Projektdokumentation

Marius Trifan
Frederic Czudnochski

Inhaltsverzeichnis

1 Unsere Mission

Das Ziel unserer ursprünglichen Mission war es, einen roten Gegenstand autonom von der Drohne verfolgen zu lassen.

Dafür hatten wir bereits einiges vorbereitet. Zum einen das Auslesen der Kamera, als auch einen Algorithmus zum Erkennen des Objektes. Das Problem, woran wir gescheitert sind, war, dass die Akkus der Drohne sehr schnell entladen wurden, wenn die Kamera eingeschaltet ist. Dadurch kam es dazu, dass wir lange Wartezeiten hatten und nicht weiter aktiv testen konnten.

Daher haben wir unser Projekt Anfang Dezember geändert. Unser neues Ziel war es, dass die Drohne auf dem Missionpad 1 startet und dann so lange geradeaus fliegt, bis sie Missionpad 2 findet, sich da dann neu ausrichtet und dann immer von Missionpad zu Missionpad fliegt, bis sie wieder bei Missionpad 1 angekommen ist.

Später haben wir dann beschlossen, dass wir zusätzlich die Fläche des umflogenen Vielecks berechnen wollten, um die Mission etwas komplexer zu gestalten.

2 Probleme und Schwierigkeiten

2.1 Probleme vor dem Start des Projekts

Da wir ursprünglich geplant hatten, dass wir, wie oben genannt, eine andere Mission fliegen, hatten wir nicht sehr viel Zeit, um unser neues Projekt umzusetzen.

Zusätzlich haben auch bei diesem Projekt die Akkus der Drohne nicht immer ausreichend lange gehalten um genug zu testen.

2.2 Ungenaue Zentrierung

Die größte Schwierigkeit im Programm war:

```
drone.go_xyz_speed_yaw_mid(0,0,20,20,0,pad,pad)
```

Diese Zeile sollte die Drohne neu über dem Missionpad ausrichten, indem sie der Drohne sagt, dass sie von Missionpad *pad* zu den Koordinaten 0,0,20 fliegen soll mit einer Geschwindigkeit von 20cm/s. Zusätzlich sorgt die letzte 0 dafür, dass sich die Drohne anpasst an die Ausrichtung des Missionpads *pad*.

Dies funktionierte bei den echten Drohnen sehr selten, bzw. fast nie, da diese bei dem Befehl eine Fehlermeldung zurückgegeben haben, dass sie diesen Befehl nicht kennen würden. In einzelnen Fällen hat es jedoch funktioniert, wodurch wir diesen Befehl weiterhin genutzt haben, da eine manuelle Ausrichtung sehr ungenau und kompliziert wäre.

2.3 Koordinatensystem und Edge-Cases

Da die Positionsdaten der Drohne immer relativ zum aktuell erkannten Pad sind, mussten beim Wechsel zu einem neuen Pad der letzte Vektor aufwendig korrigiert werden. Dabei traten sogenannte Edge-Cases (seltene, unerwartete Szenarien) aus, wie zum Beispiel:

- Ungenaue Distanzwerte wodurch alle restlichen ungültig wurden
- Nicht exakt überflogene Pads, wodurch es nicht registriert wurde
- Gleiches Pad wird 2mal erkannten

