Načrtovanje algoritma brezpilotnega letalnika za premagovanje poligona ovir

Domen Višnar, Matevž Stopar, Žan Hedžet Kostajnšek, Matej Možek

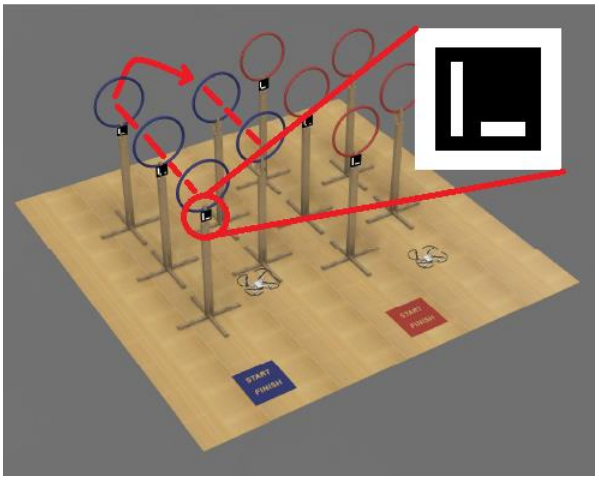
Univerza v Ljubljani, Fakulteta za elektrotehniko
E-pošta: domen2.visnar@gmail.com

Design of a drone algorithm for navigating the obstacle course

Abstract. *This paper presents development process and methodology employed in design of an algorithm for an unmanned aerial vehicle (drone), capable of successfully navigating an obstacle course. Primary emphasis of the algorithm revolves around leveraging the drone video feed, obtained from its on-board camera, to accurately determine drone orientation and approximate its position in relation to ArUco markers, which are strategically placed on the obstacles. We focused on fuzzy logic controller, switch logic controller and discrete movements controller. An extensive research was conducted, exploring the three aforementioned distinct approaches to solve the problem at hand. Switch logic controller has shown most promising results and was implemented as a final algorithm.*

1 Uvod

Delo se je osredotočalo na izdelavo algoritma za vodenje brezpilotnega letalnika DJI Ryze Tello [1], ki mora preleteti poligon z ovirami, katere identificira in orientira relativno na sebe s pomočjo lastne kamere. Poligon, ki ga je potrebno preleteti, je viden na sliki 1 in je sestavljen iz petih obročev polmera 30 cm, vsak je označen s svojo številko zapisano v ArUco [2] znački. Značka je postavljena na znani poziciji od središča obroča.
Cilj premagovanja ovir je preleteti čim več obročev v pravilnem zaporedju, opraviti obrate na določenih mestih in pravilno pristati na zato označenem mestu. V primeru izenačenja vseh kriterijev šteje hitrost opravljene naloge.



Slika 1:Poligon premagovanja ovir.

Slika 2 prikazuje nadzor in vodenje letalnika, ki se izvaja na osebnem računalniku na katerega je letalnik povezan z Wi-Fi povezavo. Računalnik iz letalnika prejema telemetrijo (hitrost, pospešek, nivo baterije…) in posnetek kamere. Prejete podatke in video računalnik obdela in glede na podlagi rezultatov pošlje letalniku ustrezne ukaze premika.

Slika 2: Shema sistema.

Za obdelavo video posnetkov smo uporabili knjižnico OpenCV [3]. Za ukaze in povezavo z letalnikom smo pa uporabili namensko knjižnico DJITelloPy [4], ki vsebuje podprt protokol in dovoljenje za vodenje letalnika.
Središče razvoja je bilo raziskava in preizkus delovanja primernega algoritma vodenja letalnika, pri čemer smo na poti do rešitve raziskali in preizkusili tri različne pristope in ustrezne načine vodenja. Kasneje se je izkazalo, da smo ob tem preverjali tudi delovanje Tello knjižnice, saj ni vedno delovala po pričakovanjih.

