Projektiranje i izgradnja drona

Sinovčić, Goran

Master's thesis / Diplomski rad

2023

Degree Grantor / Ustanova koja je dodijelila akademski / stručni stupanj: **University of Zagreb, Faculty of Organization and Informatics / Sveučilište u Zagrebu, Fakultet organizacije i informatike**

Permanent link / Trajna poveznica: https://urn.nsk.hr/urn:nbn:hr:211:806995

Rights / Prava: Attribution-NoDerivs 3.0 Unported/Imenovanje-Bez prerada 3.0

Download date / Datum preuzimanja: **2024-05-13**

SVEUČILIŠTE U ZAGREBU
FAKULTET ORGANIZACIJE I INFORMATIKE
VARAŽDIN

Repository / Repozitorij:

Faculty of Organization and Informatics - Digital Repository

zir.nsk.hr

dabar
DIGITALNI AKADEMSKI ARHIVI I REPOZITORIJI

SVEUČILIŠTE U ZAGREBU

FAKULTET ORGANIZACIJE I INFORMATIKE

V A R A Ž D I N

Goran Sinovčić

Projektiranje i izgradnja drona

DIPLOMSKI RAD

Varaždin, 2023.

SVEUČILIŠTE U ZAGREBU

FAKULTET ORGANIZACIJE I INFORMATIKE

V A R A Ž D I N

Goran Sinovčić

Matični broj: 0016116609

Studij: Baze podataka i baze znanja

Projektiranje i izgradnja drona

DIPLOMSKI RAD

Mentor:

Prof. dr. sc. Neven Vrček

Varaždin, kolovoz 2023.

Goran Sinovčić

Izjava o izvornosti

Izjavljujem da je moj završni/diplomski rad izvorni rezultat mojeg rada te da se u izradi istoga nisam koristio drugim izvorima osim onima koji su u njemu navedeni. Za izradu rada su korištene etički prikladne i prihvatljive metode i tehnike rada.

Autor potvrdio prihvaćanjem odredbi u sustavu FOI-radovi

Sažetak

Cilj ovog diplomskog rada je sveobuhvatno projektiranje i konstrukcija drona, odnosno kvadkoptera koristeći suvremene alate. U teorijskom dijelu definiraju se ključni koncepti kao i evolucijska povijest dronova. Također je dana teorijska osnova leta kvadkoptera te primjene u stvarnim industrijama, kao i legalna osnova za operiranjem dronova. Opisati će se programski jezici koji su korišteni u praktičnom dijelu. U samom praktičnom dijelu odabiru se komponente na osnovi potrebnih funkcionalnih jedinica, s ciljem ostvarenjem potrebnih karakteristika performansi. Projektira se tiskana pločica prilagođena za ovu svrhu pomoću alata za elektronički razvoj KiCad. Razvija se programski kod koji povezuje mikrokontroler sa svojim modulima i implementira sve potrebne funkcionalnosti. Izraditi će se aplikacija za Android uređaje u Android studio-u koja se povezuje na sam dron i pruža grafičko sučelje i kontrole korisniku. Na posljetku dan je zaključak i osvrt na rad.

Ključne riječi: dron; bespilotne letjelice; ESP8266; MPU-6050; BME280; kvadkopter; projektiranje; izgradnja; programiranje; Arduino; Android; Kicad;

Sadržaj

Sadržaj iii
1. Uvod 1
2. Metode i tehnike rada 2
3. Bespilotne letjelice 2
3.1. Povijest dronova 3
3.2. Teorija operacije 4
3.3. Primjene u industriji 7
3.4. Regulatorni okvir 8
4. Programski jezici korišteni za implementaciju drona 9
4.1. Android studio 9
4.2. Arduino IDE 10
4.3. KiCad EDA 11
5. Projektiranje drona 13
5.1. Izbor flight controllera (FCU) 13
5.1.1. Jedinice temeljene na mikrokontrolerima 13
5.1.2. Računala s jednom pločom (SBC) 13
5.1.3. Hibridni sustavi 14
5.1.4. ESP8266 15
5.2. Izbor motora 18
5.2.1. Istosmjerni motori s četkicom 18
5.2.2. Motori bez jezgre (eng. coreless) 19
5.2.3. Istosmjerni motori bez četkica 19
5.3. Izbor jedinice za inercijsko mjerenje (IMU) 20
5.3.1. IMU-ovi temeljeni na MEMS-u 20
5.3.2. Žiroskopi s optičkim vlaknima (FOG) IMU 21
5.3.3. Hibridni IMU 21
5.3.4. MPU-6050 22
5.4. Barometar 24
6. Komunikacijski protokoli 25
6.1. I2C 25
6.2. Wi-Fi 27
7. Izrada drona 28
7.1. Elektronički dizajn 28
7.1.1. Izgradnja ESC-a 29
7.1.2. Dizajn tiskane pločice 32

7.2. Programiranje mikrokontrolera 35
7.2.1. Web poslužitelj 36
7.2.2. Barometar 40
7.2.3. PWM 44
7.2.4. Jedinica za inercijsko mjerenje (IMU) 48
7.2.5. PID kontroler 51
7.3. Izrada android aplikacije 56
7.3.1. Telemetrija 57
7.3.2. Joystick element 63
7.3.3. HTTP klijent 71
7.3.4. Snaga signala 74
8. Zaključak 80
Popis literature 81
Popis slika 86
Popis tablica 87

1. Uvod

Bespilotne letjelice, poznatije kao dronovi, pojavile su se kao transformativni alati u raznim industrijama, revolucionirajući način na koji imamo interakciju s okolinom. Među različitim vrstama dronova, kvadrokopteri se ističu svojom manevarskom sposobnošću, stabilnošću i svestranošću. Nadalje kvadrokopteri, opremljeni s četiri rotora raspoređena u simetričan uzorak, stekli su značajnu popularnost zbog svoje sposobnosti statičkog lebdenja, izvođenja agilnih pokreta i nošenja različitih tereta.

Rastuća važnost dronova ukorijenjena je u njihovoj sposobnosti pristupa udaljenim i opasnim okruženjima, snimanja podataka visoke rezolucije i obavljanja zadataka s povećanom učinkovitošću i preciznošću. U poljoprivredi, bespilotne letjelice omogućuju poljoprivrednicima praćenje zdravlja usjeva, procjenu potreba za navodnjavanjem i optimizaciju raspodjele resursa, što dovodi do poboljšanih prinosa i održivih praksi (Torres-Sánchez et al., 2014.). U građevinskom i infrastrukturnom sektoru, bespilotne letjelice pojednostavljuju procese snimanja, mapiranja i inspekcije, što rezultira uštedom troškova i povećanom sigurnošću (Nitha, M, et al. 2022). Osim toga, uporaba bespilotnih letjelica u humanitarnim i ekološkim primjenama olakšava brzi odgovor na katastrofe, pomaže u očuvanju divljih životinja i pruža vrijedne uvide za ekološka istraživanja (Anderson & Gaston, 2013.).

Kako dronovi nastavljaju preoblikovati industrije i društvo, tema projektiranja i izrade praktičnih dronova ima ogromnu važnost u suvremenom krajoliku. Tehnološki napredak i pristupačnost različitih komponenti kao što su kontroleri leta, baterije i komunikacijski sustavi učinili su istraživačima i entuzijastima razvijanje kvadkoptera različitih specifikacija i namjena sve izvedivijim. Primjerice, motivacija autora za izradom drona specifikacija kao u radu je potencijalna vrijednost u edukacijskom sektoru, kroz modularnu izgradnju i relativno nisku cijenu sastavnih komponentni koje omogućuju široku pristupačnost, te i kao platforma za mogući daljnji razvoj i korištenje u industriji.

Ovaj diplomski rad ima za cilj dizajnirati i konstruirati kvadkopter tip drona, prikazujući integraciju različitih tehnologija koje to omogućuju. Sljedeća poglavlja ovog rada bavit će se sveobuhvatnim istraživanjem tehnologije bespilotnih letjelica, počevši s pregledom literature o povijesti i napretku tehnologije dronova. Zatim će biti predstavljena metodologija za dizajn i konstrukciju praktičnog kvadkoptera, kao i implementacija sustava upravljanja. Na posljetku, konačno poglavlje će prikazati zaključke i rezultate izvedene iz ovog istraživanja, zajedno s potencijalnim mogućnostima za buduće radove u području tehnologije dronova.

2. Metode i tehnike rada

U ovom poglavlju biti će opisane tehnike, metode i alati koji su korišteni pri izradi aplikacije i razradi teme.

Prvi alat koji će biti korišten je Kicad, koji služi za dizajniranje elektronike. Biti će potreban za razvoj prilagođene tiskane pločice koja sadrži glavni mikrokontroler koji služi kao FCU (eng. *Flight Control Unit*) , te dodatne module poput akcelerometra i žiroskopa koji služi kao IMU (eng. *Inertial Measurement Unit*), odnosno osigurava da su sve komponente u električnom kontaktu. Izlazne datoteke ovog alata su u „Gerber“ obliku koji služi kao standard u industriji za izradu tiskanih pločica i osiguravaju firmama koje se bave izradom pločica kompatibilnost s njihovim alatima.

Prije izrade pločica elektronički dizajn će se verificirati u alatu za elektroničke simulacije LTSpice, s posebnim naglaskom na verifikaciju dizajna elektroničkog kontrolera brzine (eng. *Electronic Speed Controller*) čija je uloga kontrola motora drona.

Biti će korišten i Arduino studio koji omogućuje programiranje samog mikrokontrolera, odnosno povezivanje s modulima i procesiranje podataka u stvarnom vremenu. Zasniva se na prilagođenoj implementaciji C++ s dodatnim mogućnostima upravljanja Arduino pločicama, ali i pločicama od drugih proizvođača.

Android studio koristit će se za izradu aplikacije za Android uređaje koja služi za primanje i slanje podataka između drona i korisnika. Omogućuje stvaranje grafičkog sučelja koje pruža korisniku informacije o stanju samog drona, kao i davanje ulaznih naredbi za kontrolu drona.

Većina prethodnih alata opisani su u više detalja u teorijskom dijelu rada.

Metode za učenje prije izrade drona zasnivaju se na pretraživanju znanstvenih radova, članaka, disertacija i drugih internetskih resursa za informacije, na multidisciplinarnom području projektiranja i izgradnje dronova.

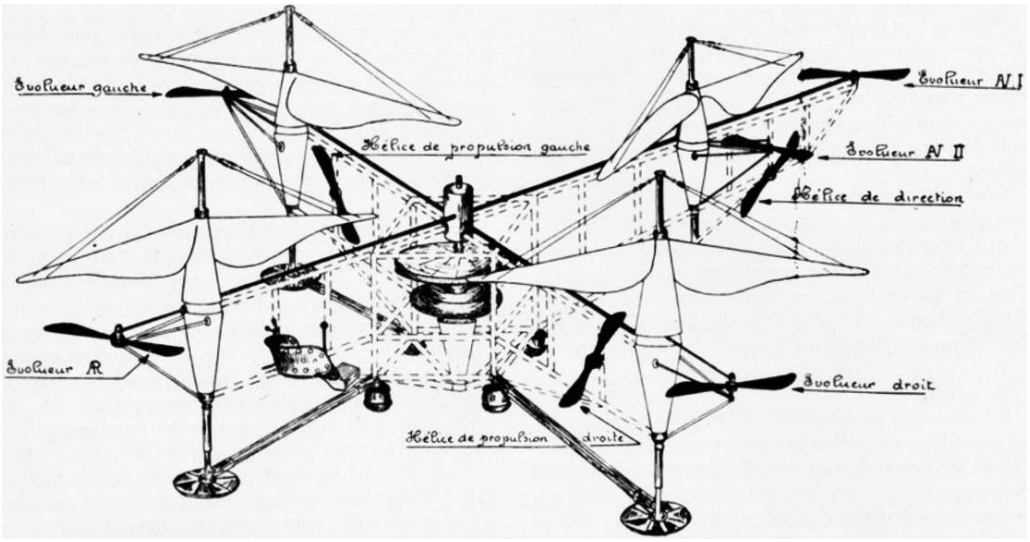
3. Bespilotne letjelice

Bespilotne letjelice, koje se često kolokvijalno nazivaju dronovi, letjelice su bez ljudskog pilota. Oni su proizvod naprednih tehnika inženjeringa i njima upravljaju daljinski piloti sa zemlje, ili su nekim slučajevima autonomno pred-programirani za izvođenje određenih operacija. Postoje brojne vrste dronova, a razlikuju se ovisno o dizajnu, namjeni i mogućnostima. To uključuje dronove s fiksnim krilima, helikoptere s jednim rotorom i dronove s više rotora. Među najčešćim vrstama bespilotnih letjelica su više rotorne bespilotne letjelice, koje uključuju trikoptere, kvadkoptere, heksakoptere i oktokoptere. Kvadrokopteri, koji su ujedno i fokus ovog rada, vrsta

su bespilotnih letjelica koje koriste četiri rotora za pogon i kontrolu. Pripadaju skupini vozila za okomito polijetanje i slijetanje (eng. *Vertical Take Off and Landing (VTOL)*) jer se okomito penju i spuštaju i ne zahtijevaju stazu (Chao, H., et al., 2010.). Sveprisutna priroda kvadkoptera na današnjem tržištu daljinski upravljanih letjelica može se pripisati njihovoj stabilnosti, upravljivosti i relativno jednostavnom dizajnu u usporedbi s drugim bespilotnim letjelicama, te su se iz ovih razloga nametnuli kao simbol revolucije dronova.

3.1. Povijest dronova

Ideja o bespilotnim letjelicama može se pratiti do ranih 1900-ih, kada su vizionari, izumitelji i inženjeri konceptualizirali mogućnost letenja bez pilota. Godine 1916. britanski inženjer Archibald Low razvio je "Aerial Target", prvu letjelicu na daljinsko upravljanje koja se koristila za obuku protuavionskih topnika tijekom prvog svjetskog rata (Miličević, Z et al, 2021.).
Koncept kvadkoptera može se pratiti unazad do ranog 20. stoljeća, s eksperimentima Étiennea Oehmichena. Francuski inženjer izgradio je i upravljao uređajem nalik helikopteru, no s četiri rotora, poznatim kao Oehmichen No.2, koji je 1924. napravio let u zatvorenom krugu od jednog kilometra (Swopes, B., 2023.) Te iako je bio kontroliran ljudskim pilotom na vozilu, to je značilo prekretnicu u povijesti zrakoplovstva, pružajući dokaz koncepta za praktičnost i stabilnost leta s više rotora. Primjerice, već tu je došlo do razvoja dva suprotno rotirajuća rotora za kontrolu rotacije vozila.



Slika 1. Oehmichen no.2 (Izvor: Swopes, B., 2023.)

Zbog tehnoloških ograničenja vremena, rani modeli bespilotnih, daljinsko upravljanih kvadkoptera nisu pružali pouzdan let i uglavnom su prebačeni na eksperimentalne uređaje.

Drugi svjetski rat označio je prekretnicu za razvoj bespilotnih letjelica, jer su vojne snage prepoznale potencijal bespilotnih letjelica za izviđanje i ciljanje bez bespotrebnog rizika za pilota.

Primjerice 1956 godine Gyrodyne QH-50 DASH (*Drone Anti-Submarine Helicopter*) je postao jedan od najranijih primjera daljinsko upravljanog drona svoje vrste. QH-50 je razvila Mornarica Sjedinjenih Američkih Država za protupodmorničko ratovanje i imao je dva suprotno rotirajuća rotora postavljena na središnju osovinu, što mu je davalo stabilnost i manevriranje za pomorske operacije, primarno detekciju i uništavanje neprijateljskih podmornica (National Air and Space Museum, 2019.)

Pojam 'quadcopter' prvi je put dokumentiran kasnih 1950-ih, u patentima za igračke i modele zrakoplova. Međutim, razvoj kvadkoptera kakve poznajemo danas ozbiljno je započeo 2000-ih, uglavnom potaknut napretkom u tehnologiji baterija, senzora i mikrokontrolera. Primjerice, stvaranje litij-polimerske (LiPo) baterije u ranim 2000-ima revolucioniralo je potencijal za male, lagane i energetski guste baterije, što je bio ključni čimbenik u razvoju komercijalno korisnih quadcoptera (Piszcz, A., 2004.). Istodobno, napredak tehnologije mikroelektro-mehaničkih uređaja (MEMS) omogućio je minijaturizaciju žiroskopa i akcelerometara, pružajući kvadkopterima potrebnu stabilnost i sposobnost manevriranja (Elmenreich, W., 2002.). Konačno, u 2000-ima došlo je do eksplozije u razvoju mikrokontrolera, s tvrtkama kao što su Arduino i Raspberry Pi koji su ponudili jeftine, moćne i lako programibilne mikrokontrolere. DJI Phantom, lansiran 2013., bio je jedan od prvih kvadkoptera usmjerenih na potrošače koji je u potpunosti integrirao ove tehnologije u jedan paket jednostavan za korištenje (Svensson,P., 2013.).

Danas su bespilotne letjelice, uključujući kvadkoptere, pronašle primjenu u raznim sektorima, od snimanja videa iz zraka do poljoprivrede, praćenja okoliša, inspekcije infrastrukture i odgovora na različite katastrofe. Kako tehnologija napreduje, istraživači i stručnjaci iz industrije istražuju nove granice u primjeni bespilotnih letjelica, kao što su robotika u rojevima i sustavi. Dodatkom umjetne inteligencije i strojnog učenja dodatno se povećava potencijal mogućnosti bespilotnih letjelica, omogućujući autonomno donošenje odluka i prilagodljiva ponašanja (Makar et al., 2018.).