Diese Probleme brauchten eine exklusive Fehlerverarbeitung im Programm

3 Ablauf des Programms

1. Das Programm startet und stellt eine Verbindung zur DJI Tello EDU Drohne her.
2. Der aktuelle Batteriestand der Drohne wird abgefragt und ausgegeben.
3. Die Drohne hebt automatisch ab und geht in den Schwebeflug über.
4. Die Mission-Pad-Erkennung wird aktiviert und die Erkennungsrichtung festgelegt.
5. Das Programm wartet, bis das erste Mission Pad erkannt wird. Dieses Pad dient als Startpunkt der Mission.
6. Sobald das erste Pad erkannt wurde, richtet sich die Drohne über diesem Pad aus.
7. Die Drohne beginnt, sich schrittweise vorwärts zu bewegen, um dem durch die Mission Pads vorgegebenen Kurs zu folgen.
8. Während des Fluges wird kontinuierlich überprüft, ob ein Mission Pad erkannt wird oder nicht.
9. Wird kein Pad erkannt, werden die aktuellen Abstände zum zuletzt bekannten Pad gespeichert und aktualisiert.
10. Wird ein neues Mission Pad erkannt, wird dieses als Ecke des Vielecks interpretiert.
11. Der zurückgelegte Weg seit dem letzten Pad wird als Seitenvektor berechnet und gespeichert.
12. Die Drohne richtet sich erneut über dem neu erkannten Mission Pad aus, um Messfehler zu reduzieren.
13. Dieser Vorgang wiederholt sich für alle weiteren Mission Pads des Vielecks.

14. Sobald Mission Pad 1 erneut erkannt wird, erkennt das Programm, dass das Vieleck vollständig abgeflogen wurde.
15. Die Drohne richtet sich ein letztes Mal über Mission Pad 1 aus.
16. Die Drohne landet automatisch.
17. Nach der Landung werden alle gespeicherten Seitenvektoren ausgegeben.
18. Aus den Seitenvektoren werden die Eckpunkte des Vielecks rekonstruiert.
19. Mithilfe der Gaußschen Trapezformel wird die Fläche des Vielecks berechnet und ausgegeben.
20. Abschließend wird die Mission-Pad-Erkennung deaktiviert und die Verbindung zur Drohne beendet.

4 Quellcode

```
from typing import Tuple
from djitellopy import tello
import time

drone = tello.Tello()
    # host="127.0.0.1" für Simulation


def center_over_pad(pad):
    drone.go_xyz_speed_yaw_mid(0,0,20,20,0,pad,pad)


def get_pad_position():
    x = drone.get_mission_pad_distance_x()
    y = drone.get_mission_pad_distance_y()
    return x, y




def polygon_area_from_vectors(vectors):
    # Reconstruct vertices
    x, y = 0, 0
    vertices = [(x, y)] # die Koordinaten der Ecken des
     Vieleck
```

```
    # Berechne die Ecken
    for dx, dy in vectors:
        x += dx
        y += dy
        vertices.append((x, y))
    print("Vertices:",vertices)

    # Shoelace formula aka. Gaußsche Trapezformel
    # NOTE: Wir verstehen diese Formel nicht, siehe dazu
https://de.wikipedia.org/wiki/Gau%C3%9Fsche_Trapezformel
    area = 0
    n = len(vertices)
    for i in range(n - 1):
        x1, y1 = vertices[i]
        x2, y2 = vertices[i + 1]
        area += x1 * y2 - x2 * y1