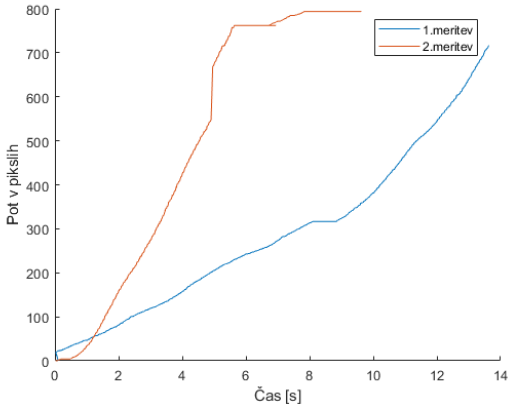
2 Meritve

Pred modeliranjem in simulacijo vodenja letalnika smo morali opraviti meritve odziva letalnika, s čimer smo se sistematično spoznali z delovanjem letalnika.
Pred samim začetkom uporabe kamere na letalniku za orientacijo smo morali kamero umeriti, kar smo storili s pomočjo šahovnice, 30 ročno posnetimi fotografijami in funkcijami znotraj knjižnice OpenCV.
Funkcija *send_rc_control()* iz podane knjižnice DJITelloPy, ki smo jo primarno uporabljali za vodenje letalnika, je simulirala igralni plošček – podobno kot bi letalnik nadzirali s krmilno palico, ki vsebuje potenciometre, katerih odčitki so v območju med 0 do 100. Ta način vodenja je bil sicer težji saj je bilo potrebno letalnik ves čas nadzirati, smo pa s tem pridobili zveznost vodenja in višjo raven nadzora.
V primeru, da bi uporabili vgrajene funkcije pomika - *go_xyz_speed()*, bi dobili diskretne vrednosti z minimalnim pomikom 20 cm in s tem težje natančno pozicioniranje.
Meritve smo pričeli z merjenjem odzivov letalnika na

enotino stopnico, pri čemer smo poslali ukaz s kakšno hitrostjo se naj giba v željeni smeri, nakar smo s pomočjo kamere in pozicijo ArUco značk relativno na letalnika merili celotno pot. Tak način omogoča pridobitev bolj vernih parametrov, ki bi bili uporabni za kasnejšo simulacijo in modeliranje za vse načine vodenja.
Iz pridobljenih podatkov smo v programu Matlab pridobili podatke o časovnem zamiku med letalnikom in računalnikom ter naklon premice s katerim bi lahko ustvarili model za vsako os posebej.
Izvedli smo meritve v levo in desno glede na značko ter gor in dol.

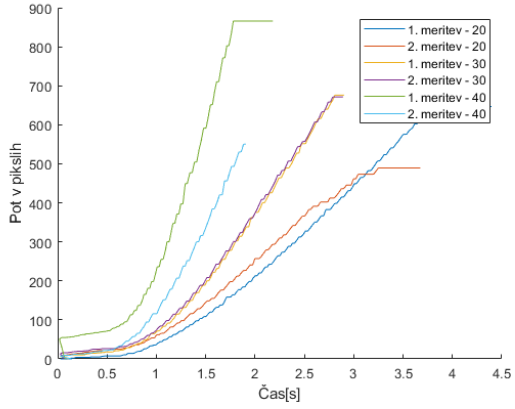
2.1 Gibanje v desno in levo

Pri nižjih željenih hitrostih se letalnik niti ni odzival oziroma je začel nenadzorovano drseti po zraku v vse smeri. Neponovljivost meritve pri nižjih hitrostih je vidna na sliki 3. Meritvi na sliki 3 sta bili izvedeni pri željeni hitrosti 10.



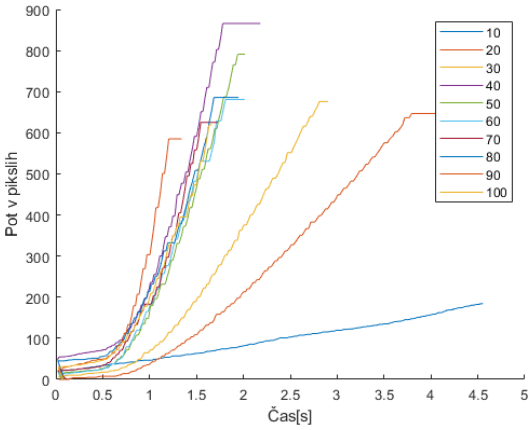
Slika 3: Meritvi pri željeni hitrosti 10.