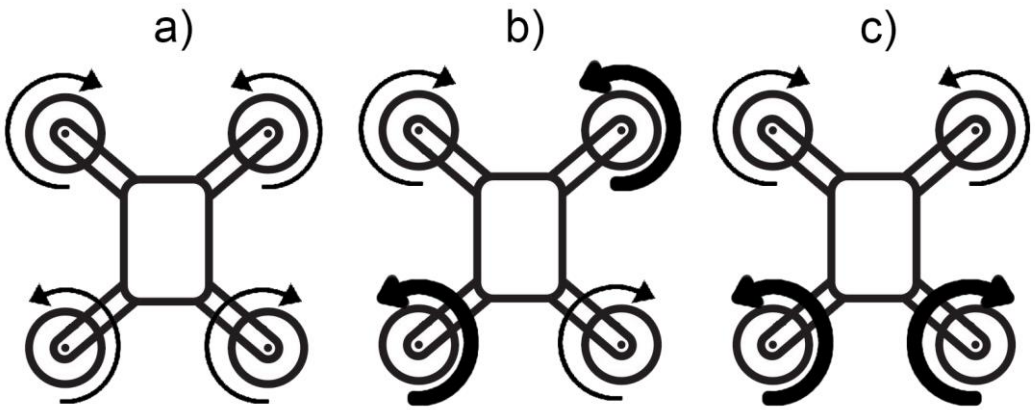
3.2. Teorija operacije

Različite vrste dronova funkcioniraju na različitim principima, te ćemo se stoga ograničiti na dronove koji su primarni interes ovog rada, a to su kvadkopteri. U principu teorija operacije je ukorijenjena u kombinaciji aerodinamike (uglavnom propelera, ali i tijela), kontrolnih algoritama i mehaničkog dizajna.

Sposobnost letenja kvadkoptera podupire Newtonov treći zakon gibanja, koji tvrdi da svaka radnja ima jednaku i suprotnu reakciju. U kontekstu kvadkoptera, rotori guraju zrak prema dolje, a reakcija te sile pokreće kvadkopter prema gore (Bouabdallah et al., 2004.). Količina uzgona koju stvara svaki rotor je funkcija veličine, oblika i brzine vrtnje rotora. Oblici aeroprofila i Bernoullijev princip igraju ključnu ulogu u objašnjenju kako rotori proizvode uzgon. Dok se rotori vrte, stvaraju razliku tlaka: niži tlak iznad i viši tlak dolje, što rezultira potiskom prema gore (Gibson et al., 2010.). Efekt tla, fenomen u kojem prisutnost tla mijenja obrasce strujanja zraka oko rotora, može utjecati na let kvadkoptera, posebno tijekom polijetanja i slijetanja.
Tri osnovne osi upravljaju kretanjem:

- Uspon (ili vertikalna translacija): Uzdizanje ravno gore i spuštanje dole.
- Nagib (ili lateralna translacija): Nagib naprijed ili natrag, lijevo ili desno.
- Skretanje (ili rotacija): Rotacija oko centralne osi.

Upravljanje ovim osima postiže se mijenjanjem brzine pojedinih rotora. Svaki rotor proizvodi uzgon i okretni moment oko središta svoje rotacije, te zbog aerodinamičnosti, i otpor u suprotnom smjeru od kretanja vozila. Kvadrokopteri generalno posjeduju četiri rotora, dva od kojih se okreću suprotno od kazaljke na satu(CCW) i dva u smjeru kazaljke na satu (CW). Ako se sva četiri rotora vrte istom kutnom brzinom, okretni moment oko osi skretanja se poništava i postaje nula. Ako još pri tom sila uzgona točno iznosi težinu drona, i one se poništavaju te dron nastoji ostati na mjestu. Ovakav scenarij prikazan je na slici 2. slučaj a).



Slika 2. Tri osnovne kontrolne osi kvadkoptera (Izvor: Izrada autora)

Ukoliko odaberemo dva suprotna motora i pojačamo njihovo okretanje, njihov uzgon ali i bitnije okretni moment će se povećati te će postati veći od drugog para motora, zbog čega se sile više neće poništiti te će se dron početi rotirati oko svoje osi, što je prikazano u slici 2 slučaj b). Efekt može biti povećan i smanjivanjem drugog para motora. Nadalje, ukoliko odaberemo 2

susjedna motora i povećamo njihovo okretanje, dron će se prirodno povisiti na toj strani a spustiti na suprotnoj, što će uzrokovati horizontalno gibanje odnosno translaciju kroz zrak. Ovo je prikazano u slici 2 slučaj c). Efekt se može povećati na način da se suprotnom paru smanje okretaji. Odabirom pravih parova dron se na ovaj način može gibati i naprijed i nazad, ali i lijevo i desno. Naravno ovo je pojednostavljen slučaj gdje sve tri vrste kontrole gledane zasebno, dok se u stvarno sve tri vrste ulaza događaju istovremeno, uz dodatno nadgledanje i kontrolu PID algoritmom za stabilnost koje moraju biti obrađene u stvarnom vremenu s niskom latencijom kako bi se spriječile oscilacije, što predstavlja značajnu kompleksnost.

U ranim danima letenja, zbog ovakvog načina letenja, kvadkopteri su bili smatrani mogućim rješenjem za neke probleme u vertikalnom letu kojeg su imali helikopteri. Problemi upravljanja izazvani momentom i problemi učinkovitosti koji potječu od repnog rotora (koji ne stvara koristan uzgon) mogu se eliminirati koristeći suprotnu rotaciju. U 1920-im i 1930-im godinama pojavio se niz pilotiranih dizajna, te su ta vozila bila među prvim uspješnim vozilima za okomito polijetanje i slijetanje. Međutim, patili su od loših performansi, a kasniji prototipovi su zahtijevali previše vještine od pilota, zbog loše stabilnosti i ograničenih kontrolnih opcija tog vremena.

Također kvadrokopteri, sa svojim izloženim rotorima i često ne aerodinamičnim tijelima, se susreću sa značajnim otporom, posebno tijekom horizontalnog leta ili brzog uzleta. Sila otpora zraka proporcionalna je kvadratu brzine, stoga su letovi velikim brzinama energetski neefikasni (Mellinger et al., 2011.).

Kako bi riješili te probleme, oslanjaju se na ugrađene senzore, poput žiroskopa i akcelerometara za praćenje orijentacije. Povratne informacije s ovih senzora kontinuirano obrađuje kontroler leta koji prilagođava brzine rotora u stvarnom vremenu kako bi se održala stabilnost i željena putanja leta. Vanjski čimbenici kao što su vjetar, varijacije gustoće zraka i magnetske smetnje mogu nepovoljno utjecati na stabilnost leta. Štoviše, aerodinamičke interakcije između rotora, osobito tijekom agilnih manevara, zahtijevaju zamršene strategije upravljanja kako bi se osigurao nesmetan let (Kushleyev et al., 2013.). Algoritmi kao što je Proportional-Integral-Derivative (PID) kontroler, osiguravaju precizan i stabilan let (Madani & Benallegue, 2006). Mehanički gledano, najjednostavniji scenarij je kada su motori i propeleri drona ekvidistantni, odnosno jednako udaljeni. Nadalje nesimetričnosti u samoj šasiji drona uzrokuju pomak centra gravitacije, i neravne sile otpora zraka pogotovo pri većim brzinama, za što sve moraju kompenzirati kontrolni algoritmi.

Napredni manevri kao što su prevrtanja uključuju složenu međuigru sila i momenta. Inherentna nestabilnost je naglašena tijekom ovakvih manevara, što zahtijeva dodatno sofisticirane algoritme upravljanja za uspješno izvršenje (Muirhead et al., 2013.). Ipak, u ovom poglavlju smo definirali potrebne elemente: treba nam kontroler, tzv. „mozak“ drona, akcelerometar i žiroskop za stabilnost, motori i propeleri, upravljač brzine vrtnje motora (ESC), i šasija koja će držati i povezivati sve ove komponente.

3.3. Primjene u industriji

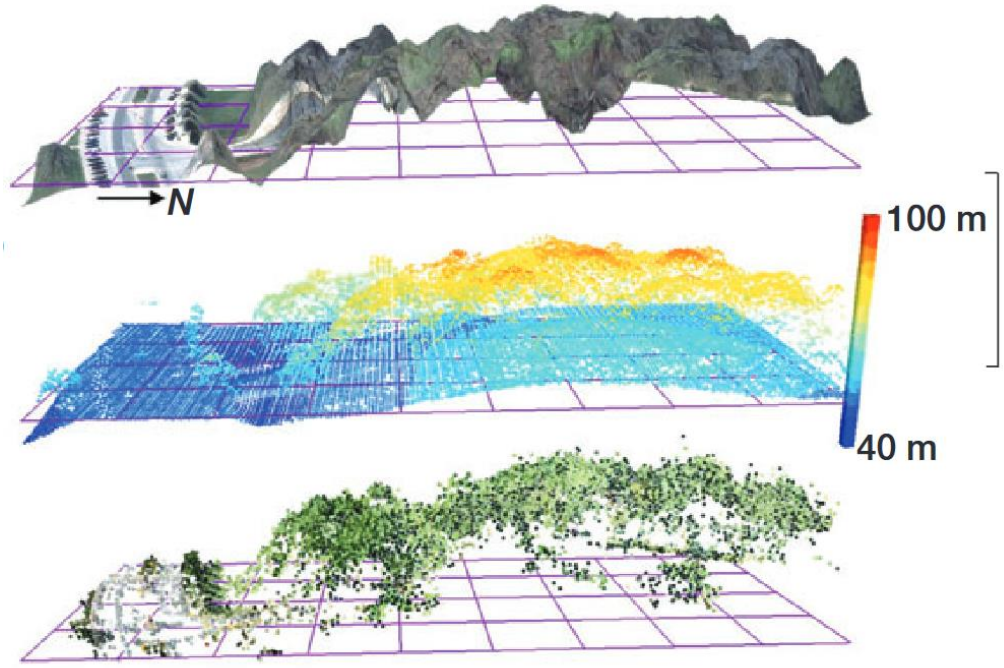
Iako je rani razvoj dronova uglavnom ovisio o ulaganjima za vojne svrhe, oni su prešli iz te niše u širok spektar komercijalnih primjena. Nakon toga, doživjeli su brzi porast svojih tehnoloških mogućnosti i primjena tijekom prošlih desetljeća. Unutar golemog područja industrije, dronovi preoblikuju poslovanje, smanjuju troškove, povećavaju učinkovitost i nude nove mogućnosti.

Značajan udio rasta industrije dronova može se pripisati poljoprivredi. Bespilotne letjelice pružaju metodu za preciznu poljoprivredu, primjerice koriste se za analizu tla i polja, sadnju, ali i prskanje pesticida i gnojiva. Dronovi opremljeni multi spektralnim senzorima mogu detektirati promjene u zdravlju biljaka puno prije ljudskog oka, što omogućuje pravovremene intervencije i bolje upravljanje usjevima (Torres-Sánchez et al., 2014.). Također, dronovi su pronašli i ulogu u lokaliziranju i kontroli štetočina i predviđanju prinosa od žetve, što može pomoći predvidjeti i amortizirati štete i pripremiti tržište ukoliko žetva nije dobra. Sve ove operacije mogu se automatizirati kroz praćenje predefiniranog puta pomoću GPS-a, te se na taj način izvode rutinski.

Dronovi također igraju i neizostavnu ulogu u inspekciji i održavanju infrastrukture poput mostova, zgrada i dalekovoda. Tradicionalne metode inspekcije često su opasne, radno intenzivne i skupe. Dronovi, sa svojim naprednim senzorima i mogućnostima snimanja, predstavljaju sigurniju i učinkovitiju alternativu. Oni nude podatke u stvarnom vremenu, omogućujući inženjerima i arhitektima praćenje napretka izgradnje, procjenu strukturalnog integriteta i osiguravanje usklađenosti sa specifikacijama dizajna. Čak i prije inspekcije i održavanja, u samom procesu građevinarstva bespilotne letjelice pomažu u mapiranju gradilišta, praćenju napretka i poboljšanju sigurnosti na gradilištu (Nitha, M, et al. 2022.). Na taj način kompanije bolje mogu procijeniti napredak i izvješćivati klijente i investitore o vremenskim okvirima izgradnje. Prije nego sama izgradnja uopće započne, često je potrebno napraviti topografska istraživanja koja treba predati regulatornim tijelima, koja se rutinski izvršavaju pomoću dronova.

Sektor rudarstva revolucionirao je bespilotne letjelice koje se koriste za volumetrijske proračune, topografsko mapiranje i praćenje operacijskih aktivnosti. UAV nudi prikupljanje podataka u stvarnom vremenu, smanjujući rizike osoblju i povećavajući operativnu učinkovitost, posebno na opasnim terenima (Bork, 2007).

Dronovi igraju ključnu ulogu u istraživanju i očuvanju okoliša. Bespilotne letjelice pomažu u praćenju krčenja šuma, praćenju divljih životinja i procjeni promjena staništa, nudeći neusporedivu prednost koja je prije bila nedostižna ili skupa. Omogućuju pogled iz ptičje perspektive na velika područja, omogućujući istraživačima prikupljanje podataka bez narušavanja prirodnog staništa (Anderson & Gaston, 2013.).



Slika 3. LIDAR snimka šume, s algoritamski odvojenom krošnjom od tla (Izvor: Anderson & Gaston, 2013.)

Također omogućuju i mapiranje velikih površina terena pomoću LIDAR senzora, te kroz različite napredne algoritme moguće je digitalno ukloniti krošnje drveća i analizirati tlo ispod što donosi veliku korist mnogim zavodima i privatnim istraživačima (Anderson & Gaston, 2013).

Na posljetku dronovi imaju i vidljivu ulogu u svakodnevnom životu. Svi smo i upoznati sa slikama različitih pejsaža koje objavljuju turističkih zajednice, i impresivnim slikama nekretnina, koji nam omogućuju novu i impresivnu perspektivu na okoliš koji nas okružuje.

3.4. Regulatorni okvir

Pravni okvir koji regulira bespilotne letjelice složena je domena koja se razvija i uključuje postizanje ravnoteže između omogućavanja inovacija i zaštite privatnosti, sigurnosti i javne sigurnosti. Sveobuhvatne norme za globalno zrakoplovstvo, uključujući bespilotne letjelice, uspostavila je Međunarodna organizacija civilnog zrakoplovstva (ICAO) (Clarke, 2014.). Međutim, mnogi aspekti propisa o bespilotnim letjelicama potpadaju pod nadležnost nacionalnih zrakoplovnih vlasti. Agencija Europske unije za sigurnost zračnog prometa (EASA) implementirala je usklađeni regulatorni okvir za bespilotne letjelice u zemljama EU-a. Propisi se temelje na riziku i podijeljeni su u „otvorene”, „specifične” i „certificirane” kategorije

(European Union Regulations, 2019.). Hrvatska, kao članica Europske unije, usklađuje svoje propise o bespilotnim letjelicama sa smjernicama EASA-e, dok Hrvatska agencija za civilno zrakoplovstvo (CCAA) nadzire provedbu i provedbu tih propisa. U skladu s okvirom EASA-e, hrvatske operacije bespilotnih letjelica dijele se na 'otvorene', 'specifične' i 'certificirane' kategorije, na temelju rizika koji operacija predstavlja (European Union Regulations, 2019.; CCAA, 2023.) . Propisi zabranjuju rad bespilotnih letjelica u zonama zabrane leta, poput zračnih luka, iznad gužve ili u urbanim područjima bez posebne dozvole. Štoviše, bespilotne letjelice uvijek bi trebale ostati unutar vidnog polja operatora i ne bi trebale prelaziti visinu od 120 metara. Operateri se nisu dužni registrirati ako je riječ o bespilotnim zrakoplovima koji se smatraju igračkama u skladu s direktivom 2009/48/EZ, odnosno ako nisu opremljeni senzorima koji mogu prikupljati osobne podatke (CCAA, 2023.). O pogledu ovog legalnog okvira, nabava licence neće biti potrebna u ovom radu.

4. Programski jezici korišteni za implementaciju drona

U prošlom poglavlju smo se upoznali sa potrebnim sastavnicama dronova, ali implementacija ovog hardvera zahtjeva i softversku komponentu. Odabir programskih jezika i alata igra ključnu ulogu u funkcionalnosti i performansama drona.
Kombinacija Arduina, Androida i KiCad-a pruža opsežan skup alata za razvoj drona. Arduino omogućuje izravnu kontrolu nad hardverskim komponentama, Android pruža platformu za razvoj kontrolne aplikacije, a KiCad omogućuje elektronički dizajn prilagođen hardverskim komponentama. Zajedno, ovi alati nude fleksibilnu i moćnu platformu za razvoj drona.

4.1. Android studio

Android Studio, službeno razvojno okruženje (IDE) za Googleov operativni sustav Android, je alat za razvoj aplikacija za razne uređaje od svog početka 2013. godine. Izgrađen je na JetBrains-ovom IntelliJ IDEA softveru i dizajniran je posebno za Android razvoj, pruža alate za izradu aplikacija na svim vrstama Android uređaja. Android Studio nudi širok raspon značajki koje ga čine svestranim alatom za programiranje Ima moćan uređivač koda s naprednim dovršavanjem koda, refaktoriranjem, analizom koda i bojanje sintakse. Uređivač vizualnog izgleda aplikacije omogućuje programu da dizajnira korisnička sučelja za svoje aplikacije sa drag-and-drop definiranih komponenti korisničkog sučelja, poput botuna, tekst kutija, i sličnog. Ovaj IDE nudi sveobuhvatno uklanjanje pogrešaka, postavljanje breakpoint-ova, pregled

stanja aplikacije na tim točkama i izvršavanje liniju po liniju. Android Studio također uključuje paket alata koji pomažu programerima da optimiziraju izvedbu svojih aplikacija, uključujući profiler memorije, CPU profiler i mrežni profiler (Anan et al., 2021).

AVD Manager u Android Studiju omogućuje programerima da testiraju svoje aplikacije na različitim Android uređajima i verzijama bez potrebe za fizičkim uređajima. Ovi virtualni uređaji mogu se prilagoditi za simulaciju različitih hardverskih značajki, veličina zaslona i verzija Androida.

