datensammlung_spurverfolgung

September 25, 2022

1 Spurverfolgung - Datensammlung

Auch hier werden wieder die folgenden Schritte durchgeführt:

1. Datensammlung
2. Training
3. Live-Einsatz

Im Unterschied zu der vorherigen Fähigkeit wird hier nicht mehr die Klassifizierung eingesetzt, sondern die **Regression**. Diese wird verwendet um dem Jetbot zu ermöglichen eine Spur zu verfolgen (bzw. einen beliebigen Pfad oder Zielpunkt).

1. Sammeln von Daten für verschiedene Positionen des Jetbots auf dem Pfad (verschiedene Abstände zum Mittelpunkt, verschiedene Winkel, etc.)

   Wie auch schon bei der Kollisionsvermeidung gilt: Datenvarianz ist entscheident!

2. Live-Bild der Kamera anzeigen
3. 'Grünen Punkt', welcher der Zielrichtung des Roboters entspricht, auf dem Bild platzieren
4. X und Y Werte des grünen Punktes zusammen mit dem Bild der Kamera des Roboters speichern

Dannach wird im Trainings-Notebook ein neuronales Netztwerk trainiert, welches die X und Y Werte des Labels vorhersagen kann. Im Live-Demo-Notebook werden die X und Y Werte verwendet um einen ungefähren Steuerwert zu berechnen.

Wie wird der Zielpunkt richtig platziert?

1. Auf das Livebild der Kamera gucken
2. Überlegen welchen Pfad der Roboter auf dem Bild fahren sollte (Entfernung abschätzen, nach der der Roboter von der Straße abkommen würde)
3. Zielpunkt so weit wie möglich in die Ferne platzieren, so dass der Roboter möglichst lange geradeaus fahren kann, bevor er von der Straße abkommt.

   Beispiel: Bleibt die Strecke lange gerade, kann der Zielpunkt weit in die Ferne gesetzt werden. Liegt eine scharfe Kurve bevor, muss der Zielpunkt nah am Roboter platziert werden, da er sonst nicht mehr die Spur halten würde.

Tut das nueronale Netzt das was ihm angelernt wurde, versichern die Label-Richtlinien folgendes:

1. Der Roboter kann sicher in Richting eines Ziels fahren (ohne dabei von der Straße abzukommen)
2. Der Zielpunkt bewegt sich immer weiter entlang der zu fahrenden Strecke

Man kann sich das Ergebnis so ähnlich vorstellen wie das Reiten eines Esels mit einer Karotte an einer Angel. Der wesentliche Unterschied ist, dass das neuronale Netzt vorgibt, wo sich die Karotte befindet.

1.0.1 Einbinden der Bibliotheken

Von entscheidender Rolle ist hier die Bibliothek OpenCV mit der die Kamerabilder sowohl dargestellt als auch mit Labeln angespeichert werden können. Bibliotheken wie uuid und datetime werden für die Namensgebung der Bilddateien verwendet.

[ ]:
```
# IPython Libraries for display and widgets
import ipywidgets
import traitlets
import ipywidgets.widgets as widgets
from IPython.display import display

# Camera and Motor Interface for JetBot
from jetbot import Robot, Camera, bgr8_to_jpeg

# Basic Python packages for image annotation
from uuid import uuid1
import os
import json
import glob
import datetime
import numpy as np
import cv2
import time
```

1.0.2 Datensammlung

Erneut soll das Kamerabild wieder angezeigt werden. Diesmal wird jedoch ein spezielles ipywidget verwendet (`jupyter_clickable_image_widget`), welches es ermöglicht auf das Bild zu klicken und die Koordinaten des Labels zu speichern.

Es wird die Kamera Klasse des Jetbot verwendet, um die CSI MIPI Kamera zu aktivieren. Das neuronale Netzwerk nimmt ein Bild mit den Maßen 224x224 Pixel als Eingabe entgegen. Die Kamera wird auf diese Größe gesetzt, um die Dateigröße des Datensatzes zu minimieren. In einigen Szenarien kann es besser sein, Daten in einer größeren Bildgröße zu sammeln und später auf die gewünschte Größe zu skalieren.