V razponu hitrosti od 20 do 40 so bile meritve zadovoljivo ponovljive, na sliki 4 so prikazane meritve pri hitrosti 20, 30 in 40. Za vsako hitrost sta bili opravljeni dve meritvi. Izkaže se, da sta bili 20 in 30 izredno ponovljivi med tem ko so se pri hitrosti 40 pojavila odstopanja.



Slika 4: Združene dvojne meritve pri različnih hitrostih.

Zaznali smo omejitev nasičenja zgornje hitrosti, kjer smo začeli z meritvami pri željeni hitrosti v okolici 40, a se višje željene hitrosti niso odrazile v višjih realnih hitrostih, ampak se je letalnik premikal z maksimalno hitrostjo.

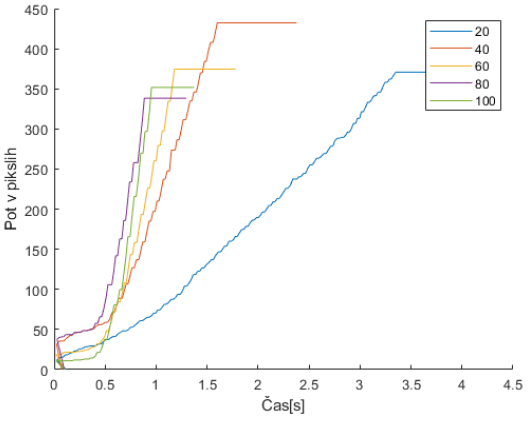


Slika 5: Posamezne meritve pri različnih hitrostih.

Izkazalo se je, da ima letalnik interno omejitev hitrosti pri nadzoru preko računalnika, ki je razvidna iz slike 6. Najbolj strme premice so v isti gruči naprej od hitrosti 40, čeprav bi morale biti ločene. Od tod je sledilo, da letalnik lahko krmilimo samo v razponu hitrosti od 20 do 40, kar se je izkazalo za nezadostno, pri čemer je za vodenje najbolj problematičen spodnji del hitrostnega razpona.
Iz slike 5 je očitno razviden tudi časovni zamik pri odzivu na ukaz računalnika. Ta zamik je bil pri dobri Wi-Fi povezavi okoli 0.6 s.

2.2 Gibanje gor in dol

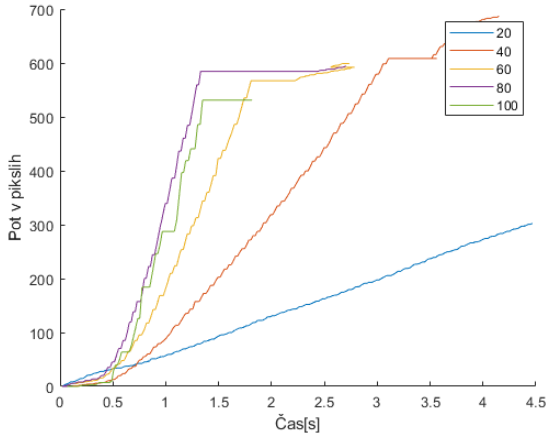
Izvedli smo meritve gibanja gor in dol glede na značko. Hitrost gibanja smo spreminjali v korakih po 20. Najprej smo napravili meritve pri gibanju navzgor, ki so zbrane na sliki 6.



Slika 6: Meritve poti pri gibanju navzgor za različne hitrosti.

Podobno kot pri gibanju levo in desno, so se tudi tukaj pojavile enake omejitve pri gibanju v obeh smereh, kar je

razvidno iz slike 7.
Časovni zamik odziva na ukaz je v povprečju 0.1 s krajši.



Slika 7: Meritve poti pri gibanju navzdol za različne hitrosti.