U teoriji, korištenje Android studija nam omogućuje moćne mogućnosti, zbog tipično velike procesorske moći uređaja na kojima se kod izvršava. Primjerice, bilo bi moguće dodati napredne funkcionalnosti poput planiranja leta i automatske kontrole. Fleksibilnost ove platforme omogućuje stvaranje aplikacija prilagođenih specifičnim potrebama korisnika.

4.2. Arduino IDE

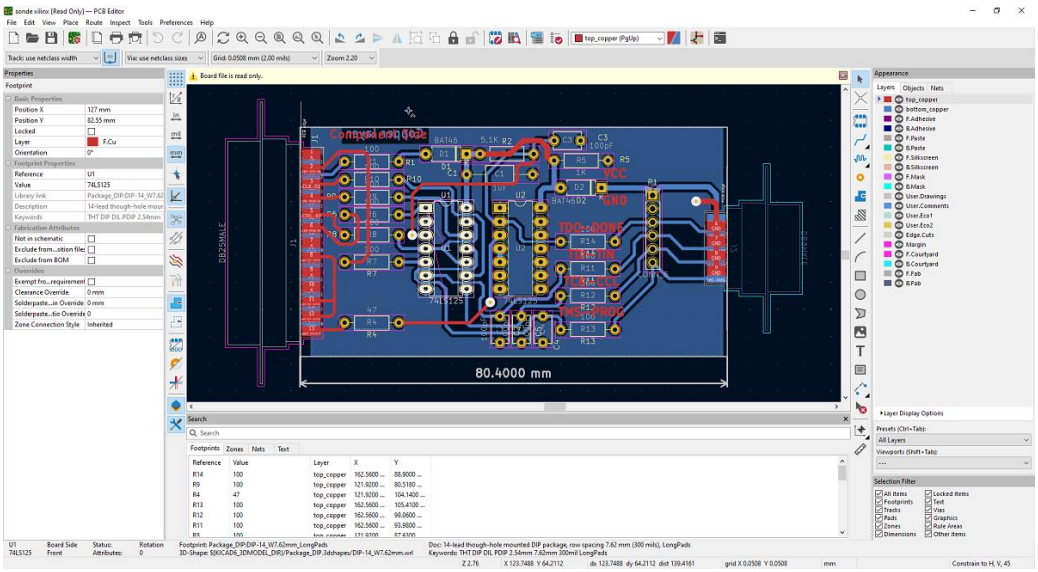
Arduino, elektronička platforma otvorenog koda, značajno je transformirala proces izrade prototipa hardvera i DIY elektroničkih projekata. Središnji dio ove platforme je Arduino Integrated Development Environment (IDE), intuitivna aplikacija za pisanje, kompiliranje i učitavanje programa na Arduino pločice. Projekt Arduino, kojeg je 2005. pokrenula skupina talijanskih istraživača, zamišljen je kao platforma jednostavna za korištenje za početnike zainteresirane za stvaranje digitalnih uređaja i interaktivnih objekata. U početku izgrađen oko Processing-a, drugog programskog jezika i okruženja otvorenog koda je evoluirao tijekom vremena, proširujući svoj doseg na široku raspon Arduino pločica što pruža pogodno okruženje za razvoj. Arduino IDE pruža pojednostavljeno okruženje kodiranja za pisanje skica (eng. *sketch*, naziv Arduino datoteke koda) u Arduinovoj varijanti jezika C/C++.

Sadrži bojanje za ključne riječi, automatsko uvlačenje i podudaranje vitičastih zagrada, što sve poboljšava čitljivost koda. Nakon što je kod napisan, IDE nudi kompilaciju jednim klikom, provjeru grešaka i prevođenje koda u format koji Arduino hardver može razumjeti (Margolis, 2011.). Arduino IDE dolazi s integriranim upraviteljem pločica koji pojednostavljuje proces dodavanja podrške za različite Arduino ploče. Korisnicima omogućuje odabir između raznih ploča, od standardnih poput Arduino Uno do specijaliziranijih opcija poput Sparkfun RED-V i ESP8266. Osim toga, Library Manager omogućuje korisnicima jednostavno uključivanje unaprijed napisanih biblioteka u svoje projekte. Te biblioteke često pružaju pojednostavljena sučelja za rad s određenim hardverom ili izvršavanje određenih zadataka, značajno ubrzavajući proces razvoja (Monk, 2013). Arduino IDE ima serijski monitor koji omogućuje komunikaciju u stvarnom vremenu između Arduino pločice i računala. Ova funkcija je vrijedna

za otklanjanje pogrešaka i može se koristiti za slanje ili primanje podataka (McRoberts, 2011). Arduino IDE u kombinaciji s velikim brojem podržanog hardvera, pronašao je široku primjenu u brzoj izradi prototipova, omogućujući tvrtkama, istraživačima i hobistima brzo i jeftino eksperimentiranje s elektroničkim dizajnom. Također, jednostavnost Arduino IDE čini ga izvrsnim alatom za obrazovne svrhe, podučavajući studente o kodiranju, elektronici i fizičkom računalstvu (slično poput BBC micro-bit-a trenutno u upotrebi u osnovnoškolskom nastavnom kurikulumu). IDE-ovo korisničko sučelje i zajednica koja ga podržava čine ga idealnim polazištem za početnike koji istražuju svijet ugrađenih sustava (Monk, 2013.). Nažalost nekim korisnicima takva jednostavnost može predstavljati i ograničenje, zbog manjka naprednih funkcionalnosti poput debugging-a. Također, korisnici koji koriste nedavno objavljene pločice za koje još ne postoji podrška moraju čekati dok se ona ne doda. To znači varijabilno vrijeme čekanja zbog prirode otvorenog koda i volonterskog pristupa razvoju, iako generalno vrijeme nije pre dugačko i često korporacija koja izdaje pločicu i napiše pripadajuću biblioteku za Arduino IDE, zbog njegove popularnosti.

4.3. KiCad EDA

KiCad, program za dizajniranje elektronike (EDA) otvorenog koda, pojavio se kao moćan alat za dizajnere koji se bave stvaranjem shema i tiskanih ploča (PCB). Iako nije programski jezik, KiCad igra ključnu ulogu u procesu dizajna hardvera drona, dopuštajući dizajn i optimizaciju prilagođenog PCB-a. Pokrenut 1992. godine od strane Jean-Pierre Charrasa, francuskog akademika, KiCad je zamišljen kao softverski paket za olakšavanje dizajna PCB-a i olakšavanje razvoja elektroničkih sustava. Prima kontinuirani razvoj od svog početka, sa aktivnom zajednicom suradnika koji su s vremenom dodavali nove značajke i optimizacije (Kicad, 2023). KiCadov paket uključuje Eeschema, program za snimanje shema, i Pcbnew, alat za izradu tiskanih ploča. Zajedno, oni omogućuju dizajnerima da svoje ideje prevedu u formalne sheme, a zatim ih pretvore u fizičke dizajne za izradu PCB-a. Nudi bogatu biblioteku komponenti, uključujući otiske, shematske simbole i 3D modele. Korisnici mogu dodatno prilagoditi ove biblioteke, stvarajući svestrano i korisniku prilagođeno okruženje za dizajn elektronike. Softverski paket također uključuje ugrađeni 3D preglednik koji može generirati 3D prikaze dizajnirane PCB pločice (Kicad, 2023). To omogućuje dizajnerima da i vizualno provjere svoje dizajne i spriječe pogreške prije proizvodnje.



Slika 4. Prikaz korisničnog sučelja KiCad EDA (Izvor: kicad.org)

KiCad može generirati Gerber datoteke, standardni format koji koriste kuće za proizvodnju tiskanih pločica. Ova značajka osigurava da dizajneri mogu jednostavno prenijeti svoje KiCad dizajne u proizvodnju. Kao alat otvorenog koda, KiCad pruža kompetitivnu alternativu komercijalnom EDA softveru, često nudeći vrlo usporedive značajke bez povezanih troškova. Zbog toga su ga široko prihvatili hobisti, akademici, pa čak i neki komercijalni subjekti, posebno oni koji žele investirati u softver otvorenog koda. Štoviše, sposobnost KiCad-a da radi na različitim operativnim sustavima (Windows, Linux i macOS) proširuje njegovu korisničku bazu i jača njegovu reputaciju (Kicad, 2023). Jedan od potencijalnih nedostataka je relativno strma krivulja učenja za početnike, iako je napravljeno puno napretka u izradi intuitivnijeg sučelja u novijim verzijama. Također, postoje vrlo ograničene integrirane opcije simulacije shema, zbog kojih će korisnici morati koristiti dodatna softverska rješenja. Budući da je projekt otvorenog koda, KiCad može imati i sporija vremena odaziva na ispravljanje greški i ažuriranja u usporedbi s komercijalnim alatima, što može dovesti do prekida rada ako korisnici naiđu na neriješen problem, no u praksi broj takvih problema je ograničen.

5. Projektiranje drona

U sljedećem poglavlju opisan je proces planiranja i projektiranja drona, kroz pronalaženje različitih komponenti potrebnih za ispunjavanje željenih specifikacija drona. Za svaku funkcionalnu jedinicu drona opisane su moguće vrste rješenja, njihove prednosti i nedostatci, te na posljetku je odabran određeni modul koji najbolje odgovara izazovima izgradnje drona.

5.1. Izbor flight controllera (FCU)

Jedinica za kontrolu leta (FCU) djeluje kao mozak kvadkoptera, integrirajući podatke senzora i implementirajući logiku upravljanja kako bi se osigurao stabilan i kontroliran let (Kendoul, 2012.). Kako polje dizajna kvadkoptera sazrijeva, razvijene su različite vrste FCU-ova, svaki s jedinstvenim prednostima i ograničenjima. Ovo poglavlje bavit će se dubinskom usporedbom prednosti i nedostataka uobičajeno korištenih tipova FCU-ova: jedinica temeljenih na mikrokontrolerima, računala s jednom pločom (SBC) i hibridnih sustava.

5.1.1. Jedinice temeljene na mikrokontrolerima

Jedinice temeljene na mikrokontrolerima, kao što su Arduino i PIC mikrokontroleri, nude jednostavnost, ekonomičnost, te sa svojom niskom potrošnjom struje i energetsku učinkovitost (Lupashin i sur., 2010.). Rade u stvarnom vremenu, i imaju dovoljnu procesorsku snagu za tipične algoritme upravljanja kvadkopterom. Sposobni su operacijom s vrlo niskom latencijom što omogućuje trenutni odaziv za input-e korisnika ili podataka iz senzora, što je kritično za stabilnost. Također, zbog njihove open-source prirode imaju velik broj biblioteka i podrške dostupne na internetu, što pojednostavljuje razvijanje i podršku. Zbog velikog broja GPIO header-a, ostvarivanje povezanosti s dostupnim dodatnim modulima ne predstavlja problem. U drugu ruku, njihovi ograničeni računalni resursi i memorija možda nisu prikladni za napredne operacije, poput autonomne navigacije visoke razine i obrade slike. Također, obično zahtijevaju dodatni hardver za povezivanje, kao što su Wi-Fi ili Bluetooth moduli (Lupashin et al., 2010). No kao što ćemo vidjeti ovo nije nužno pravilo, jer postoje specijalizirani mikrokontroleri s integriranim opcijama povezivosti.

5.1.2. Računala s jednom pločom (SBC)

Prednosti SBC-ova (eng. Single Board Computer) kao što su Raspberry Pi i BeagleBone je što posjeduju znatno veću računalnu snagu i memoriju, omogućujući složenije zadatke poput obrade slika, strojnog učenja i naprednih navigacijskih algoritama (Zufferey et al., 2013.). Dolaze s ugrađenim mrežnim mogućnostima i pokreću cijele operativne sustave, omogućujući

lakši razvoj i otklanjanje pogrešaka. Također imaju povećanu sposobnost da obrađuju više procesa istovremeno, što znači da možemo imati i zadatke poput logganja podataka, prijenosa telemetrijskih podataka, ili upravljanjem kompleksnim uređajima montiranima na dron (primjerice, LIDAR). Međutim, oni troše više energije i veći su od mikrokontrolera. Štoviše, pokretanje punog OS-a uvodi nedeterminističko ponašanje, što potencijalno dovodi do latencije kontrole (Zufferey et al., 2013.). Oni su također tipično i teži, što diže ukupnu masu drona, te su i skuplji za nabavku.

5.1.3.Hibridni sustavi

Hibridni sustavi iskorištavaju prednosti i mikrokontrolera i SBC-a. Koriste mikrokontroler za kontrolu u stvarnom vremenu i SBC za zadatke visoke razine. Primjeri uključuju kontroler leta Pixhawk s pratećim Raspberry Pi koji može izvršavati logiku leta u stvarnom vremenu, dok u isto vrijeme procesira visoko rezolucijske slike za mapiranje u stvarnom vremenu (Krajník et al., 2011). Ovi sustavi su obično složeniji i skuplji. Jedna od glavnih nedostataka za potrebe ovog rada je priroda ovakvih uređaja nije open-source, odnosno kod je relativno zatvoren i nije toliko jednostavna implementacija specifičnih funkcionalnosti. Dodatno, komunikacija i sinkronizacija između mikrokontrolera i SBC-a može biti izazovna za implementaciju i može dovesti do dodatnih točaka kvara (Krajník et al., 2011). Također, dodatak više komponenti povećava potrošnju struje.

Tablica 1. Prednosti i nedostatci različitih tipova FCU

FCU	Prednosti	Nedostatci
Mikrokontroler	Ekonomičnost, Niska potrošnja energije, niska latencija, rad u stvarnom vremenu, open-source podrška, velik broj modula	Ograničeni resursi, tipično zahtijevaju dodatne module za dodatne funkcionalnosti
SBC	Procesorska moć, ugrađene mogućnosti, paralelno procesiranje zadataka,	Potrošnja energije, veća latencija, cijena, težina
Hibridni	Rad u stvarnom vremenu, procesorska moć	Najviša cijena, najveća kompleksnost, zatvoren kod, potrošnja energije, težina

(Izvor: Izrada autora)

Na posljetku je odabran pristup s mikrokontrolerom iz nekoliko razloga. Prvenstveno zbog open-source prirode ovakvih kontrolera s dodatnom prednosti velikog broja modula na tržištu, osiguravaju najbolju podršku za implementaciju potpuno prilagođenih rješenja bez ograničenja zatvorenog koda. Naime, iako postoji broj hibridnih FCU-a koji implementiraju većinu potrebnih letnih funkcionalnosti, uglavnom su predviđeni za rad sa radio kontrolerom, te bi se samostalna

implementacija Android kontrolera mogla pokazati upitnom. Nadalje zbog male veličine drona potrebno bilo je potrebno osigurati malu masu i malu energetsku potrošnju kontrolera, kao i vrlo nisku latenciju procesiranja ulaza iz modula, mikrokontroler ostvaruje ove prednosti, te ovi zahtjevi ujedno i isključuju SBC kontrolere. Dron kojeg radimo u ovom radu ne izvršava nikakve kompleksne funkcionalnosti kao što bio slučaj u industrijskoj primjeni, pa nije bilo potrebe za velikom razinom procesorske moći što bi zahtijevalo ili hibridne ili SBC kontrolere leta, stoga nije postojao razlog za povećanjem kompleksnosti (u pogledu veće težine i veličine koji zahtijevaju veće baterije, motore i ostale komponente) i ostalih potencijalnih prepreka koje ove implementacije mogu imati.

Tradicionalno, jedna od prednosti mikrokontrolera je ujedno i mana, a to je da oni tipično imaju mali broj drugih komponenti već integriranih u sustav kao što je slučaj s SBC i hibridnim pristupima, što je značilo da će biti potrebno pronaći primjerene module pomoću kojih se može izgraditi cjelokupni elektronički sustav kvadkoptera. Jedna od najvećih prepreka je mrežna povezanost između kontrolera i daljinskog upravljača, u ovom slučaju putem Wi-Fi protokola. Naime, velik broj dostupnih modula koji omogućuju ovakvu komunikaciju je fizički relativno velikih dimenzija, uz veću potrošnju energije i težinu, što nas približava manama SBC-a.

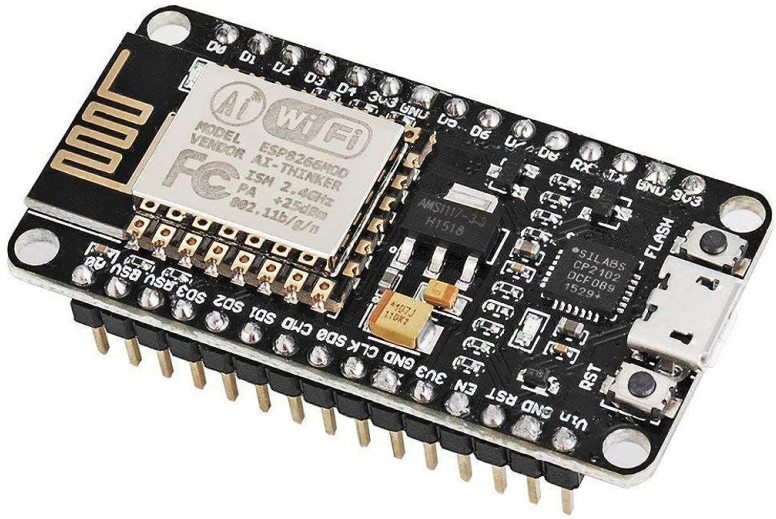
Jedno od rješenja koje je na kraju odabrano je uporaba ESP8266 Wi-Fi SoC-a (eng. *System on a Chip*). U idućem poglavlju opisati ćemo neke od karakteristike ovog mikrokontrolera.