    return abs(area) / 2


# --------------------------
#     HAUPTPROGRAMM
# --------------------------

drone.connect()

print("Batterie:", drone.get_battery(), "%")



# Start
drone.takeoff()
time.sleep(1)

# Mission Pads aktivieren
drone.enable_mission_pads()
drone.set_mission_pad_detection_direction(0)  # 2 = beide

print("Warten auf Mission Pad 1 ...")
pad = drone.get_mission_pad_id()
print(f"Pad {pad} erkannt!")


center_over_pad(pad)
```

```
last_pad = pad

step_2d = (0,0) # speichert die Steigung der Seite
dist_pad = get_pad_position() # Abstand zum Anfang der Seite
print("Starte Vorwärts-Navigation...\n")

side_vectors=[] #  Vektoren, die zu den jeweiligen Seite von
der letzte Seite führen
# Endlosschleife bis Pad 1 erneut kommt
while True:
    # Schrittweise vorwärts fliegen
    drone.move_forward(50)
    time.sleep(0.5)

    # Aktuelles Pad abfragen
    pad = drone.get_mission_pad_id()

    # Aktueller Abstand zum Pad
    dist_current_pad = get_pad_position()
    print(dist_current_pad)

    # Hier entsprechen Pads den Ecken des Vieleck

    # Nachdem ein neuer Pad gefunden wurde,
    # speichere die Steigung der neuer Seite
    if dist_pad == (0,0):
        print("Reseting the step_2d")
        step_2d = dist_current_pad

    # Wenn kein Pad gefunden wurde,
    # aktualisiere den Abstand zum Pad
    if pad == -1:
        dist_pad=dist_current_pad

    elif pad != -1:   # Ein Pad wurde erkannt
        print(f"Mission Pad erkannt: {pad}")

        # Man konnte warscheinlich die If-Bedingung weglassen,
        # aber wir sind uns nicht sicher
        # Es könnte ,,Edge-Cases" geben
        if pad != last_pad: # Eine neue Ecke wurde gefunden,
damit auch eine neue Seite
            # Da der Abstand nun relativ zum neuen Pad ist,
            # hat man nicht den aktuellsten Abstand zum
letzten Pad
```

```
            # und muss berechnet werden.
            # + step_2d macht genau das.
            # - dist_current_pad[0] falls die Drohne nicht
genau auf dem neuen Pad steht
            dist_pad = dist_pad[0] -
dist_current_pad[0]+step_2d[0], dist_pad[1] -
dist_current_pad[1]+step_2d[1]
            print("Side done with len:",dist_pad)
            # speichere den neun Vektor
            side_vectors.append(dist_pad)
            dist_pad = (0,0)


        # Wenn wieder Pad 1 → landen
        if pad == 1 and last_pad != 1:
            print("Pad 1 erneut erreicht → Landen...")
            center_over_pad(pad)
            drone.land()
            break

        # Über dem Pad ausrichten
        center_over_pad(pad)

        last_pad = pad

print("\nGot the Following side Vectors:")
print("\t",side_vectors)

print("\n Calculated Area is:
\t",polygon_area_from_vectors(side_vectors))
# Mission Pads deaktivieren (optional)
drone.disable_mission_pads()
drone.end()
```