2.3 Ugotovitve

Med izvajanjem meritev se je izkazalo, da je letalnik izredno občutljiv na več zunanjih vplivov.
Najbolj izrazit vpliv predstavlja nivo baterije, saj se je že pri 10% zmanjšanju napetosti na bateriji letalnik obnašal popolnoma drugače - postajal je vedno bolj nevodljiv, neodziven na ukaze in šibkejši.
Na delovanje je prav tako precej vplivala temperatura baterije in motorjev, saj je letalnik samodejno zmanjševal moč motorjev.
V prostoru z več letalniki se je kot vpliv izkazala tudi kakovost Wi-Fi povezave, saj se včasih ni želel povezati ali pa sprejeti ukaza.
Zaradi uporabe video kamere za orientacijo in vodenje je imela osvetljenost prostora tudi zelo velik vpliv na delovanje. Prvi vpliv je padec hitrosti sličic na glavni kameri in s tem posledično počasnejše odzivanje na okolje. Ta vpliv bi lahko rešili, če bi lahko spreminjali parametre kamere, kot so osvetljenost, hitrost zajema, kontrast… ampak nanjo nismo imeli nobenega vpliva, saj deluje v avtomatskem načinu. Drugi vpliv osvetljenosti izhaja iz IR kamere in laserja na dnu letalnika, ki sta interno uporabljena za stabilizacijo letalnikove pozicije in višine.
Zaradi obilice naštetih vplivnih parametrov okolice in letalnika parametrov modela nismo mogli določiti do mere željene uporabnosti, a smo navzlic temu iz pridobljenega niza parametrov modela izvedli simulacije, ki so na žalost odražale naštete omejitve okolice in letalnika.

3 Algoritem

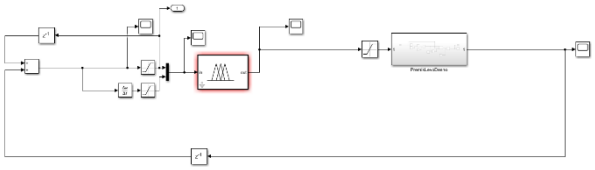
Izdelali in preizkusili smo dva algoritma. Zaradi konca tekmovanja nam premišljene in delno izdelane tretje metode ni uspelo preizkusiti.

3.1 Mehka logika

Pri izbiri algoritma smo največ časa posvetili mehki logiki, saj je kot najzanesljivejša in najhitrejša rešitev obetala največ. Opravili smo tudi preizkus delovanja, ki pa se zaradi nestabilnosti sistema, ki izvirajo iz naštetih omejitev letalnika, ni izšel po pričakovanjih.

3.1.1 Modeliranje

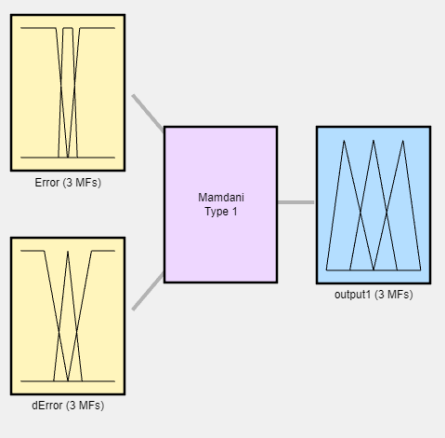
Iz pridobljenih meritev na slikah 3-7 smo določili naklon premice in po metodi Strejcev [5] izvedli simulacijo letalnika v programu Matlab Simulink [6] z uporabo mehke (ang. fuzzy) logike. V programu Matlab Simulink smo sestavili model, ki je prikazan na sliki 8.



Slika 8: Shema modela uporabljenega za simulacijo.