5.1.4.ESP8266

ESP8266, koji je razvio Espressif Systems, je Wi-Fi čip s punim TCP/IP stackom i mogućnošću mikrokontrolera. Od svog pokretanja 2014., bio je sastavni dio demokratizacije razvoja IoT-a (eng. *Internet of Things*) zbog svoje pristupačnosti i značajki. Jedna od ključnih značajki ESP8266 je njegova Wi-Fi sposobnost, koja podržava 802.11 b/g/n protokole i funkcionira na 2.4 GHz. Može raditi u načinu rada stanice ili Soft Access Point i Wi-Fi Direct, omogućujući širok raspon mrežnih aplikacija (Espressif, 2023). ESP8266 uključuje nekoliko ulazno/izlaznih pinova opće namjene (GPIO), što omogućuje povezivanje sa senzorima, aktuatorima i drugim perifernim uređajima. Ovisno o specifičnom modelu, može imati do 17 GPIO pinova. Čip također pruža podršku za I2C, SPI i UART, koji su široko korišteni komunikacijski protokoli u ugrađenim sustavima(Espressif, 2023).

Modul ESP8266 često uključuje priključenu SPI flash memoriju. Veličina ove memorije varira s različitim modulima, a obično se kreće od 512 KB do 4 MB. Ova flash memorija prvenstveno se koristi za pohranjivanje korisničkog aplikacijskog programa i ključna je za OTA (Over-The-Air) nadogradnju firmvera (Espressif, 2023). Uz izvorni SDK koji nudi Espressif, postoji više razvojnih platformi dostupnih za programiranje ESP8266. ESP8266 se ističe svojom energetskom učinkovitošću. Može se pohvaliti s nekoliko načina rada u rasponu od aktivnog načina do načina dubokog mirovanja, što mu omogućuje upotrebu u aplikacijama kritičnim za

napajanje. Način dubokog mirovanja posebno je istaknuta značajka gdje se čip može staviti u stanje niske potrošnje kako bi se uštedjelo trajanje baterije, što je od ključne važnosti u mnogim IoT aplikacijama (Espressif, 2023).



Slika 5 Prikaz ESP8266 razvojne pločice (Izvor: Kolban, 2016.)

- Procesor: L106 32-bit RISC mikroprocesorska jezgra bazirana na Tensilica Diamond Standard 106Micro koja se pokreće na 80 ili 160 MHz.
- Memorija:
    - 32 KB instrukcijskog RAM-a
    - 32 KB instrukcijskog cache RAM-a
    - 80 KB korisničkog RAM-a
    - 16 KB ETS systemskog RAM-a
- Vanjski QSPI flash: do 16 MB podržano (512 KB do 4 MB tipično uključeno)
- IEEE 802.11 b/g/n Wi-Fi
    - Integriran TR prekidač (duplekser), balun, LNA, pojačalo snage i mreža uparene impedancije
    - WEP ili WPA/WPA2 autentikacija, ili otvorene mreže
- 17 GPIO pinova
- Serial Peripheral Interface Bus (SPI)
- I²C (softverska implementacija)
- I²S interface sa DMA (dijeljeni pinovi sa GPIO)
- UART na dediciranim pinovima, plus transmit-only UART se može uključiti na GPIO2
- 10-bitni ADC (Pretvarač iz analognih u digitalne vrijednosti)

Vrijedno je i spomenuti da su specifikacijama definirane i snage odašiljanja i primanja signala pomoću sva 3 podržana Wi-Fi standarda:

,

Tablica 2. Wi-Fi parametri ESP8266

Snaga odašiljanja	802.11 b: + 20 dBm
	802.11 g: + 17 dBm
	802.11 n: + 14 dBm
Snaga primanja	802.11 b: - 91 dbm (11 Mbps)
	802.11 g: - 75 dbm (54 Mbps)
	802.11 n: - 72 dbm (MCS7)

Izvor: (Espressif, 2023.)

Iz ovog vidimo da je najveći domet, odnosno snaga signala, postignuta sa protokolom b, iako je tu i znatno manja snaga primanja u praksi ovo neće biti problem zbog niske količine podataka koje se prenose.

U slučaju našeg kvadkoptera, ove fizičke specifikacije su i više nego dovoljne za implementaciju kontrole i stabilizacije u stvarnom vremenu, s obzirom da ne moramo raditi kompleksne procese poput obrada slika i sličnog. Moramo još i provjeriti koja programska okruženja su nam na raspolaganju, jer izbor okruženja ovisi i o brzini razvoja i jednostavnosti otklanjanja pogrešaka, što olakšava izradu prototipova aplikacija. Espressif je releasao njihov SDK za direktno programiranje njihovog SoC-a bez korištenja dodatnog serial mikrokontrolera u listopadu 2014 godine. Od tad, na tržište je izašlo još mnogo drugih SDK-a, neki od kojih su bazirani na C++, Lua-i, i JavaScript-u.

Pogledajmo neka od okruženja koji su nam dostupni za implementaciju kvadkoptera.

- Arduino — firmver temeljen na C++. Ovako, ESP8266 CPU i njegove Wi-Fi komponente mogu se programirati kao bilo koji drugi Arduino uređaj. ESP8266 Arduino Core dostupan je putem GitHuba.
- ESP8266 BASIC — Interpreter otvorenog koda sličan BASIC-u posebno prilagođen za Internet stvari (IoT). Samostalno hostano razvojno okruženje sa pristupom kroz preglednik.
- ESPHome — Sustav za kontrolu ESP8266/ESP32 jednostavnim, ali moćnim konfiguracijskim datotekama i upravljanje njima na daljinu putem sustava kućne automatizacije.
- NodeMCU — firmver temeljen na Lua.
- uLisp — verzija Lisp programskog jezika posebno dizajnirana za rad na procesorima s ograničenom količinom RAM-a.
- ZBasic za ESP8266 — Izveden iz Microsoftovog široko korištenog Visual Basica 6, koji je prilagođen kao kontrolni jezik za ZX obitelj mikrokontrolera i ESP8266.

- Zerynth — IoT okvir za programiranje ESP8266 i drugih mikrokontrolera u Pythonu.
- ESP-Open-RTOS — ESP8266 softverski okvir otvorenog koda temeljen na FreeRTOS-u.
- ESP-Open-SDK — Besplatan i otvoren (što je više moguće) integrirani SDK za ESP8266/ESP8285 čipove.
- MicroPython — Priključak MicroPythona (implementacija Pythona za ugrađene uređaje) za platformu ESP8266.
- Mongoose OS — operativni sustav otvorenog koda za ESP8266 i ESP32. Razvijanje u C-u ili JavaScriptu.

Na osnovi visoke razine podrške i popularnosti, krug potencijalnih SDK-a je sužen na NodeMCU i Arduino. Također, neke od opcija nisu dovoljno performantne, niti zamišljene za primjene u stvarnom vremenu s vrlo niskom latencijom odaziva. Kao balans svih zahtjeva, na kraju je odabran Arduino SDK okruženje bazirano na C++ jeziku, s kojim je ujedno i autor upoznatiji od nekih ostalih opcija.

U rezimeu, ESP8266 nudi značajke koje ga čine robusnim i svestranim alatom za IoT razvoj, ali i za razvoj kvadkoptera. Njegova ugrađena Wi-Fi mogućnost, fleksibilne GPIO opcije, učinkovita potrošnja energije i izdašna podrška za razvoj softvera samo su neke od značajki koje ga čine dobrim izborom za ovakve primjene, pogotovo među studentima, hobistima ali i profesionalcima. Niska cijena i potrošnja energije ESP8266, zajedno s njegovim mrežnim mogućnostima, učinili su ga popularnim za IoT aplikacije kao što su pametne kuće, industrijska automatizacija i nadzor okoliša. Njegova kompatibilnost s razvojnim platformama poput Arduina omogućuje brzu izradu prototipova, što ga čini prikladnim za hobiste i obrazovne svrhe. Njegova pristupačnost demokratizirala je razvoj povezanih uređaja, što je dovelo do porasta inovacija na lokalnom nivou (Kolban, 2016.).

5.2. Izbor motora

U dizajnu kvadkoptera, odabir odgovarajućeg tipa motora igra značajnu ulogu u određivanju učinkovitosti, manevarskih sposobnosti i ukupnih performansi drona. Razumijevanje ovih prednosti i nedostataka omogućuje izbor prilagođenijih motora za određenu uporabu. Postoje dvije glavne vrste motora koje se koriste u izgradnji kvadkoptera: Istosmjerni motori sa četkicama i bez četkica.

5.2.1. Istosmjerni motori s četkicom

Istosmjerni motori s četkicom jednostavni su za upravljanje jer im je potrebna samo istosmjerna voltaža za kontrolu brzine. Ova jednostavnost smanjuje zahtjeve za sofisticiranom elektronikom, čime se smanjuje ukupna cijena kvadkoptera. Jedna od prednosti je također i

njihova dostupnost u manjim veličinama. Glavni nedostatci ovih motora leže u manjoj efikasnosti za istu veličinu zbog trenja četkica, kao i u manjem omjeru okretnog momenta naspram težine (Bouabdallah, 2007.). Još jedan od dugoročnih problema je i njihov kraći životni vijek, prvenstveno zbog trošenja četkica.

5.2.2.Motori bez jezgre (eng. coreless)

Kada su izašli, motori bez jezgre su predstavljali znatni napredak u tehnologiji elektromotora. Ove motore karakterizira nepostojanje željezne jezgre u namotima, što dovodi do jedinstvenog niza prednosti i nedostataka. Motori bez jezgre poznati su po svojoj visokoj gustoći snage i učinkovitosti. Nepostojanje željezne jezgre smanjuje magnetske gubitke, što dovodi do poboljšanih performansi (Lawhorn et al., 2021.). Također je utvrđeno da motori bez jezgre imaju prednosti u smislu odvođenja topline. Odsutnost željezne jezgre omogućuje bolje hlađenje, što povećava pouzdanost i životni vijek motora (Lawhorn et al., 2021.). Nažalost ovi motori imaju i svoje nedostatke, primjerice proces proizvodnje motora bez jezgre može biti složeniji u usporedbi s tradicionalnim motorima. Nepostojanje jezgre zahtijeva precizne tehnike namotavanja, što može povećati proizvodne troškove (Lawhorn et al., 2021.). Također, unatoč svom potencijalu, motori bez jezgre relativno su novi u smislu tehnologije elektromotora. Do sad je proveden ograničen broj istraživanja, te su se većina fokusirala u primjenama poput pogona električnih zrakoplova, što bi moglo spriječiti njihovu široku primjenu (Lawhorn et al., 2021.).

5.2.3.Istosmjerni motori bez četkica

S druge strane motori bez četkica imaju većinu suprotnih karakteristika, s duljim životom i boljom efikasnosti. Često ovakvi motori i imaju bolje karakteristike rasipanja topline, što ih čini prikladnijim za uporabu u neprekidnom režimu rada. Mane leže u njihovoj tipično većoj veličini, kompleksnijim zahtjevima elektronike, i višoj cijeni (Bouabdallah, 2007.).

Tablica 3. Prednosti i nedostatci motora korištenih za kvadkoptere

Tip motora	Prednosti	Nedostatci
Istosmjerni motor s četkicama	Niža cijena, jednostavnija kontrola, dostupnost u manjim veličinama	Kraći život, niža efikasnost, manji omjer snage i težine
Istosmjerni motor bez četkica	Duži život, veća efikasnost, bolji omjer okretnog momenta naspram težine, rasipanje topline	Viša cijena, Kompleksnija kontrola
Motor bez jezgre	Omjer snage i težine, energetska učinkovitost, rasipanje topline, jednostavna kontrola, dostupnost u manjim veličinama	Kompleksnija proizvodnja, viša cijena, relativna neistraženost, nisu dostupni u većim veličinama

(Izvor: Izrada autora)

Generalno s četkicom bili su popularni su u dronovima igračkama zbog svoje niske težine i manjku potrebe za kompleksnom popratnom elektronikom potrebnom za motore bez četkica, što doprinosi nižoj ukupnoj težini i cijeni drona. S druge strane, motori bez četkica često se koriste za potrebe trkaćih dronova kojima je bitan omjer snage i težine, kao i dronovima koji moraju nositi veći teret, ili raditi duže vremena neprekidno. Dobra kombinacija obojih karakteristika potencijalno se nalazi u motorima bez jezgre, koji održavaju jednostavnost kontrole, nisku težinu i cijenu motora s četkicama, dok u isto vrijeme inoviraju i postižu visok omjer snage i težine, efikasnost, i rasipanje topline.

Iz ovih razloga trenutno na tržištu dominiraju u kategoriji mikro-dronova, sa svojim balansom poželjnih karakteristika. Odabrana je veličina „1020“, odnosno 10mm dijametar s 20mm visinom, što je jedan od najvećih dostupnih veličina za motore bez jezgre koji se primjenjuju u dronovima.

5.3. Izbor jedinice za inercijsko mjerenje (IMU)

Jedinice za inercijalno mjerenje osnova su moderne tehnologije kvadkoptera, pružajući potrebne podatke za stabilizaciju, navigaciju i sustave upravljanja (Martin & Salaun, 2010.). Stabilnost i upravljivost kvadkoptera uvelike se oslanjaju na podatke u stvarnom vremenu koje pruža ugrađeni IMU. IMU mjeri linearna ubrzanja i kutne brzine kvadkoptera, koje zatim koristi kontroler leta za prilagodbu brzina pojedinačnih motora i održavanje stabilnog leta

5.3.1. IMU-ovi temeljeni na MEMS-u

MEMS IMU su kompaktni, integrirani uređaji koji pomoću akcelerometra i žiroskopa mjere linearno i kutno gibanje. Jedna od ključnih prednosti MEMS IMU je njihova kompaktna veličina i mala težina, što ih čini idealnim za primjene u kojima su volumen i masa kritični faktori, kao što su mobilni uređaji i dronovi (Mishra et al., 2019.). Nadalje, MEMS IMU poznati su po niskoj potrošnji energije, što je ključni čimbenik kod uređaja koji se napajaju baterijama. Još jedna značajna prednost MEMS IMU-a je njihova isplativost. U usporedbi s tradicionalnim IMU-ima, MEMS IMU-i znatno su jeftiniji za proizvodnju i kupnju, što ih čini prikladnom opcijom za širok raspon primjena.

Unatoč brojnim prednostima, MEMS IMU nisu bez nedostataka. Jedan od primarnih nedostataka MEMS IMU je njihova podložnost pogreškama zbog promjena temperature. To posebno vrijedi za žiroskope, koji mogu poprimiti značajno smanjenje performansi kada su

izloženi temperaturnim varijacijama (Yang et al., 2018). Drugi izazov povezan s MEMS IMU-ima je prisutnost intrinzičnih grešaka senzora, kao što su bias, drift i šum. Te pogreške su rezultat fizičke konstrukcije samog uređaja, koji je mehanički u prirodi. Te pogreške mogu značajno utjecati na točnost mjerenja i zahtijevaju napredne tehnike kalibracije za ublažavanje. Nedavna istraživanja usmjerena su na razvoj naprednih algoritama i tehnika kalibracije za ublažavanje ovih pogrešaka i poboljšanje performansi MEMS IMU (Yang et al., 2018).

5.3.2.Žiroskopi s optičkim vlaknima (FOG) IMU

Iako nisu uobičajeni u potrošačkim kvadkopterima zbog svoje visoke cijene, žiroskopi s optičkim vlaknima (FOG) nude vrhunske performanse i koriste se u vrhunskim uređajima profesionalne kvalitete. Oni iskorištavaju interferenciju svjetlosnih valova za mjerenje promjena u rotaciji. FOG IMU nudi nekoliko prednosti u odnosu na druge vrste. Jedna od ključnih prednosti je njihova visoka osjetljivost i točnost, što ih čini idealnim za aplikacije koje zahtijevaju izrazito precizno praćenje pokreta (Brossard & Bonnabel, 2019.). Nadalje, FOG IMU-i su imuni na elektromagnetske smetnje, što je čest problem s drugim vrstama IMU-a. Još jedna značajna prednost FOG IMU je njihova robusnost. Za razliku od MEMS IMU-a, FOG IMU-i ne mijenjaju svoje karakteristike pod značajnim promjenama temperature, što ih čini pouzdanijima u različitim okolinskim uvjetima.

No i FOG IMU-ovi imaju svoje nedostatke. Jedan od glavnih nedostataka je njihov trošak. Proizvodnja FOG IMU uključuje kompleksne proizvodne procese, što ih čini znatno skupljima od ostalih vrsta IMU. Još jedan izazov je njihova fizička veličina. Iako su manji od tradicionalnih mehaničkih žiroskopa, još uvijek su veći od MEMS IMU-ova, što može biti limitirajući čimbenik u primjenama u kojima je prostor vrlo ograničen (Su et al., 2017.).

5.3.3.Hibridni IMU

Nijedan IMU nije savršen, svaki tip senzora koji se koristi za IMU ima svoje prednosti i mane. Ideja hibridnih IMU-a je da se kombinacijom različitih vrsta senzora postignu snage a izbjegnu nedostatci pojedinih senzora. Primjerice, postoje algoritmi spajanja senzora, koji se koriste za kombiniranje podataka iz akcelerometara, žiroskopa i magnetometra, kako bi se izračunali točniji podaci o orijentaciji. Postoje i hibridni IMU koji kombiniraj klasični IMU, zajedno s lokalizacijom temeljenom na kameri, kako bi pružili točnije pozicioniranje u zatvorenom prostoru od bilo kojeg sustava pojedinačno (Poulose & Han, 2019.). Još jedna značajna prednost hibridnih IMU je njihova otpornost na promjene okoliša. Iskorištavanjem višestrukih senzora, hibridni IMU-ovi se mogu prilagoditi promjenama u okolišu i održavati točno praćenje kretanja čak i pod izazovnim uvjetima (Zhao et al., 2021.). Hibridni IMU imaju velik potencijal

u specijaliziranim primjenama, gdje su poznata ograničenja te uvjeti u kojima će uređaj raditi, za koje se može stvoriti sistem s idealnim karakteristikama.

