch

Das letzte Beispiel zeigt, wie ein 'Heartbeat' umgesetzt werden kann, um den Roboter zu stoppen. Dabei handelt es sich um einen leichten Weg um zu prüfen, ob die Verbindung zum Roboter noch aktiv ist. Über den Schieberegler kann die Zykluszeit des Heartbeats (in Sekunden) angepasst werden. Kann keine Kommunikation zwischen Jetbot und Computer innerhalb zwei Heartbeats abgeschlossen werden, wird das '`status`'-Attribut des Heartbeats auf `dead` gesetzt. In dem Moment wo die Verbindung wieder hergestellt wird, wechselt des `status`-Attribut wieder zu `alive`.

[ ]:
```
from jetbot import Heartbeat

heartbeat = Heartbeat()

# this function will be called when heartbeat 'alive' status changes
def handle_heartbeat_status(change):
    if change['new'] == Heartbeat.Status.dead:
        robot.stop()

heartbeat.observe(handle_heartbeat_status, names='status')

period_slider = widgets.FloatSlider(description='period', min=0.001, max=0.5,␣
↪step=0.01, value=0.5)
traitlets.dlink((period_slider, 'value'), (heartbeat, 'period'))

display(period_slider, heartbeat.pulseout)
```

Beispiel haft kann der unten stehende Codeschnipsel ausgeführt werden, währen der Schieberegler nach unten geschoben wird.

[ ]:
```
robot.left(0.2)

# now lower the `period` slider above until the network heartbeat can't be␣
↪satisfied
```