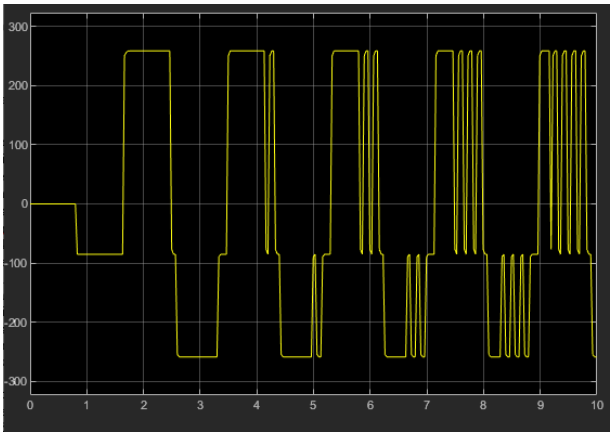
3.1.2 Iskanje parametrov in simulacija

Osnova za iskanje niza parametrov so bili ročno nastavljeni parametri mehke logike. Na sliki 9 vidimo grafično predstavitev osnovnih parametrov, kjer »Error« pomeni odmik od središča slike, »dError« pomeni hitrost spremembe. Fuzzy modele in parametre smo nastavljali v aplikaciji Matlab Fuzzy logic designer [7]. Pridobljene parametre smo nato vstavili v večji model v programu Matlab Simulink – slika 8.



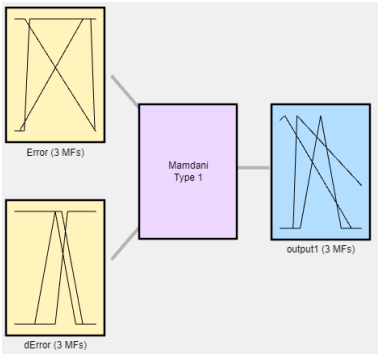
Slika 9: Izgled osnovnih parametrov.

Na sliki 10 vidimo rezultat simulacije z uporabo osnovnih parametrov videnih na sliki 9.



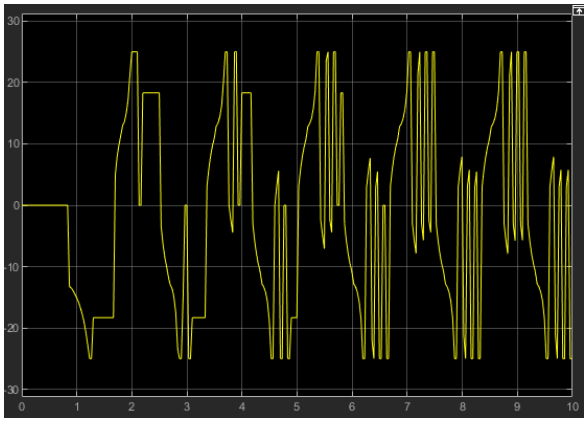
Slika 10: Simulacija odziva z osnovnimi parametri.

Kvalitetnejše parametre mehke logike smo pridobili tako, da smo uporabili osnovne parametre in jih s pomočjo Matlab programa in vgrajenih funkcij za iskanje optimalnih parametrov z genetskimi algoritmi tudi poiskali. Slika 11 prikazuje vrednosti pridobljenih parametrov mehke logike.



Slika 11: Izgled parametrov po iskanju z genetskimi algoritmi.

Odziv, ki smo ga dobili po uporabi genetskega algoritma za iskanje najboljših parametrov mehkega vodenja je viden na sliki 12.

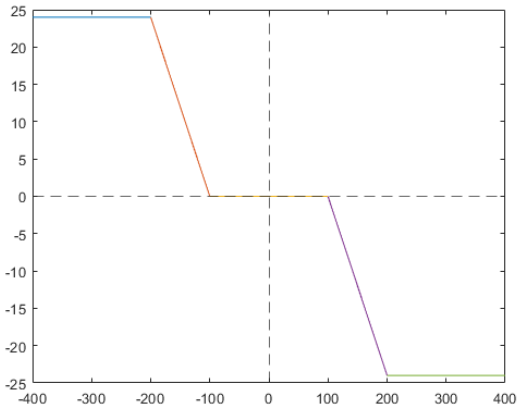


Slika 12: Simulacija odziva.

Z ročnim nastavljanjem parametrov mehkega vodenja in kasneje z genetskim algoritmom iskanja nam ni uspelo pridobiti stabilnega sistema.