Unatoč svojim prednostima, hibridni IMU-ovi imaju i svoje nedostatke. Jedan od glavnih nedostataka je njihova složenost. Integracija više senzora zahtijeva složene algoritme za integraciju senzora, što može povećati procesorske zahtjeve sustava (Zhao et al., 2021.). Drugi izazov povezan s hibridnim IMU-ima je potreba za kalibracijom i sinkronizacijom između različitih modaliteta senzora. To može biti složen proces i može zahtijevati dodatna hardverska ili softverska rješenja (Shahzad et al., 2019.). Nadalje, potreba za integracijom više senzora znatno doprinosi ukupnom volumenu i težini rješenja.

Tablica 4. Prednosti i nedostatci različitih tipova IMU

IMU	Prednosti	Nedostatci
MEMS	Veličina, Težina, Potrošnja energije, ekonomičnost	Osjetljivost na temperaturu, bias, drift, šum
FOG	Performanse, stabilnost, robusnost	Potrošnja energije, cijena, veličina, težina
Hibridni	Performanse, specijaliziranost	Kompleksnost, cijena, veličina, težina

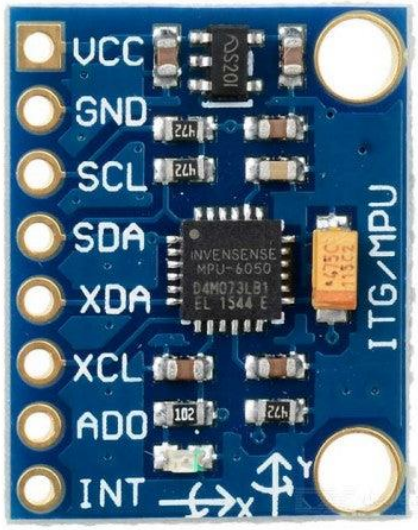
Table 1. Izvor: Izrada autora

Izbor IMU za aplikacije kvadkoptera prvenstveno ovisi o namjeravanoj primjeni i proračunu. IMU-ovi koji se temelje na MEMS-u kao što su MPU-6050 ili MPU-9250 dovoljni su za većinu aplikacija. Za profesionalne primjene koje zahtijevaju visoku preciznost i pouzdanost, mogu se koristiti naprednija rješenja poput IMU-a temeljenog na FOG-u, uz penalitet cijene i dodatne kompleksnosti ukupnog sustava. Za određene specijalizirane potrebe u kojima su potrebe visoke performanse, ili izrada kompleksnog ponašanja, kombinacija senzora kroz hibridni pristup može biti primjerena. S obzirom da naš dron spada među jednostavnije, sa svojim ograničenjima u veličini i dostupnoj količini baterije, MEMS uređaj bi bio najprimjereniji. Jedan od popularnih modela korištenih široko u industriji je MPU-6050, koji je dostupan i kao modul spreman za spajanje sa mikrokontrolerom preko I2C protokola, te je stoga odabran. Pogledajmo pobliže specifikacije ovog modula.

5.3.4.MPU-6050

MPU-6050 je široko korišten modul koji integrira 3-osni žiroskop i 3-osni akcelerometar na jednom čipu. Ovakav tip senzorskog modula, koji proizvodi InvenSense, se može kategorizirati kao inercijalno mjerna jedinica (IMU) i obično se koristi u raznim aplikacijama, uključujući robotiku i praćenje kretanja (Rafiq et al., 2020).Raspon vrijednosti za žiroskop može se postaviti na ±250, ±500, ±1000 ili ±2000 stupnjeva u sekundi, dok se akcelerometar može konfigurirati na ±2, ±4, ±8 ili ±16 g (InvenSense , 2023). 3-osni akcelerometar unutar MPU-

6050 radi na principu kapacitivnosti. Promjene u ubrzanju uzrokuju promjenu u kapacitetu, koji se prevodi u podatke o ubrzanju duž odgovarajućih osi zahvaljujući analogno digitalnom konverteru (eng. *ADC*). Žiroskop s 3 osi radi na principu Coriolisovog efekta. Senzor mjeri kutnu brzinu otkrivanjem Coriolisove sile na vibrirajućem elementu. Ovo omogućuje šest stupnjeva slobode (6-DOF) mjerenjem rotacijskog gibanja (okretanje, nagib i skretanje) i translacijskog gibanja (x, y i z ubrzanja). Digitalna obrada pokreta (DMP) je bitna značajka u MPU-6050, gdje se složeni proračuni kao što je spajanje senzora mogu prenijeti s mikrokontrolera na DMP sistem unutar uređaja-a (InvenSense, 2023). Ova je mogućnost korisna u aplikacijama koje zahtijevaju praćenje kretanja u stvarnom vremenu, kao što je u slučaju s dronom.



Slika 6: Prikaz MPU-6050 modula (Izvor: InvenSense, 2023.)

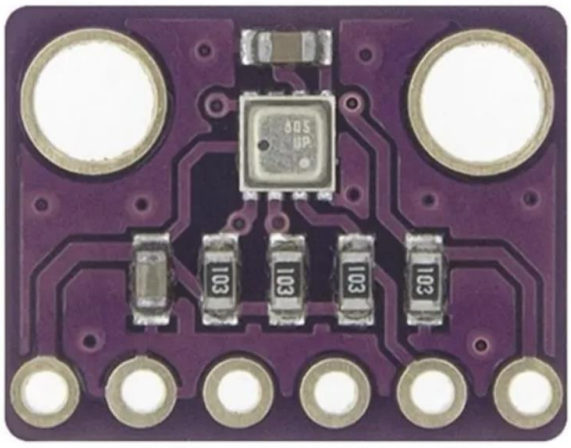
MPU-6050 podržava i I2C i serijske komunikacijske protokole, olakšavajući njegovu integraciju s različitim platformama mikrokontrolera. Također uključuje značajke upravljanja napajanjem koje mu omogućuju upotrebu u aplikacijama niske potrošnje. S različitim načinima rada, uključujući način mirovanja niske potrošnje energije, nudi fleksibilnost u balansiranju performansi i potrošnje energije (InvenSense, 2023.).

Jedan od uobičajenih problema sa MEMS uređajima koje smo spomenuli je fenomen drift-a, koji rezultira odstupanjem između stvarnih uvjeta i očitanja senzora. To je uzrokovano akumuliranjem pogrešaka koje proizlaze iz izračunavanja integrala ubrzanja. Međutim, metode kao što su početna kalibracija sustava i spajanje senzora uvelike pomažu rješavanju pogreška pomaka (Rafiq et al., 2020). Još jedna tehnologija ugrađena u MPU6050 može pomoći s ovim, a to je DMP. DMP ili Digital Motion Processor prima sirove podatke s akcelerometra i žiroskopa te ih obrađuje kako bi se smanjila buka i poboljšala točnost. DMP može kombinirati podatke s

različitih senzora kako bi se dobila preciznija slika pokreta i orijentacije. Na primjer, kombiniranjem podataka s akcelerometra i žiroskopa može se dobiti stabilnija i točnija procjena orijentacije. DMP također može konvertirati sirove podatke u kvaternion format, koji je matematički prikladan za predstavljanje 3D rotacija. Budući da DMP obavlja složene izračune unutar samog čipa, glavni mikrokontroler (kao što je ESP8266) može biti oslobođen tih zadataka, što dovodi do uštede energije i resursa. Sve ove karakteristike znače da je ovaj uređaj primjeren za uporabu u dronovima, što se pokazalo istinom i u industriji.

5.4. Barometar

Moderni dronovi često uključuju senzore za okoliš kako bi optimizirali karakteristike leta u različitim atmosferskim uvjetima. Među njima je i BME280 koji sadrži digitalni senzor tlaka, vlage i temperature u jednom kompaktnom paketu, što ga čini vrijednim alatom za mnoge primjene, uključujući dizajn kvadkoptera (Bosch Sensortec, 2020.). Iako nije striktno potreban za ostvarivanje leta, za nas je barometar koristan modul za dron jer pomoću njega možemo odrediti i nadmorsku visinu, i trenutnu visinu drona od zemlje. Napredni moderni dronovi mogu i značajno poboljšati letne karakteristike i sigurnost. Primjerice, promjene tlaka i temperature mogu utjecati na gustoću zraka, što utječe na stvaranje uzgona (Beard & McLain, 2012). Uključivanjem senzora BME280, kvadkopteri mogu izvršiti potrebne prilagodbe u svojim kontrolnim algoritmima za poboljšanu stabilnost i učinkovitost.
BME280 je potpuno integrirani MEMS senzor za okoliš koji može mjeriti atmosferski tlak u rasponu od 300 do 1100 hPa, temperaturu od -40 do +85 °C i relativnu vlažnost od 0% do 100% (Bosch Sensortec, 2020.).



Slika 7. Prikaz BME280 modula (Izvor: Amazon.com)

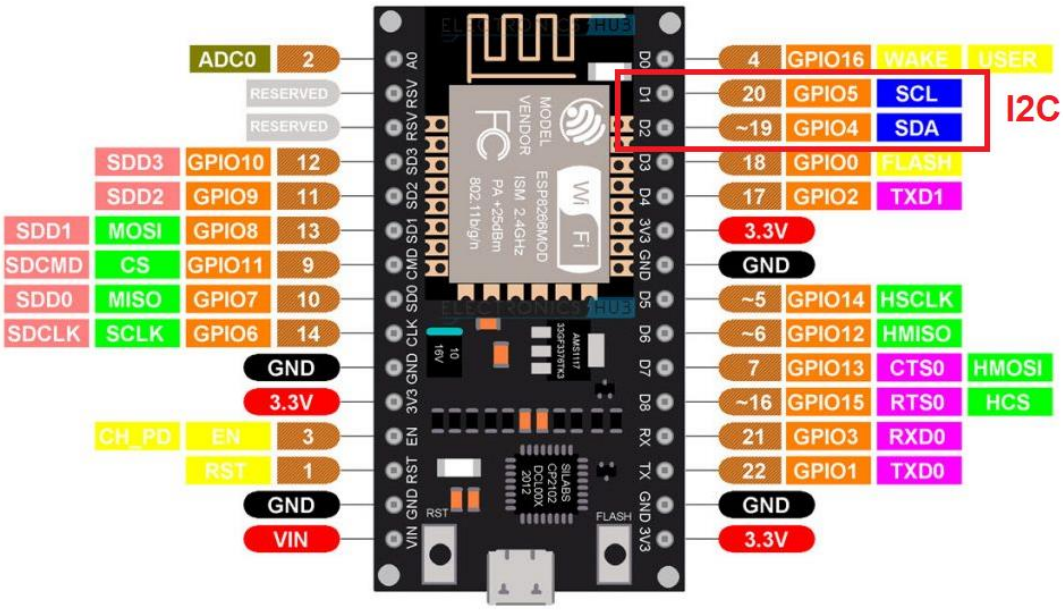
Barometarski senzor u BME280 radi na principu piezootpora. Varijacije atmosferskog tlaka uzrokuju deformaciju elementa senzora, što dovodi do promjene otpora, koji se zatim mjeri i prevodi u podatke o tlaku. Senzor temperature radi na principu otporne detekcije temperature, gdje se promjena otpora materijala zbog varijacija temperature mjeri i pretvara u podatke Senzor vlažnosti radi na principu kapacitivnosti, gdje promjene u relativnoj vlažnosti mijenjaju dielektričnu konstantu polimernog materijala, što dovodi do promjene u kapacitetu. Ta se promjena mjeri i pretvara u podatke o vlažnosti (Bosch Sensortec, 2020.).
BME280 ima i I2C i SPI sučelja, što ga čini kompatibilnim sa širokim spektrom mikrokontrolera i drugih digitalnih sustava. Ova fleksibilnost u komunikacijskim protokolima omogućuje integraciju s različitim hardverom uključujući i našim ESP8266. Također se ističe svojom kompaktnošću, s dimenzijama od samo 2,5 x 2,5 x 0,93 mm. Ovaj kompaktni dizajn olakšava njegovu integraciju u prostorno ograničene aplikacije, poput vremenskih stanica i malih bespilotnih letjelica (Bosch Sensortec, 2020.).

6. Komunikacijski protokoli

Prije same izrade drona i programiranja mikrokontrolera, objasniti ćemo komunikacijske tehnologije koji će omogućiti interoperabilnost svih dijelova sustava. Primjerice, već smo spomenuli da i naš mikrokontroler i IMU oboje podržavaju I2C protokol pomoću kojeg ćemo ih implementirati, ali što je I2C? Nadalje, tu je i pitanje protokola kojim će naš dron komunicirati s jedinicom za upravljanje (Wi-Fi), i iako postoji veće intuitivno razumijevanje zbog prolifičnosti ovog standarda, mnoge pojedinosti nisu poznate.

6.1. I2C

Protokol Inter-Integrated Circuit (I2C), kojeg je razvio Philips Semiconductor (sada NXP Semiconductors) 1982 godine, je jednostavan, dvosmjerni i sinkroni komunikacijski protokol za povezivanje perifernih uređaja niske brzine s mikrokontrolerom ili mikroprocesorom (NXP Semiconductors, 2021.). Omogućuje povezivanje u controller/target konfiguracijama gdje se specificira koji uređaj je upravljač, odnosno svojevrsni hub komunikacije ukoliko je prisutno više I2C uređaja. Moguće je korištenje više uređaja na istoj liniji, jer svaki uređaj može biti na drugoj adresi, koja je konfigurabilna. Na ESP8266, I2C protokol se često koristi za komunikaciju s perifernim uređajima kao što su senzori, zasloni ili drugi mikrokontroleri. Na sljedećoj slici nalaze se ESP8266 I2C GPIO konektori označeni na slici.

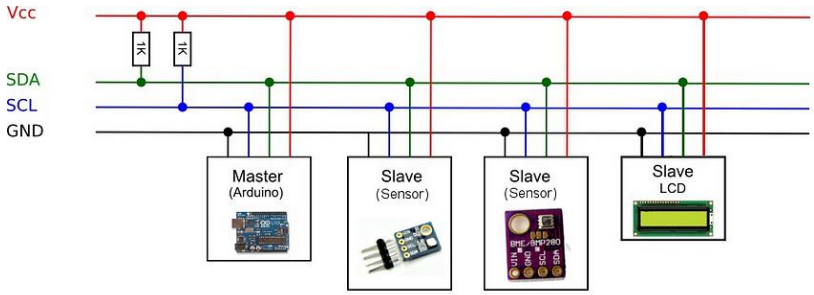


Slika 8. I2C pinovi na ESP8266 (Izvor: ElectronicsHub.com)

Sami podatci su poslani u paketima, na nesimetričnoj liniji, što znači da nema diferencijalno signaliranje. U praktičnom smislu, to znači da je I2C ograničen na kratke dužine i niske brzine prije nego integritet signala degradira. Protokol I2C široko je prihvaćen zbog svoje jednostavnosti, mogućnosti podrške za više uređaja sa samo dvije žice i ugrađene funkcije provjere pogrešaka (NXP Semiconductors, 2021.)..

Dva signalna voda su SCL i SDA:

- SCL (eng. serial data), služi za primanje i slanje podataka serijski
- SDA (eng. serial clock), služi za slanje i primanje frekvencije kristalnog oscilatora



Slika 9. Shematski prikaz I2C sabirnice (Izvor: Medium.com, 2022)

Uglavnom je korišten u uređajima gdje je jednostavnost i niska cijena proizvodnje važniji od brzine. U ovom slučaju, količina podataka i udaljenosti na kojima se prenose je toliko mala, da ne postoji ni potreba za kompleksnijim protokolom.

6.2. Wi-Fi

Wi-Fi je jedan od protokola s kojim smo svi upoznati u dnevnom životu, te igra ključnu ulogu u modernoj eri povezanih uređaja. Njegova prolifičnost ujedno čini i jednu od njegovih najvećih snaga, a to je da skoro svako posjeduje Wi-Fi uređaj u svom džepu. Temeljen na standardu IEEE 802.11, ističe se svojom širokom primjenom i mogućnostima prijenosa podataka velikom brzinom. Pojam Wi-Fi označava "Wireless Fidelity", odražavajući njegov dizajn kao bežični pandan Ethernetu (IEEE, 2020).

Na temelju standarda IEEE 802.11, Wi-Fi podržava različite propusnosti i frekvencije, nudeći mnoštvo brzina prijenosa podataka ovisno o specifičnom implementiranom standardu (npr. 802.11a, 802.11b, 802.11g, 802.11n, 802.11ac, itd. Konkretno ESP8266 podržava 802.11b/g/n protokole pokrivajući frekvencijski raspon od 2,4 GHz do 2,5 GHz.

- 802.11b je predstavljen 1999. godine i često se naziva Wi-Fi 1, donio je značajan napredak u bežičnoj komunikaciji kroz frekvencijski spektar od 2,4 GHz do 11 Mbps. Jedna od glavnih prednosti 802.11b bila je njegova sposobnost da ponudi relativno dobar domet, što ga čini posebno korisnim za postavke kućne mreže. Međutim, njegova upotreba 2.4 Frekvencija GHz učinila ga je osjetljivim na smetnje od drugih uređaja koji rade unutar istog spektra, kao što su bežični telefoni, mikrovalne pećnice i druga kućanska elektronika.
- 802.11g poznatiji kao Wi-Fi 3, definiran je 2003. godine i također je radio unutar frekvencijskog raspona od 2,4 GHz, nudeći brzine do 54 Mbps. Ovo poboljšanje postignuto je usvojivši tehniku modulacije ortogonalnog frekvencijskog multipleksiranja (OFDM), koja je bila učinkovitija i otpornija na smetnje. S obzirom na to da je radila na istoj frekvenciji od 2,4 GHz kao 802.11b, zadržala je kompatibilnost s prethodnim verzijama.
- 802.11n je ratificiran 2009. i označen kao Wi-Fi 4, 802.11n je predstavljao značajan napredak u bežičnom umrežavanju, budući da je podržavao frekvencijske pojaseve od 2,4 GHz i 5 GHz, iako su mnogi uređaji primarno koristili spektar od 2,4 GHz. Uvođenjem tehnologije višestrukog ulaza, višestrukog izlaza (MIMO), 802.11n može koristiti više antena i za prijenos i za prijem, drastično poboljšavajući brzine prijenosa podataka i pouzdanost signala.

