Uporaba knjižnice DJITelloPy

V prvem koraku namestite Python verzijo > 3.6 in ustvarite virtualno okolje (venv) z ukazom:

```
cd \pot_do_mape_kjer_zelite_kreairati_venv\...\
python -m venv droneVenv
```

Nato aktivirajte venv z ukazom:

`\pot_do_venv\...\droneVenv\Scripts\Activate.ps1` (za okolje Windows PowerShell) ali

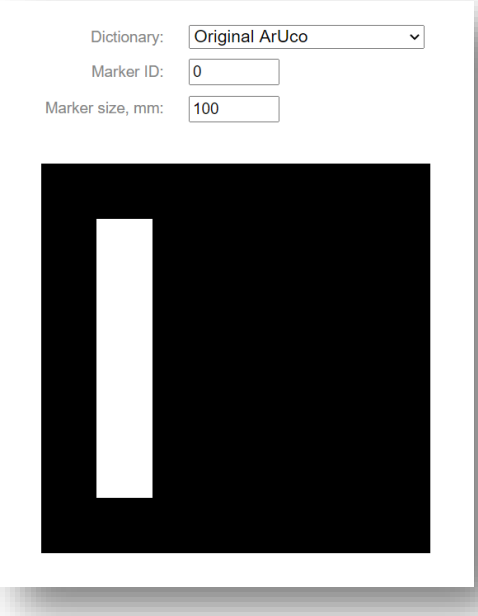
`source ./droneVenv/bin/activate` (za okolje Linux)

Okolje venv aktivirate tudi v urejevalniku kode VS Code (spodaj desno).

Za samo namestitev knjižnice sledite navodilom na spletni strani [1] oz. uporabite ukaz:

```
cd \pot_do_ DJITelloPy\…\DJITelloPy
pip install -e .
```

Dopolnjena knjižnica že vključuje funkcijo *detectAruco(arucoID)* za prepoznavo ArUco značk, ki bodo nameščene pod obroči. Značke (»Original ArUco«) lahko generirate na spletni strani [2], pri čemer je potrebno izbrati ustrezen ID značke (od 0 do 5) in velikost 100 mm (slika 1).



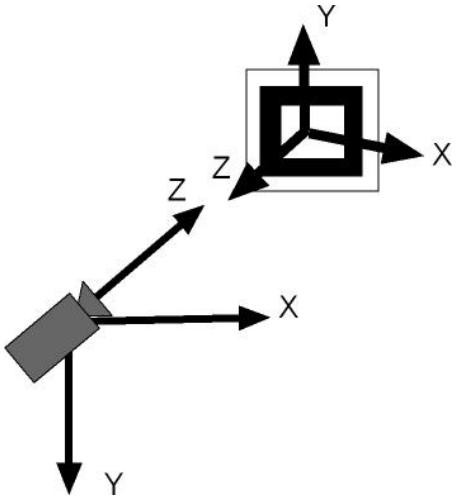
Slika 1: Generiranje ArUco značk

Da bi funkcija za prepoznavo ArUco značk pravilno delovala, je potrebno predhodno kalibrirati kamero drona in določiti intrinzične parametre. Znotraj mape DJITelloPy se že nahaja datoteka »calib.txt«, ki vključuje te parametre. Ker ti parametri niso povsem identični za vse Tello drone oz. vgrajene kamere, je priporočeno, da te parametre ponovno izračunate. Za ta namen povežite prenosni računalnik na dostopno točko (WiFi) drona in zaženite program, preko katerega lahko zajemate slike z drona (ločljivost slik naj bo 960x720). Nato natisnjeno šahovnico položite na ravno podlago ter ročno premikajte dron tako, da boste pod različnimi koti in na različnih razdaljah zajeli slike šahovnice (zajemite vsaj 20 slik). Slike nato kopirajte v mapo Calibration/data ter zaženite program »Calibration.py«. Če je bila kalibracija uspešna, potem program generira datoteko »calib.txt« v kateri so zapisani intrinzični parametri in parametri popačenja. Datoteko »calib.txt« nato kopirajte v mapo DJITelloPy, kjer se nahaja datoteka main.py.

Funkcija *detectAruco*, ki se nahaja v »Tello_Control.py« vrne rotacijski Rvec vektor in translacijski Tvec vektor, ki predstavljata transformacijo iz koordinatnega sistema (KS ArUco značke v koordinatni sistem kamere (glej sliko 2). Ta dva vektorja sta nato uporabljena v funkciji:

T1, T2, yaw = self.transformArucoToDroneCoordinates(self.Rvec, self.Tvec), ki izbrani 3D točki »specific_point_aruco_1« in »specific_point_aruco_2«, definirani v KS ArUco značke, preslika v KS drona (glej sliko 3)

Pri uporabi funkcije »go_xyz_speed (x,y,z,speed)« velja koordinatni sistem drona (slika 3), kjer x+ pomeni premik naprej, y+ premik v levo in z+ premik navzgor v cm. Pri uporabi funkcij go_xyz_speed (x,y,z,speed), move_right(dist), move_up(dist) itd. morajo biti premiki večji od 20 cm sicer se dron ne premakne. Maksimalna hitrost je 100 cm/s. Ko posamezna funkcija vrne True, vemo, da se je ukaz pravilno izvedel.



Slika 2: Koordinatni sistem kamere in ArUco značke

Slika 3: Koordinatni sistem drona

Predlagamo, da celoten algoritem vodenja implementirate znotraj funkcije controlAll(T1, T2, yaw), ki se izvaja na svoji niti (Thread-u). V priloženi funkciji imate že implementiran osnovni primer letenja skozi obroč, kjer se dron najprej poravna z obročem oz. ArUco značko na razdalji točke T1 + pomik določen v funkciji self.distC(T1,yaw,**0.5**), kar je skupaj 1,5 m stran od obroča. Namreč, potrebno je paziti, da pri uporabi vodenja na osnovi slike (relativni položaj drona glede na ArUco značko: self.arucoId = 0,1,...,5) dron ne pride preblizu obroča, saj lahko ArUco značka pade izven vidnega polja.

V podanem primeru algoritem vodenja znotraj controlAll nato preveri, da je položaj drona dovolj natančno določen glede na obroč (uporaba nizko pasovnega filtra), da lahko nadaljnje premike izvajamo samo na osnovi optičnega toka (preko kamere na dnu drona) oz. inercialnega navigacijskega sistema drona (IMU) npr. z uporabo funkcije »curve_xyz_speed«. Začetno poravnavo lahko implementirate tudi preko funkcije »send_rc_control« (namesto move_right, move_back itd.), tako da implementirate PID/MPC regulator za osi x,y,z ter zasuk yaw.

Opis posameznih funkcij najdete v dokumentu »tello.py«. Za pisanje in urejanje kode predlagamo Visual Studio Code.

Povezave:

[1] DJITelloPy: https://github.com/damiafuentes/DJITelloPy

[2] ArUco: <http://chev.me/arucogen/>

[3] ArUco detection: <https://github.com/opencv/opencv_contrib/tree/master/modules/aruco>