3.2 Stikalno krmiljenje

Ker z mehko logiko nismo dosegli pričakovane stabilnosti sistema, smo idejo algoritma poenostavili. V primeru, da je letalnik preveč odmaknjen v eno smer, se prične gibati v nasprotno smer s konstantno hitrostjo. Oddaljenost, pri kateri preklopi smer leta in hitrost gibanja, smo nastavili ročno s preizkušanjem. Opisani način je deloval, čeprav je letalnik pričel večkrat osciliral okoli središčne točke obroča. Središčno točko obroča je zelo redko izgubil, kar je omogočilo prelet skozi obroč. Izdelani algoritem smo dodatno nadgradili z bolj zveznim prehodom pri hitrosti, pri čemer smo uvedli profile hitrosti glede na odmik od središčne točke.
Pozitivna hitrost je pomenila pomik v desno, kar je prištevalo k napaki, negativna hitrost pa je pomenila pomik v levo in je odštevala napako. Na sliki 13 je prikazan profil hitrosti v odvisnosti od napake za gibanje levo in desno.



Slika 13: Profil hitrosti v odvisnosti od odmika letalnika od središča v pikslih.

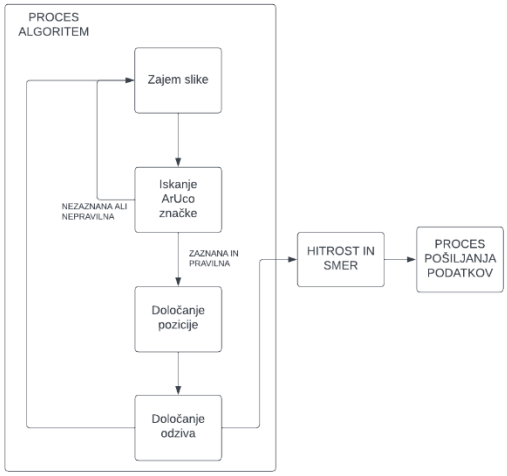
Profil gibanja je bil za vsako os drugačen, a vsi so delovali na podoben način.

3.3 Diskretno krmiljenje

Izkazalo se je, da je letalnik neprimerljivo bolj stabilen pri vodenju z diskretnimi ukazi kot pri vodenju s simulacijo igralnega ploščka. Najmanjši možni pomik je bil 20 cm. Nad to spodnjo mejo smo se lahko premikali v resoluciji 1 cm (npr. 21 cm, 34 cm, 56 cm…).
Algoritem bi upošteval najmanjši pomik 20 cm. Če je središče značke odmaknjeno za manj kakor 20 cm, bi se letalnik premaknil za 20 cm v dodatek napaki in potem nazaj (za napako ± 20 cm), kar bi nas v teoriji postavilo na točko 0 cm. To bi napravili za vse tri osi – levo/desno, naprej/nazaj, gor/dol, kar bi nas postavilo v popolno pozicijo za premik skozi obroč.
Ta algoritem ni bil nikoli preizkušen, bil pa je premišljen, zasnovan in delno kodiran.

4 Implementacija algoritma

Zaradi velikih zakasnitev, ki smo jih odkrili med meritvami letalnika in odvisnosti obnašanja letalnika od zunanjih dejavnikov ter nepoznavanja/nedostopnosti interne kode letalnika, smo se odločili, da bomo implementirali preprostejši algoritem od začetno predvidene mehke logike. Odločili smo se za stikalno krmiljenje, ki se je na koncu zaradi svoje preprostosti in hitrega delovanja izkazal kot najbolj primeren.
Algoritem se je izvajal na računalniku in napisan je bil v programskem jeziku Python. Na zajeti sliki smo najprej poiskali ArUco značko in ji določili številko, če je imela značka željeno številko smo ji še določili pozicijo glede na sredino posnetka. Zaznano pozicijo oziroma odmik od središča smo posredovali funkciji za stikalno logiko, katera je vrnila potrebno hitrost in smer gibanja ter ga predala procesu za pošiljanja podatkov, ki se je ves čas izvajal v ločeni niti, zato da letalnik ne bil šel v neaktivno stanje in bil brez nadzora.



Slika 14: Shema delovanja algoritma.

5 Zaključek

Predstavili smo izvedene algoritme vodenja brezpilotnega letalnika pri premagovanju poligona ovir. Kljub začetnemu fokusu na mehko logiko, za katero se je izkazalo, da je sistem preveč labilen in odvisen od zunanjih dejavnikov, se je od vseh treh algoritmov najbolje odrezal najpreprostejši – stikalna logika. Zaradi svoje preprostosti delovanja in implementacije je deloval najhitreje in najbolj zanesljivo.