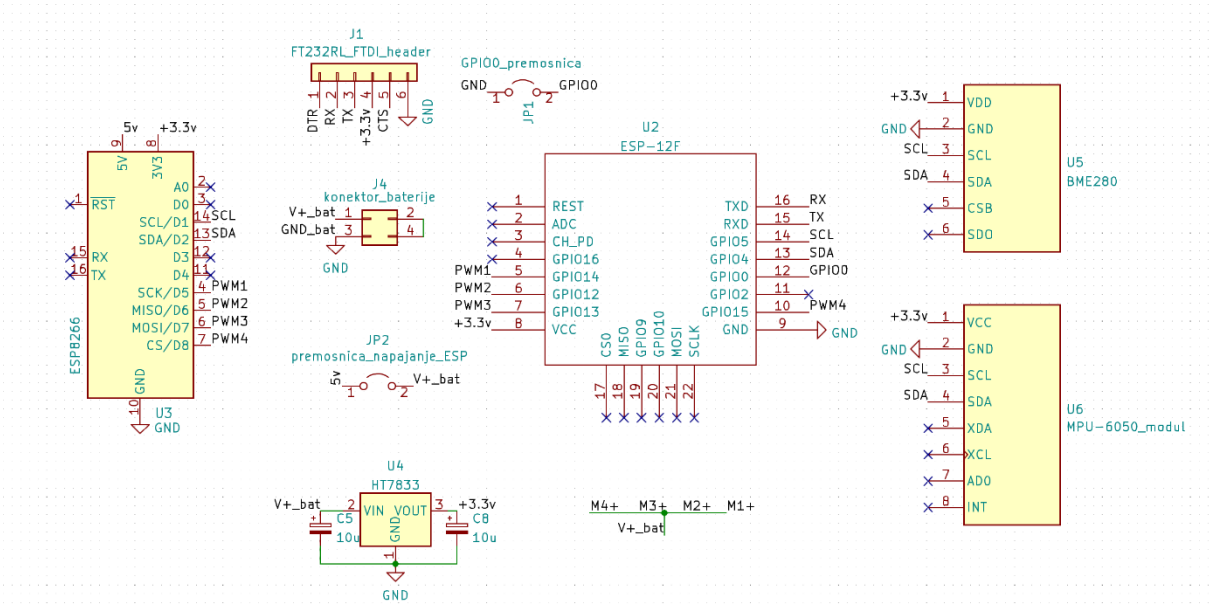
Kao što smo spomenuli može raditi u različitim načinima rada kao što je način rada stanice (STA), način rada Access Point-a (SoftAP) ili oba u isto vrijeme (STA+SoftAP način rada), nudeći visok stupanj fleksibilnosti u bežičnoj komunikaciji (Espressif, 2023.). Ideja povezivanja drona putem Wi-Fi-ja već postoji u literaturi, primjerice Qin et al. 2017, koji predlaže mogućnost spajanja većeg broja dronova na baznu stanicu za zemlji, što omogućuje

kompleksno ponašanje rojeva dronova i jednostavniju skalabilnost nego povezivanjem svakog drona na radio kontroler. Velika brzina Wi-Fi-ja omogućuje i jednostavnije implementiranje prijenosa većeg broja podataka baznoj stanici, poput slika ili drugih letnih parametara (Qin et al., 2017.). Štoviše, korištenje Wi-Fi-ja omogućuje dronovima da iskoriste postojeće mrežne infrastrukture. Na primjer, u urbanim sredinama, dronovi bi se mogli povezati s javnim Wi-Fi mrežama kako bi poboljšali svoj komunikacijski domet ili smanjili opterećenje svojih komunikacijskih sustava na brodu (Alazab et al., 2020.).

7. Izrada drona

7.1. Elektronički dizajn

Moglo bi se reći da tiskana pločica, odnosno PCB (eng. *Printed Circuit Board*), predstavlja središnji živčani sustav modernih dronova, međusobno povezanih senzora, motora i upravljačkih sustava. Standardni PCB-ovi možda neće uvijek zadovoljiti specifične zahtjeve naprednih aplikacija bespilotnih letjelica, kao što su posebni senzori, prilagođeni kontroleri motora ili faktori kompaktnog oblika. U idealnom slučaju, PCB-ovi ne samo da moraju omogućiti besprijekornu integraciju, već bi trebali ponuditi i modularnost za prilagođavanje budućih nadogradnji ili specifičnih zahtjeva misije. U ovom slučaju, za danu kombinaciju odabranog kontrolera i ostalih komponenti, nije pronađena prikladna tiskana pločica za izgradnju drona. Moguće bi bilo koristiti tzv. „protoboard“ za izradu prototipa, ali dizajniranje prilagođenog PCB-a omogućuje optimizaciju težine, veličine i parametara izvedbe. Svaka tiskana pločica prvo počinje sa shematskom reprezentacijom.



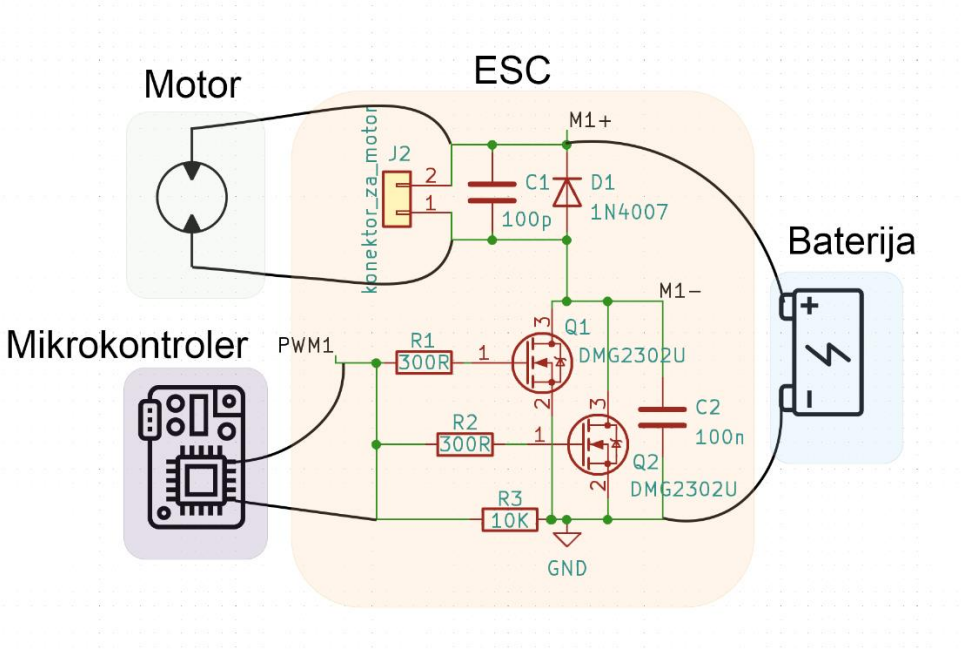
Slika 10. Shematski prikaz glavnih komponenti drona (Izvor: Izrada autora)

Implementirana su 2 različita modula za ESP8266, jedan koji se nalazi na svojoj pločici (WeMos D1 mini Pro) koji već implementira sve regulatore i CP2104 serijalni pretvarač za programiranje pločice, a drugi koji sadrži samo implementaciju ESP8266 (ESP-12F) i zahtjeva dodatne komponente. Ovo je napravljeno iz razloga težine i praktičnosti. Naime, u ranim faza prototipiranja i razvoja znatno je jednostavnije raditi sa pločicom, ali kada je većina rada napravljena ESP-12F bi pružio manje mase, i moduli bi se mogli rotirati prema unutra, što bi dodatno dalo kompaktniji centar gravitacije. Konektor za bateriju je napravljen tako da može primiti do dvije baterije paralelno, što povećava ukupni kapacitet i vrijeme leta drona.
Energetska učinkovitost ključna je za bespilotne letjelice, koje često rade na ograničenim kapacitetima baterija. Bitna je učinkovitost sheme i krugovi za distribuciju energije, komponente s niskom potrošnjom energije u stanju pripravnosti i učinkoviti mehanizmi za regulaciju napona. Zbog toga je odabran HT7833 linearni regulator s niskim padom voltaže (LDO), što znači da ga je moguće koristiti kada su ulazne i izlazne voltaže blizu jedna druge, primjerice kada nam je ulazna voltaža 5V, a izlazna 3.3V. Ima izrazito nisku potrošnju energije u pripravnosti od samo 4 mikroampera te vrlo niski šum. Odabran kao napajanje za ESP-12F modul, BME280 i MPU-6050 kako bi oni mogli postići najbolje performanse. Ovo napajanje se može odabrati koristeći premosnicu JP2. Ukoliko se koristi ESP-12F mora se koristiti zasebni FTDI programer za programiranje EEPROM memorije, koji se spaja na J1, i nakon programiranja može odspojiti za smanjenje težine drona. On se koristi jer sam ESP-12F nema nikakvo sučelje s USB-om, stoga je potreban takav pretvarač iz USB-a u TTL serijski protokol. Kako bi se uključio mod programiranja, potrebno je GPIO0 konektor spojiti s zemljom, za što služi premosnica JP1.

7.1.1. Izgradnja ESC-a

U općenitom smislu, elektronički regulatori brzine (ESC) su bitne komponente u raznim elektromehaničkim sustavima koji zahtijevaju kontrolu brzine elektromotora. On tumači ulazne signale iz mikrokontrolera (često PWM signale) i sukladno tome modulira snagu isporučenu motoru. Tipični ESC-ovi sastoje se od mikrokontrolera, tranzistora snage, i popratnih komponenti za sigurnost, mikrokontroler pruža ulazne signale, dok upravljački tranzistori snage na bazi toga moduliraju snago koja prolazi kroz motor. ESC-ovi za motore s četkicama prvenstveno funkcioniraju modulirajući napon koji se dovodi u motor, čime kontroliraju njegovu brzinu. Za razliku od svojih parnjaka bez četkica, ESC s četkastim motorom imaju jednostavniji kontrolni mehanizam, često ne zahtijevajući složeno prebacivanje faza. Kontrola brzine postiže se putem modulacije širine impulsa, koja mijenja efektivni napon motora (Hammoodi et al.,

2020.). U primjenama sa motorima s četkicama učinkovitost i pouzdanost ESC-a su najvažniji. Trošenje četkica, uzrokovano trenjem između četkica i komutatora, često dovodi do smanjene učinkovitosti i vijeka trajanja stoga ESC moraju biti vješti u rukovanju promjenjivim otporima i zaštiti motora. Također nedavne studije pokazale su da pogonski sustavi koji koriste motore s četkicama odgovarajućim ESC-ovima i prijenosnim omjerima mogu postići čak i do 35% poboljšanja u učinkovitosti u odnosu na određene sustave bez četkica (Winslow et al., 2016.). Nadalje, poboljšanja poput regenerativnog kočenja u ESC-ovima omogućuju učinkovitiju upotrebu energije (Green, 2015). Takvi ESC-ovi postepeno dobivaju na popularnosti zbog napretka u automobilskim sustavima, poput hibridnih elektromagnetskih kočionih sustava. Robusne sheme kontrole brzine i proklizavanja mogu se razviti za ove sustave, uzimajući u obzir kočnice i dinamične parametre vozila. Takve se kontrolne sheme mogu integrirati u različite sigurnosne sustave vozila, uključujući prilagodljivi tempomat i sustave protiv blokiranja kotača (Yazdanpanah & Mirsalim, 2015). Ta tehnologija onda postaje sve pristupačnija i postepeno se prenosi i na dronove. Vrhunska istraživanja neprestano pomiču granice funkcionalnosti ESC-a. S pojavom umjetne inteligencije i strojnog učenja, očekuje se da će ESC imati prediktivne i adaptivne sposobnosti, predviđanje zahtjeva motora i prilagodbu u stvarnom vremenu (Porselvi T., et al, 2022.). Pogledajmo shematski prikaz konstrukcije samog ESC-a.



Slika 11. Shematski prikaz ESC-a i asociranih komponenti (Izvor: Izrada autora)

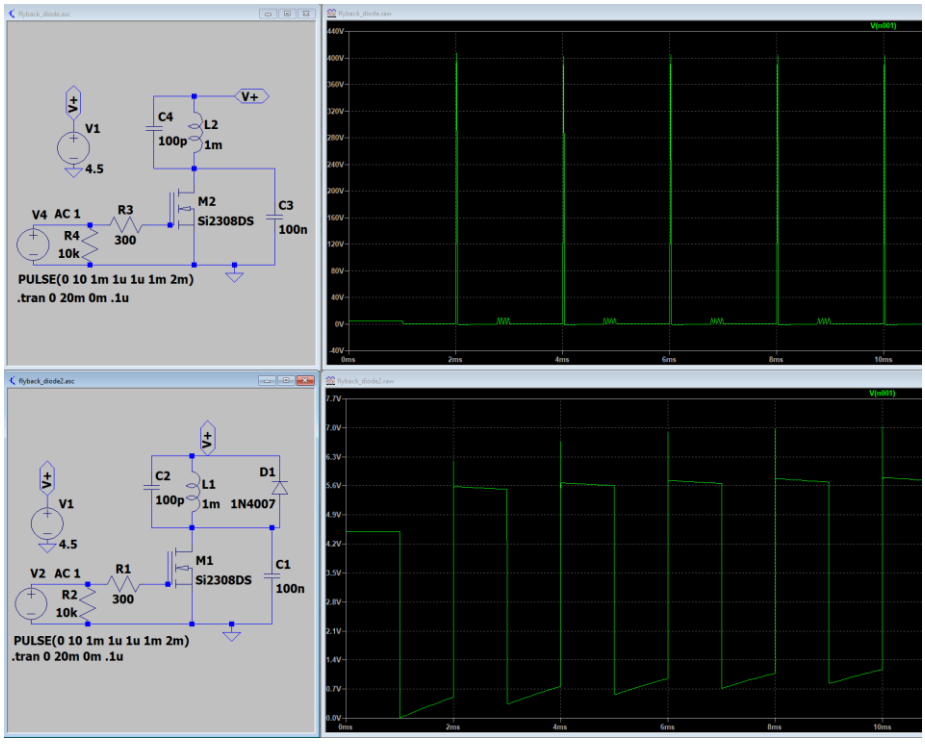
Ova shema je prikazana jednom, ali je u stvarnosti fizički ponovljena četiri puta, jednom za svaki motor. U sredini vidimo sam ESC sa svojom pripadajućom shemom, te su također označena i mjesta gdje se na njega spajaju i ostale komponente sustava. Primjerice motor se

spaja na kontaktima komponente J2, baterija na točki M1+ i zemlji, te mikrokontroler na GPIO konektoru kojeg smo konfigurirali za PWM izlaz, i zemlji.

Sastoji se od dva MOSFET tranzistora koji služe kao elektroničke sklopke koje puštaju i zaustavljaju struju na osnovi signala kojeg mu prezentiramo na bazi (pin 1 na tranzistorima). Pojednostavljeno, ovi tranzistori prate ulazni signal što je bliže moguće, ali za razliku od mikrokontrolera, mogu podnositi velika opterećenja. Zbog sigurnosti su postavljena dva tranzistora paralelno, kako bi zajedno dijelili posao te tako smanjili opterećenje na pojedinom tranzistoru. Ovo produljuje život ESC-u, te kao dodatnu korist znači da ako jedan prestane raditi u letu, dron će i dalje letjeti jer će drugi tranzistor preuzeti njegov teret.

Ispred baze svakog tranzistora nalazi se mali otpornik vrijednosti 300 ohm-a koji služi za ograničavanje struje koju tranzistor može povući iz mikrokontrolera, što sprječava njegovo uništenje. Zbog visoke stope promjene voltaže (delta V) na bridovima PWM signala, potrebno je osigurati izrazito brzo paljenje i gašenje tranzistora, pri čemu pomaže otpornik R3, jer trenutačno ispražnjava kapacitivnost mikrokontroler u zemlju, umjesto da se signal zadržava mikrosekundama dulje. Mali kondenzator C1 smanjuje brzinu mogućih brzih prijelaza napona, što uzrokuje manje zračenja i olakšava opterećenje bateriji, tako što se on prvi prazni u prvim trenutcima paljenja motora. Motor C2 služi istoj svrsi, ali za tranzistor.

Jedan od kritičnih elemenata ESC-a je dioda D1, tzv. „supresijska dioda“. Radi se o tom što motor djelomično predstavlja induktor, koji u sebi zadržava energiju i nakon što se tranzistor ugasi. U sljedećoj simulaciji napravljenoj u LTSpice, modelirali smo motor kao induktor.