5 Version 2

Das bislang beschriebene Programm funktioniert in der Theorie perfekt, allerdings hat sich herausgestellt, dass in der neusten Version der Firmware der "jump" Befehl nicht mehr verfügbar ist.

Somit mussten wir uns in der letzten Stunde vor Abgabe eine alternative überlegen. Diese alternative war, dass die Drohne statt vorwärts, in x-Richtung fliegt. Dafurch erreichen wir das gleiche, das die Drohne immer in Richtung des nächsten Pads fliegt.

Das einzige Problem, das noch verbleibt, ist, dass wir nicht die Winkel wissen, in denen die Pads zueinander liegen. Und daher können wir noch nicht die Fläche ausrechnen.

Wir konnten bislang noch nicht ausprobieren, ob der Befehl get_yaw() die Rotation relativ zum aktuellen Missionpad angibt oder nicht. Falls doch, ist auch die Flächenmessung wieder möglich. Aber das konnte bislang noch nicht getestet werden.

6 Quellcode Version 2

```
from djitellopy import tello
import time
import os

drone = tello.Tello()


def center_over_pad(pad):
#    drone.go_xyz_speed_mid(130,0,20,50,0,pad,pad+1)
    drone.go_xyz_speed_mid(0,0,30,20,pad)


def get_pad_position():
    x = drone.get_mission_pad_distance_x()
    y = drone.get_mission_pad_distance_y()
    return x, y


def polygon_area_from_vectors(vectors):
    # Reconstruct vertices
    x, y = 0, 0
    vertices = [(x, y)] # die Koordinaten der Ecken des
     Vieleck
```

```
    # Berechne die Ecken
    for dx, dy in vectors:
        x += dx
        y += dy
        vertices.append((x, y))
    print("Vertices:",vertices)

    # Shoelace formula aka. Gaußsche Trapezformel
    # NOTE: Ich verstehe diese Formel nicht, siehe dazu
https://de.wikipedia.org/wiki/Gau%C3%9Fsche_Trapezformel
    area = 0
    n = len(vertices)
    for i in range(n - 1):
        x1, y1 = vertices[i]
        x2, y2 = vertices[i + 1]
        area += x1 * y2 - x2 * y1

    return abs(area) / 2


# --------------------------
#     HAUPTPROGRAMM
# --------------------------

drone.connect()

print("Batterie:", drone.get_battery(), "%")



# Start
drone.takeoff()
time.sleep(1)

# Mission Pads aktivieren
drone.enable_mission_pads()
drone.set_mission_pad_detection_direction(2)  # 2 = beide

drone.get_mission_pad_id()
drone.get_mission_pad_id()

print("Warten auf Mission Pad 1 ...")
pad = drone.get_mission_pad_id()
start_pad = pad
```

```
if start_pad <=0 :
    print(f"EXIT due to mid = {start_pad}")
    drone.land()
    os._exit(1)
#visited_pads = [pad]
print(f"Pad {pad} erkannt!")


center_over_pad(pad)

last_pad = pad

step_2d = (0,0) # speichert die Steigung der Seite
dist_pad = get_pad_position() # Abstand zum Anfang der Seite
print("Starte Vorwärts-Navigation...\n")

side_vectors=[] #  Vektoren, die zu den jeweiligen Seite von
der letzte Seite führen
# Endlosschleife bis Pad 1 erneut kommt
dist_flown = 50
while True:
    # Schrittweise vorwärts fliegen
    drone.go_xyz_speed_mid(dist_flown,0,50,30,last_pad)
    dist_flown += 50
    time.sleep(0.5)

    # Aktuelles Pad abfragen
    pad = drone.get_mission_pad_id()

    # Aktueller Abstand zum Pad
    dist_current_pad = get_pad_position()
    print(f"Entfernung zum Pad {pad} ist: {dist_current_pad}")
     entsprechende Pads den Ecken des Vieleck

    # Nachdem ein neuer Pad gefunden wurde,
    # speichere die Steigung der neuer Seite

    # Wenn kein Pad gefunden wurde,
    # aktualisiere den Abstand zum Pad

    if pad != -1:   # Ein Pad wurde erkannt
        print(f"Mission Pad erkannt: {pad}")

        # Man konnte warscheinlich die If-Bedingung weglassen,
```

```
        # aber ich bin mir nicht sicher
        # Es konnte ,,Edge-Cases" geben
        if pad != last_pad: # Eine neue Ecke wurde gefunden,
            damit auch eine neue Seite
            # Da der Abstand nun relativ zum neuen Pad ist,
            # hat man nicht den aktuelsten Abstand zum letzten
            Pad
            # und muss berechnet werden.
            # + step_2d macht genau das.
            # - dist_current_pad[0] falls die Drohne nicht
            genau auf dem neuen Pad steht
            dist_flown -= drone.get_mission_pad_distance_x()
            print(f"debug:
            {drone.get_mission_pad_distance_x()}")
            print("Side done with len:",dist_flown)

            # speichere den neun Vektor
            side_vectors.append(dist_pad)
            dist_pad = (0,0)
            dist_flown = 0
            center_over_pad(pad)


        # Wenn wieder Pad 1 → landen
        if pad == start_pad and last_pad != start_pad:
            print(f"Pad {start_pad} erneut erreicht →
            Landen...")
            center_over_pad(pad)
            drone.land()
            break

        # Über dem Pad ausrichten

        last_pad = pad

print("\nGot the Following side Vectors:")
print("\t",side_vectors)

print("\n Calculated Area is:\t",
polygon_area_from_vectors(side_vectors))
# Mission Pads deaktivieren (optional)
drone.disable_mission_pads()
drone.end()
```