Der folgende Codeblock zeigt das Livebild der Kamera auf welches geklickt werden kann, um ein Label zu platzieren. Daneben wird das letzte Bild angezeigt, auf welchem ein grüner Kreis zu sehen ist, wo zuvor geklickt wurde. Darunter wird die Anzahl der gespeicherten Bilder angezeigt.

Wird links auf das Livebild geklickt, dann wird ein Bild inklusive Label im `dataset_xy` abgespeichert mit dem Namen

`xy_<x value>_<y value>_<uuid>.jpg`

Beim TRainieren werden die Bilder geladen und die X und Y Werte aus dem Dateinamen extrahiert.

Dabei sind `<x value>` und `<y value>` die Koordinaten **in Pixeln** (gezählt von der linken oberen Ecke).

[ ]:
```
from jupyter_clickable_image_widget import ClickableImageWidget

DATASET_DIR = 'dataset_xy'

# we have this "try/except" statement because these next functions can throw an␣
 ↪error if the directories exist already
try:
    os.makedirs(DATASET_DIR)
except FileExistsError:
    print('Directories not created because they already exist')

camera = Camera()

# create image preview
camera_widget = ClickableImageWidget(width=camera.width, height=camera.height)
snapshot_widget = ipywidgets.Image(width=camera.width, height=camera.height)
traitlets.dlink((camera, 'value'), (camera_widget, 'value'),␣
 ↪transform=bgr8_to_jpeg)

# create widgets
count_widget = ipywidgets.IntText(description='count')
# manually update counts at initialization
count_widget.value = len(glob.glob(os.path.join(DATASET_DIR, '*.jpg')))

def save_snapshot(_, content, msg):
    if content['event'] == 'click':
        data = content['eventData']
        x = data['offsetX']
        y = data['offsetY']

        # save to disk
        #dataset.save_entry(category_widget.value, camera.value, x, y)
        uuid = 'xy_%03d_%03d_%s' % (x, y, uuid1())
        image_path = os.path.join(DATASET_DIR, uuid + '.jpg')
        with open(image_path, 'wb') as f:
            f.write(camera_widget.value)

        # display saved snapshot
        snapshot = camera.value.copy()
        snapshot = cv2.circle(snapshot, (x, y), 8, (0, 255, 0), 3)
        snapshot_widget.value = bgr8_to_jpeg(snapshot)
        count_widget.value = len(glob.glob(os.path.join(DATASET_DIR, '*.jpg')))

camera_widget.on_msg(save_snapshot)
```

```
data_collection_widget = ipywidgets.VBox([
    ipywidgets.HBox([camera_widget, snapshot_widget]),
    count_widget
])

display(data_collection_widget)
```

Wie bereits in anderen Notebooks muss am Ende wieder die Kamera freigegeben werden (Verbindung trennen).

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

1.0.3 Next

Auch hier gilt wieder: Wurden genügend Daten gesammelt, so können diese wieder zum Trainieren auf einen Leistungsstarken Computer übertragen werden.

> Sollte der Wunsch bestehen das Trainieren auf dem Jetbot selbst durchzuführen, so kann dieser Schritt übersprungen werden. Es sollte sich jedoch auf eine lange Berechnungszeit eingestellt werden.

Der folgende Codeblock komprimiert die Bilddaten wieder in eine zip Datei, welche dann auf einen Leistungsstarken Computer übertragen werden kann.

[ ]:
```
def timestr():
    return str(datetime.datetime.now().strftime('%Y-%m-%d_%H-%M-%S'))

!zip -r -q road_following_{DATASET_DIR}_{timestr()}.zip {DATASET_DIR}
```

Es sollte nun eine Datei angelegt worden sein mit dem Namen road_following_<Date&Time>.zip. Diese kann heruntergeladen werden.