Med procesom razvoja in izdelavo različnih algoritmov smo se srečali z množico težav in rešitev. Težavo je predstavljalo nepoznavanje in nedostopnost interne kode letalnika, da bi lahko krmiljenje izvedli kar na njegovem procesorju. Največji problem je bila odvisnost letalnika od mnogih zunanjih dejavnikov, ki so močno vplivali na delovanje in zmožnosti delanja meritev.

V prihodnosti si vsekakor želimo izboljšati model letalnika, da ga bomo lahko dobro simulirali in pridobili kvalitetne parametre za mehko logiko, da bi lahko preizkusili tudi implementacijo in vodenje z mehko logiko. Da bi lahko izvedli popolno primerjavo vseh treh algoritmov, si želimo tudi preizkusiti delovanje diskretnega vodenja.

Z razvitim in implementiranim algoritmom smo se udeležili tekmovanja DroneChallenge2023 [8], kjer smo se pomerili še z ostalimi tekmovalci in dosegli četrto mesto. Zahvala ekipi DroneChallenge za odlično organizacijo in priložnost.

Literatura

[1] Ryze Robotics, "Tello - Ryze Robotics," [Online]. Available: https://www.ryzerobotics.com/tello. [Dostopano: 18.07.2023].

[2] OpenCV, "ArUco Marker Detection - OpenCV 4.x Documentation," [Online]. Available: https://docs.opencv.org/4.x/d5/dae/tutorial_aruco_detection.html. [Dostopano: 18.07.2023].

[3] OpenCV, "OpenCV: Open Source Computer Vision Library," [Online]. Available: https://opencv.org/. [Dostopano: 18.07.2023].

[4] D. Fuentes, "DJITelloPy: Python Library for DJI Tello Letalnike," GitHub. [Online]. Available: https://github.com/damiafuentes/DJITelloPy. [Dostopano: 18.07.2023].

[5] Strejc V, Näherungsverfahren für aperiodische Übergangscharakteristiken. Regelungstechnik 7, (1959), p.124–128. [Dostopano: 18.07.2023].

[6] MathWorks, "Simulink - MATLAB & Simulink," [Online]. Available: https://www.mathworks.com/products/simulink.html. [Dostopano: 18.07.2023].

[7] MathWorks, "Fuzzy Logic Designer App - MATLAB & Simulink," [Online]. Available: https://www.mathworks.com/help/fuzzy/fuzzylogicdesigner-app.html. [Dostopano: 18.07.2023].

[8] Dnevi Avtomatike, "DroneChallenge2023," [Online]. Available: https://dnevi-avtomatike.si/?page_id=429. [Dostopano: 18.07.2023].

[9] E. MUJIĆ, „Vodenje brezpilotnega letalnika s pomočjo kamere“, Diplomsko delo, Univerza v Ljubljani, 2022 [Online]. Dostopano: https://repozitorij.uni-lj.si/IzpisGradiva.php?id=140024. [Dostopano: 18.07. 2023]

[10] MathWorks. "Tune Fuzzy Inference Systems - MATLAB & Simulink," MathWorks. [Online]. Available: https://www.mathworks.com/help/fuzzy/tune-fuzzy-inference-systems.html. [Dostopano: 18.07.2023]

[11] V. Indrawati, "Waypoint Navigation for Unmanned Ground Vehicles," Universitas Surabaya, Surabaya, Indonesia, 2018. [Online]. Available: http://repository.ubaya.ac.id/26729/7/Veronica%20IndraIndr_Waypoint%20Navigation.pdf. [Dostopano: 18.07.2023]

[12] MathWorks. "Design Fuzzy Logic Controller for Artificial Pancreas - MATLAB & Simulink," MathWorks. [Online]. Available: https://www.mathworks.com/help/fuzzy/design-fuzzy-logic-controller-for-artificial-pancreas.html. [Dostopano: 18.07.2023]