Slika 12. Simulacija ESC-a u LTspice (Izvor: Izrada autora)

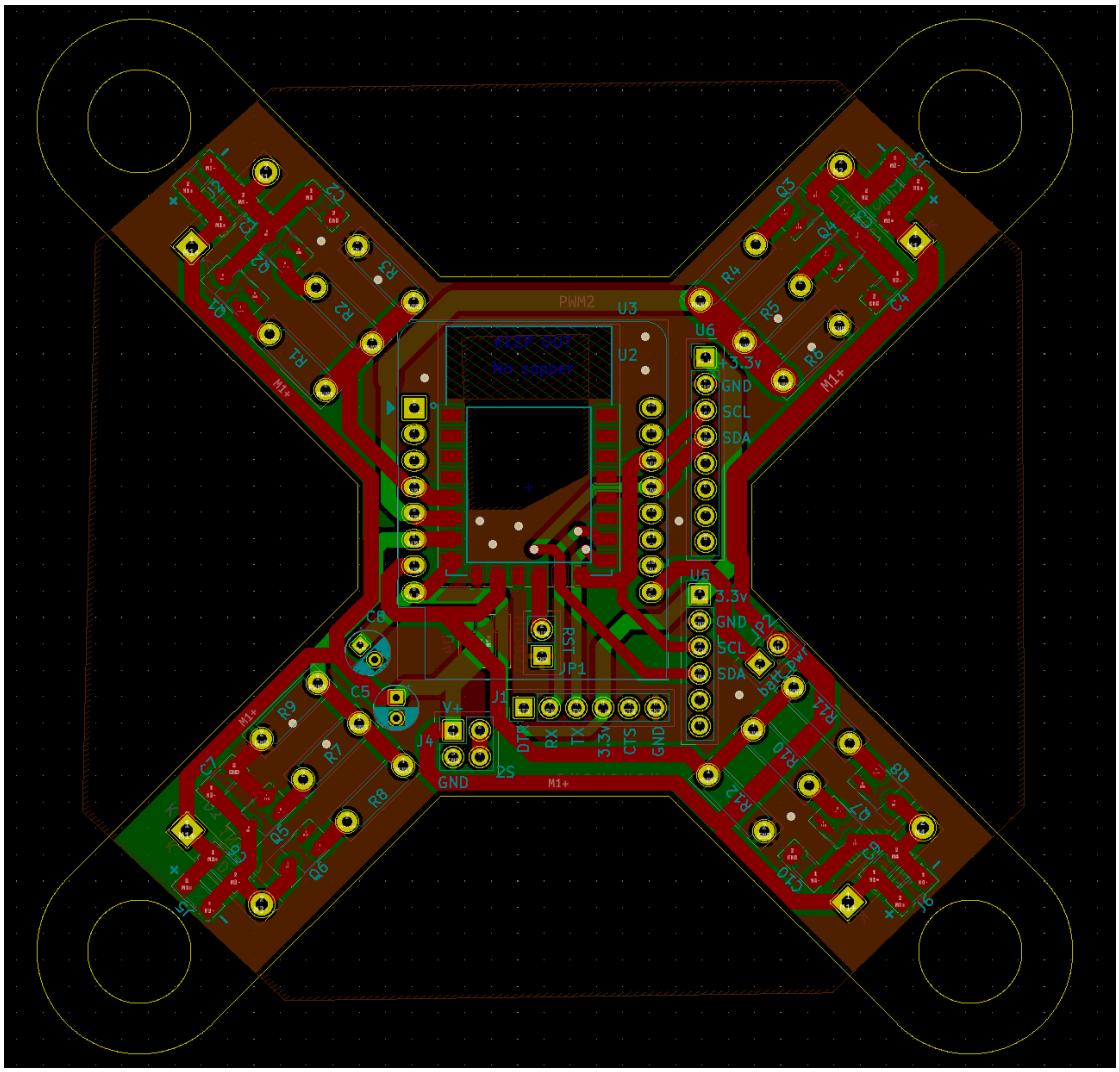
Induktivnost je svojstvo električnih krugova, koje ukazuje na tendenciju kruga da se odupre promjenama struje zbog magnetskih polja koja se stvaraju unutar njega. Kada struja teče kroz zavojnicu, nastaje magnetsko polje koje pohranjuje energiju i suprotstavlja se brzim promjenama struje (Johnson, 2012.). Ovo generira povratnu elektromotornu silu što generira vrlo visoke skokove struje. Ako ova struja nema gdje otići, visoka je vjerojatnost štete na komponentama, te generiranja velike količine elektromagnetske smetnje koja može poremetiti i rad komponenti izvan ESC-a. Na slici vidimo kako bez diode u gornjoj simulaciji, visina ovog napona može postići i 400V, što bi sigurno uništilo naš tranzistor. U donjoj simulaciji gdje je prisutna dioda D1, vrhovi su eliminirani, jer ona pruža put struji da se vrati nazad bateriji prije nego što se uspije skupiti do visokih razina.

7.1.2. Dizajn tiskane pločice

Nakon što je definiran shematski prikaz potrebno je to pretočiti u fizički dizajn. Postoji nekoliko temeljnih načela dizajna tiskanih pločica:

- Razmatranja rasporeda: Određivanje prioriteta postavljanja komponenti je ključno. Komponente poput žiroskopa i akcelerometara bi trebalo staviti što bliže centru gravitacije, kako bi se poboljšala stabilnost leta. ESC-ovi (elektronički regulatori brzine) i komponente za distribuciju energije možda će trebati strateško pozicioniranje za upravljanje toplinom.
- Mehanički: Dronovi zahtijevaju lagane, ali izdržljive PCB-ove. Materijali kao što je FR-4 često se koriste zbog svoje ravnoteže između težine, trajnosti i cijene. S obzirom na dinamičnu prirodu leta drona i moguću izloženost elementima, PCB-ovi za drone trebaju biti dizajnirani da izdrže vibracije, povremene udare i čimbenike okoliša poput vlažnosti ili temperaturnih fluktuacija.
- EMI i upravljanje šumom: Neposredna blizina visokofrekventnih komponenti, sustava napajanja i osjetljivih senzora zahtijeva stroge strategije upravljanja elektromagnetskim smetnjama (EMI) i bukom. Učinkovito upravljanje šumom, postignuto uzemljenjem, oklopom i strategijama rasporeda, je važno. Još jedna tehnika je stvaranje takozvanih „copper pour“-a gdje se cijela površina prelije kontinuiranim bakrom, koji poboljšava i EMI kroz pojednostavljenje povratnih struja, ali i služi kao štit.
- Upravljanje toplinom: s komponentama poput ESC-a koje generiraju značajnu količinu topline, učinkovito upravljanje toplinom, po mogućnosti putem toplinskih otvora ili

hladnjaka, postaje ključno. Pozicioniranje na mjestima gdje imaju adekvatnog kretanja zraka također znatno pomaže.

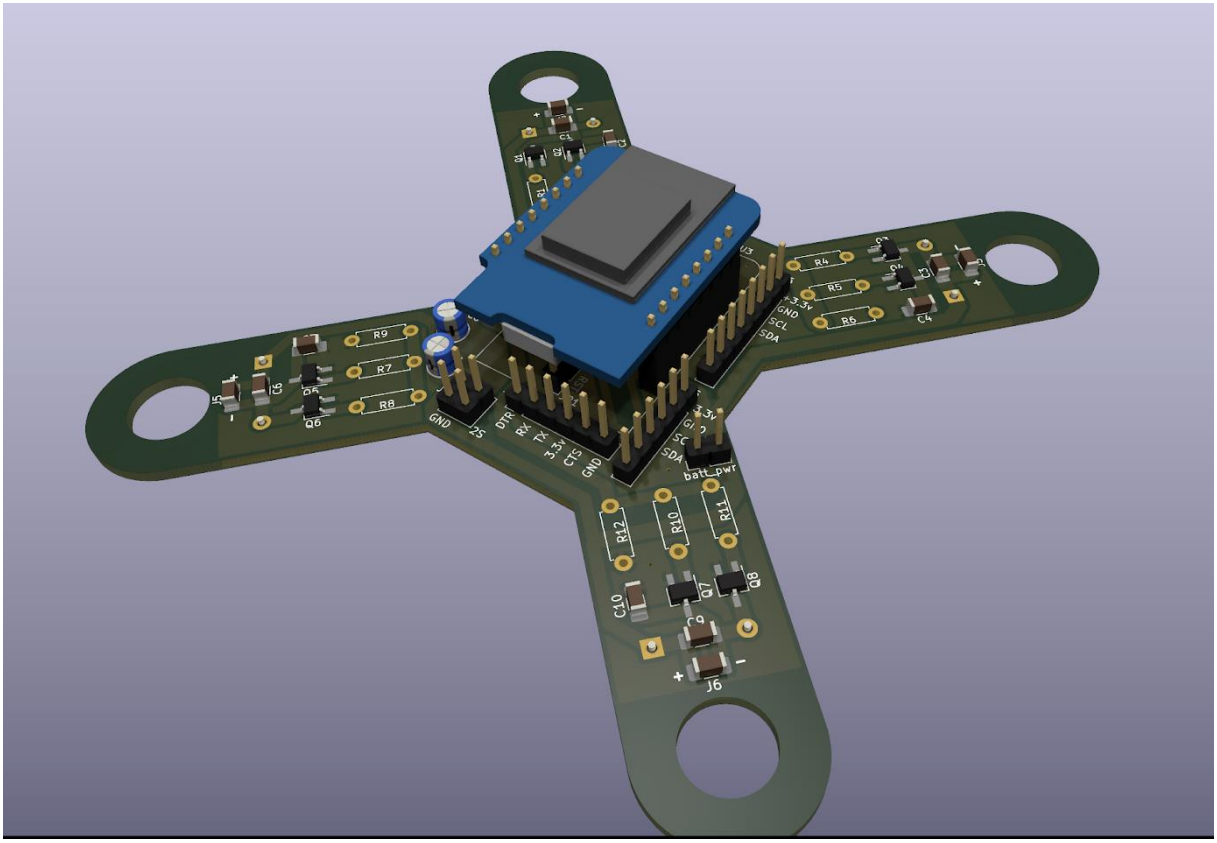


Slika 13. Fizička realizacija tiskane pločice drona (Izvor: Izrada autora)

Prvo su definirane dimenzije krajnjih točaka drona, što je 100 x 100 milimetara, nakon čega su postavljene rupe za motor na četiri kraja. Pošto znamo da smo odabrali motore 1020, znači da su rupe veličine 10mm. Zatim su uz rupe postavljeni ESC-ovi, iz razloga što kraće dužine bakrenih tragova što smanjuje otpor i induktivnost, i povećava efikasnost. Također, to je lokacija gdje postoji znatno kretanje zraka jer se nalazi odmah ispod propelera, što poboljšava hlađenje. Centralno su postavljen ESP-12F modul, a iznad njega drugi ESP8266 modul s pločicom. Zbog Wi-Fi prirode ovih uređaja bilo je potrebno specificirati zonu ekskluzije bakra, što bi ometalo snagu i s time i domet signala. Pokraj se nalaze dodatni moduli koji su pozicionirani što bliže za smanjenje otpora tragova. Pošto tiskana pločica ima dva sloja dio tragova je provučen na gornjem a dio na donjem sloju, što omogućuje optimizaciju dužina i smanjenje elektromagnetskih efekata. Svi tragovi moraju imati zatvoren strujni krug, što znači

da mora postaji put za povratne struje. Naj-efikasniji način za postići ovo je pomoću „copper pour“ tehnike gdje se bakar povezan na uzemljenje distribuira po cijeloj pločici. Na samim vrhovima drona je on maknut iz razloga težine, te također tu ne služi ulogu u električnom sustavu.

Na posljetku prisjećamo se da KiCad posjeduje i napredne mogućnosti 3D prikaza naše pločice. Ovo nam omogućuje da vizualno vidimo i provjerimo da smo stvorili ono što smo naumili, te tako uočimo potencijalne greške prije izgradnje, kojih u ovom slučaju nije bilo.



Slika 14. Računalno generirani 3D prikaz izgleda drona (Izvor: Izrada autora)

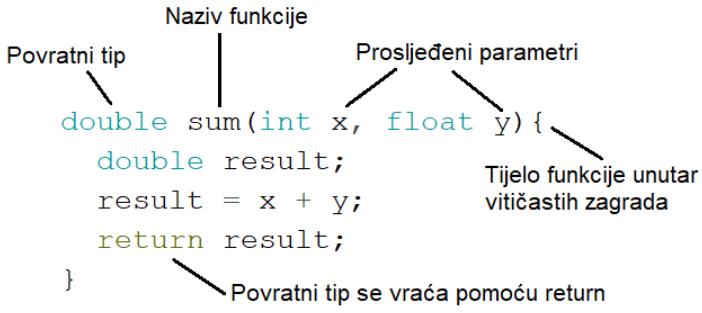
U sljedećim poglavljima fokusirati ćemo se na softversku implementaciju, programiranje mikrokontrolera i Android uređaja da postignu komunikaciju, ali i povezivanje mikrokontrolera sa svojim modulima koji su potrebni za ostvarivanje stabilnog leta.

7.2. Programiranje mikrokontrolera

Kao što smo već prikazali odabrali smo Arduino studio kao razvojno okruženje za naš ESP8266 mikrokontroler. Rješenje se sastoji od niza C++ biblioteka koji su povezani sa glavnom izvršnom klasom u kojoj se pozivaju. Od izlaska na tržište većine korištenih modula do danas je prošao dovoljan period vremena da sad postoji popriličan broj različitih biblioteka koje pojednostavljuju primjerice, povezivanje s modulom putem I2C protokola. Nakon kompilacije one će se spremiti i izvršavati direktno sa flash memorije našeg mikrokontrolera, koja iznosi 16MB i tvori primarnu limitaciju u smislu veličine koda kojeg možemo napisati, što u ovom slučaju neće predstavljati problem. Ova memorija je neizbrisiva stoga jednom kad je zapisana u flash, dron će je dalje automatski svaki put izvršavati pri pokretanju.
Arduino IDE se zasniva na C++ uz par razlika koje ćemo spomenuti. Kako bi lakše čitali kod, podsjetimo se ukratko na sintaksu.
Varijable su definirane kao: [tip podatka] [naziv varijable] = [vrijednost varijable];
U varijable ćemo pohranjivati različite podatke, od rednih brojeva konektora na kojima su nam spojeni motori i moduli, do različitih PWM signala, iznosa mjerenja iz modula i različitih parametara koji su nam potrebni. Jedan primjer takve varijable možemo vidjeti ispod, tipa int koja definira varijablu naziva *'motorPin'* na GPIO konektoru 12.

```
int motorPin = 12;
```

Drugi bitan građevni element našeg koda su funkcije. One čine kod čitljivijim tako što razlažu kod na više logičkih cjelina, koje obavljaju određen zadatak kojeg želimo izvršiti više puta, obično i s različitim ulaznim podatcima. Pisanje funkcija je dobra praksa jer je kod i lakši za održavanje i smanjuje broj pogrešaka.



Slika 15. Struktura funkcije u Arduinu (Izvor: izrada autora)

Razlike u odnosu na klasični C++ je i dodatak obaveznih setup() i loop() funkcija koje će se izvršiti nakon prijenosa na mikrokontroler, specifično setup() se poziva samo jednom prilikom pokretanja i zamišljeno je da se u njoj inicijaliziraju svi moduli i ostale stvari potrebne prije samog rada. Loop() se izvršava iznova konstantno, funkcionalno nalik na beskonačnu while() petlju ali uz dodatne optimizacije.
U idućim podpoglavljima ćemo prikazati na koji način je povezan cjelokupni sustav mikrokontrolera s pojedinim modulima, odnosno kako je to postignuto implementacijski u softveru. Jedan od kriterija pri implementaciji je i jednostavnost koda. ESP8266 sadrži procesor smanjenog seta instrukcija (eng. *RISC*), što znači da većina operacija zahtjeva veći broj ciklusa procesora da se izvrše. Jednostavne operacije poput zbrajanja cijelih brojeva su puno brže od kompleksnih petlja i grananja, koje bi bilo idealno prebaciti na stranu moćnijeg uređaja kao što je u ovom slučaju kontroler (mobitel), koliko je moguće.

7.2.1.Web poslužitelj

Kako bi klijent mogao komunicirati s našim dronom potrebno je uspostaviti konekciju, za koji smo definirali korištenje Wi-Fi protokola. Pogledajmo kod inicijalizacije jednog takvog poslužitelja na ESP8266 mikrokontroleru.

```
#include <ESP8266WiFi.h>
#include <WiFiClient.h>
#include <ESP8266WebServer.h>

ESP8266WebServer server(80);    // Stvori webserver koji sluša na portu 80
const char* ssid = "FOI_dron";  //Definicija SSID-a i šifre wifi mreže
const char* password = "Foidronpass12";
IPAddress    apIP(42, 42, 42, 42);
```

Prvo su povezane standardne biblioteke za korištenje s ESP8266 Wi-Fi mogućnostima, te je inicijaliziran objekt jedne od biblioteka, pritom prosljeđujući vrijednost odabranog porta u konstruktor. Prosljeđujemo broj 80 jer je to mrežni port definiran za HTTP protokol. U dva parametra definiramo naziv (SSID) mreže koju želimo stvoriti, i pripadajuću šifru za spajanje, kao i željenu IP adresu.

```
void setup(void){
  //Konfiguracija ESP8266 wifi mogućnosti
  WiFi.mode(WIFI_AP);
  // subnet FF FF FF 00
  WiFi.softAPConfig(apIP, apIP, IPAddress(255, 255, 255, 0));
  WiFi.softAP(ssid, password, 1, 0, 1);
  Serial.print("AP IP adresa: ");
  Serial.println(WiFi.softAPIP());

  server.on("/", HTTP_GET, handleRoot);
  server.on("/motors", handleMotors);  // Pozovi funkciju motors kada se
primi post na URI /motors
  server.onNotFound(handleNotFound);   // Kada klijent zatraži nepoznati URI
  server.begin();                      // Pokreni server
  Serial.println("HTTP poslužitelj pokrenut");
}
void handleRoot() {
  server.send(200, "text/plain", "Uspjesno ste povezani na dron");
}
void handleNotFound(){
  server.send(404, "text/plain", "404: Navedeni url ne postoji");
}
```

U setup() funkciji konfiguriramo WiFi mod operacije, koji može poprimiti tri vrijednosti:

- WIFI_STA – Mod stanice koji se koristi da se poveže s već postojećom Wi-Fi mrežom. Ako se konekcija prekine ili kontroler ponovno pokrene, ESP8266 će se automatski povezati sa zadnje korištenom pristupnom točkom kada je ponovno dostupna.
- WIFI_AP – pristupna točka koja stvara novu mrežu, i pruža pristup mreži stanicama koje se žele spojiti na nju. Tehnički u slučaju ESP8266 ovo je „*Soft Access Point*“ jer ne sadrži hardverske resurse tradicionalne pristupne točke poput router-a, drugo ime za ovakav oblik je „virtualni router“. Glavna limitacija je sa podržanim brojem klijenata (u slučaju ESP8266 maksimalno 8), i to što nema povezanosti na Internet, oboje od čega nije problem u ovom slučaju.

Nakon što je odabran AP potrebno ga je konfigurirati, što se radi kroz sljedećih nekoliko funkcija. Prvo pozivamo WiFi.softAPConfig() koji prima 3 parametra. Prvi je lokalna IP adresa koja je već prethodno definirana u objektu „IPAdress“, i služi kao IP adresa pristupne točke. Idući parametar je IP adresa gateway-a koji je mrežni čvor koji služi za komuniciranje s drugom mrežom koja koristi drukčiji mrežni protokol, ova adresa dodaje default rutu u IP routing tablici.

Posljednja je subnet maska koja dijeli IP adresu u dva dijela, jedan koji identificira host računalo a drugi koji identificira mrežu kojoj pripada. U ovom slučaju 24 bita su korištena za mrežu, i posljednjih 8 bitova za adresiranje računala.

U funkciji WiFi.SoftAP() konfiguriraju se dodatnih 5 parametra:

- SSID – naziv mreže, sadrži maksimalno 63 znakova. Jedino obavezno polje koje ne smije biti prazno
- Password – šifra stanice koja je potrebna za povezivanje. Ako se izostavi mreža je otvorena, a ako se uključi mora sadržavati minimalno 8 znakova po WPA2-PSK standardu.
- Channel – broj kanala na kojem se otvara mreža, mora biti numerička vrijednost od 1 do 13
- Hidden – ako je postavljeno na 1, mreža neće moći biti pronađena kroz identifikator (npr. na listi mreža na mobitelu)
- Max_connection – Definira maksimalni broj stanica koje se mogu spojiti, prima numeričku vrijednost od 0 do 8, sa 4 kao početnom vrijednosti.

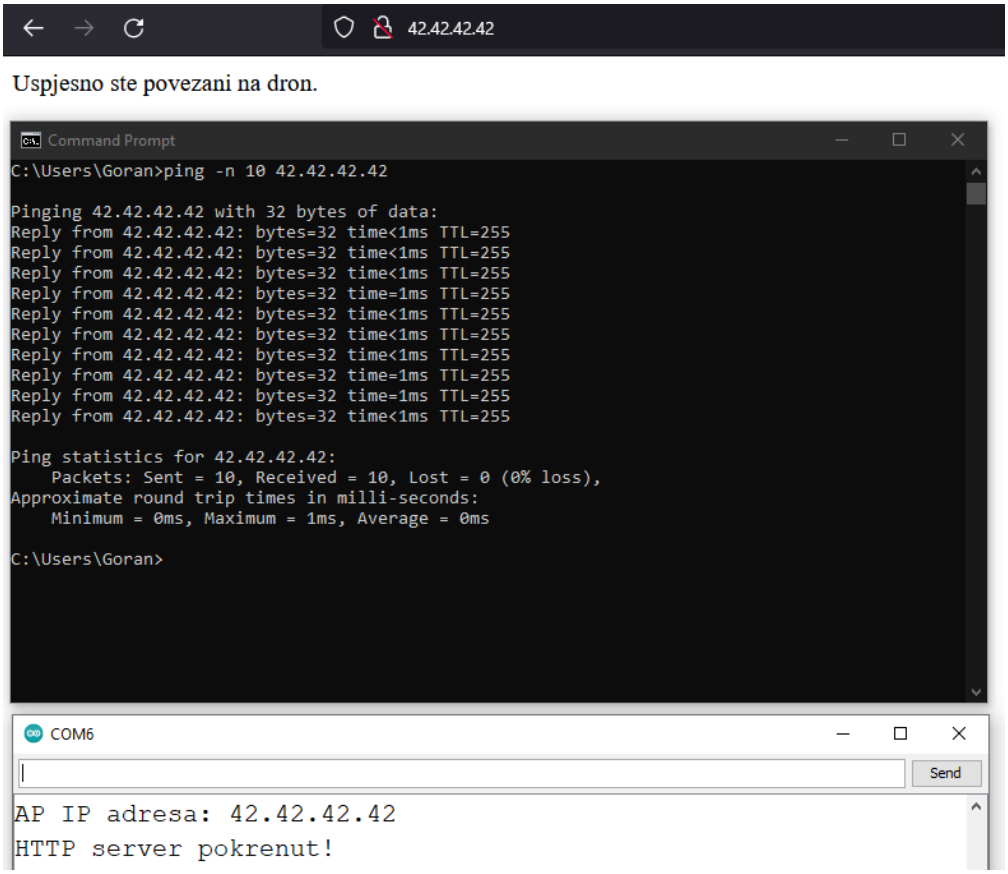
Kao test konfiguracije u sljedeće dvije linije ispisujemo na konzolni izlaz ispis trenutne IP adrese mrežnog sučelja pristupne točke pomoći WiFi.softAPIP() funkcije ugniježđene u Serial.println() funkciju.

Nakon konfiguracije potrebno je pokrenuti server što radimo koristeći prethodno inicijalizirani objekt klase ESP8266WebServer, tako što definiramo takozvane handler-e. Definiraju se pozivanjem server.on() funkcije koja prima 2 obavezna i 1 izborni parametar.

- Relativni URI – Prvi parametar specificira URI put relativno na početnu stranicu. Primjerice ako je adresa servera „42.42.42.42“, onda „/“ predstavlja prazan put, odnosno samu početnu stranicu, a „/motors“ bi predstavljao „42.42.42.42/motors“.
- HTTP metoda – Izabrana metoda koja se može izvršiti, u ovom slučaju specificirali smo da se može napraviti samo HTTP GET zahtjev, klijent može zatražiti podatke od servera. Ovaj parametar je ujedno i izborni parametar.
- Handler funkcija – Specifira koja funkcija će se pozvati kako bi obradio zahtjev. Drugim riječima kada je URI posjećen, poziva se definirana funkcija.

U sljedećoj liniji smo također i definirali handler funkciju za sve nepoznate, odnosno nedefinirane URI-je pomoću server. onNotFound().
U svakoj handler funkciji se formira HTTP poruka koja se zatim šalje putem server.send() funkcije koja prima 3 parametra.

- Code – HTTP response kod, npr. 200 za „OK“ ili 404 za „Nije pronađeno“
- Content_type – Tip HTTP sadržaja, npr. „text/plain“ ili „image/png“, itd.
- Content – sam sadržaj tijela



Slika 16. Prikaz pristupa poslužitelju (Izvor: Izrada autora)

Kada pokrenemo dron, adresa poslužitelja će nam se izbaciti u serijski monitor, te ukoliko se povežemo na mrežu koju smo specificirali sa SSID i šifrom, te otvorimo tu adresu u web poslužitelju, vidimo da naš GET zahtjev radi ispravno i poruka je vraćena nazad. Rezultat testa latencije u naredbenom prozoru pokazuje da je vrijeme puta od klijenta, do servera, i nazad do klijenta, u prosjeku oko 1 milisekunde, što je povoljno za uspostavljanje responzivnosti drona.

7.2.2.Barometar

U ovom poglavlju fokusirati ćemo se na inicijalizaciju i kalibraciju BME280 modula za pritisak i temperaturu. Pogledajmo kod za inicijalizaciju modula:

```
#include <Wire.h>
#include <BME280I2C.h>

// Definiranje I2C pinova i standardnog tlaka
#define SDA_PIN 4    // GPIO4 - SDA pin
#define SCL_PIN 5    // GPIO5 - SCL pin
#define SEALEVELPRESSURE (1013.25)

// Instancijacija klase i definiranje parametara za kasnije korištenje
BME280I2C bme;
float temperature, humidity, pressure, altitude, droneAlt;
float initAlt = 0;
bool bme_found = false;

// U setup funkciji inicijaliziramo I2C i BME280 modul i otvaramo serijski
monitor
void setup(void){
  Serial.begin(115200);
  Wire.begin(SDA_PIN, SCL_PIN);
  if (!bme.begin()) {
    Serial.println("Nije pronađen BME280 senzor, provjerite konekciju!");
  }else{
    bme_found=true;
  }}
```

Prvo povezujemo biblioteku <BME280I2C.h> koja nam omogućuje znatno bržu i pouzdaniju uspostavu konekcije s ovim modulom, i <Wire.h> koja je ugrađena s Arduino studiom i omogućuje upravljanje I2C konekcijama, te zatim definiramo na kojim konektorima se nalaze I2C portovi. U slučaju našeg ESP8266 to su pinovi 4 i 5. Nakon toga potrebno je instancirati objekt iz naše klase biblioteke, i stvaramo nekolicinu parametara koji će se kasnije koristiti za spremanje vrijednosti temperature, tlaka zraka, nadmorske visine, i visine drona od tla.
U setup() funkciji se odvija sama inicijalizacija modula, gdje otvaramo serijski monitor na 115200 baud rate i pozivamo Wire.begin() sa parametrima naših I2C pinova. Ova funkcija inicializira Wire.h biblioteku i spaja se na I2C bus kao kontroler ili peripheral, i poziva se samo

jednom. Funkcija *'bme.begin()*' vraća true ako je senzor pronađen, i false ako nije, pomoću čega provjeravamo da li je inicijalizacija uspješna i ako je postavljamo *'bme_found'* parametar na true, što će nam koristiti kasnije.

```
void calculateValues() {
  temperature = bme.temp();
  pressure = bme.pres();

  float seaLevelPressure = SEALEVELPRESSURE;  // Standardni tlak zraka na
morskoj razini
  float lapseRate = 0.0065;  // Stopa pada temperature s visinom u K/m
  float temperatureOffset = 15.0;  // Odmak temperature za kompenzaciju

  // Izračun visine na osnovi formule
  altitude = ((pow((seaLevelPressure / pressure), (1 / 5.257)) - 1.0) *
            (temperature + 273.15 + temperatureOffset)) / lapseRate;
}
 if(initAlt==0){

    Serial.println("Postavljam početnu visinu");
    initAlt = altitude;
  }

  droneAlt = altitude-initAlt;

}
```

Sada kada smo povezani na modul, možemo iščitati vrijednosti vrijednosti sa senzora, postaviti vrijednost tlaka zraka na morskoj razini, kao i stopu pada temperature s visinom i kompenzacijskim odmakom temperature. Ovo sve nam je potrebno kako mogli izračunati trenutnu nadmorsku visinu u metrima pomoću sljedeće formule.

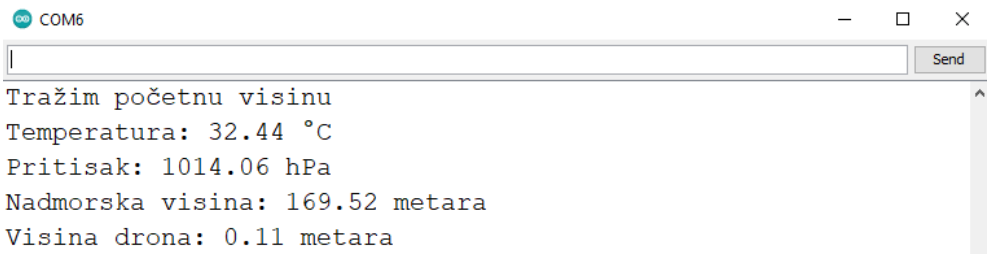
$$nadmorskaVisina = \frac{\left(\left(\frac{morskiTlak}{trenutniTlak}\right)^{\frac{1}{5.257}} - 1\right) * (temperatura + 273.15 + temperaturaOdmak)}{stopaPada}$$

Zanimljivost kod ove formule je da nema definitivan izvor u literaturi, iako je široko korištena u avijaciji, meteorologiji i drugim područjima gdje su potrebne brze procjene visine na osnovi

ograničenih podataka, i primjenama u troposferi, jer nije precizna na izrazito velikim visinama. Razvila se postepeno kroz vrijeme i vuče svoje korijene iz 17. stoljeća, sa doprinosima od Blaise Pascal-a, Torriceli-ja i drugih znanstvenika.

- morskiTlak - atmosferski tlak na razini mora i služi kao referentna točka za tlak na površini Zemlje.
- trenutniTlak - atmosferski tlak na trenutnoj nadmorskoj visini. Mjeri u istim jedinicama kao i tlak na razini mora.
- temperatura – temperatura na trenutnoj nadmorskoj visini. Mjeri se u stupnjevima Celzijevima.
- temperaturaOdmak - Ovo je vrijednost pomaka dodana temperaturi. Koristi se za uzimanje u obzir varijacija temperature zbog čimbenika kao što su stopa pada i lokalni uvjeti. Temperatura bi trebala biti u Kelvinima, tako da se pomak dodaje Celzijevoj temperaturi i zatim pretvara u Kelvine dodavanjem 273,15.
- stopaPada - je stopa kojom temperatura opada s povećanjem nadmorske visine. Mjeri se u stupnjevima Celzija po jedinici nadmorske visine (npr. metrima). Obično se koristi standardna vrijednost, kao što je -0,0065°C/m.

Vrijednost eksponencije „$\frac{1}{5.257}$“ je izveden iz barometričke formule i koristi se da bi se kompenziralo u varijaciji pritiska s visinom u standardnoj atmosferi, i primjer je vrijednosti koja je s vremenom definirana eksperimentalno kako bi najbolje odgovarala formuli. Izlazna vrijednost je konačna visina u metrima.
Postavimo li ispis ovih vrijednosti pomoću Serial.print() funkcije, dobivamo sljedeće vrijednosti.



Slika 17. Prikaz funkcionalnosti BME280 modula (Izvor: Izrada autora)

Nadmorska visina izlazi začuđujuće precizno, oko 169 metara za grad Varaždin koji slovi u prosjeku 173 metara nadmorske visine. Sljedeće je potrebno ove informacije postaviti dostupnima kroz poslužitelj.

```
void setup(void){
   // …
  server.on("/temp", handleTemp);
  server.on("/pres", handlePres);
  server.on("/altitude", handleAltitude);
  server.on("/droneAlt", handleDroneAlt);
   // …
  server.begin();
}
void handleTemp() {
  String message = "";
  message+="Temperatura: ";
  message+=temperature;
  message+="*C";
  server.send(200, "text/plain", message);
  Serial.println(message);
}
void handleAltitude() {
  String message = "";
  message+="Nadmorska visina: ";
  message+=altitude;
  message+="m";
  server.send(200, "text/plain", message);
  Serial.println(message);
}
void handlePres() {
  String message = "";
  message+="Pritisak: ";
  message+=pressure;
  message+="hPa";
  server.send(200, "text/plain", message);
  Serial.println(message);
}
```

Vratimo se nazad u setup() funkciju i dodajmo handler-e za poslužitelja. Kada klijent zatraži jedan od navedenih URI ekstenzija, automatski se poziva asocirana handler funkcija. Svaki parametar je odvojen u svoj zasebni URI jer se ne moraju svi parametri zatraživati istom frekvencijom sa strane klijenta. Slanje pomoću server.send() i objašnjenje njezina tri ulazna parametara je objašnjeno u prethodnom poglavlju.

7.2.3.PWM

Na posljetku kako bi se sami motori zavrtili, potrebno je prevesti naredbe u niz modulacija pulseva koji se šalju ESC-u. Pogledajmo kako je to napravljeno za ovaj dron.

```
// Funkcija koja obrađuje ulaze za motore
void handleMotors() {
  String message = "";
  if (server.arg("Y")== ""){     //Parametar nije pronađen
  message += "Argument za uzlaz nije pronađen\n";
  } else {     //Parametar pronađen
  message += "Uzlaz argument = ";
  message += server.arg("Y");    //Vraća vrijednosti upita parametra
  message += "\n";
  }
    if (server.arg("X")== ""){     //Parametar nije pronađen
  message += "Argument za okretanje oko osi nije pronađen\n";
  } else {     //Parametar pronađen
  message += "Okretanje oko osi argument = ";
  message += server.arg("X");      //Vraća vrijednosti upita parametra
  message += "\n";
  }
    if (server.arg("F")== ""){     //Parametar nije pronađen
  message += "Naprijed  i nazad argument nije pronađen\n";
  } else {     //Parametar pronađen
  message += "Naprijed i nazad argument = ";
  message += server.arg("F");     //Vraća vrijednosti upita parametra
  message += "\n";
  }
    if (server.arg("R")== ""){     //Parametar nije pronađen
  message += "Lijevo i desno argument nije pronađen\n";
  } else {     //Parametar pronađen
  message += "Lijevo i desno argument = ";
  message += server.arg("R");     //Vraća vrijednosti upita parametra
  message += "\n";
  }
server.send(200, "text/plain", message);
  // [...]
```

Iznad se nalazi funkcija handleMotors() koja se poziva svaki put kad klijent posjeti. Nju smo definirali u kodu u setup() funkciji kao „server.on("/motors", handleMotors)“.
Ona prvo provjerava za prisutnost 4 kontrolna parametra, označenih kao „Y, X, F, R“.
Kada klijent-upravljač posjeti ovaj URI, treba predati ta četiri parametra zajedno sa asociranom vrijednosti, za koji ćemo vidjeti kod u sekciji programiranja Android aplikacije. Na strani mikrokontrolera ova provjera se izvršava pomoću *'server.arg()*' funkcije u koju se prosljeđuje vrijednost argumenta kojeg tražimo. Ukoliko nije prisutan dokumentira se greška, a ukoliko je vraća se vrijednost argumenta u poruku. Kada se akumulira poruka kroz sva 4 parametra, možemo je vratiti nazad kao odgovor na upit klijentu, kako bi dobio potvrdu o primitku i HTTP kod 200.

```
  // Funkcija koja obrađuje inpute za motore
void handleMotors() {

  // [...]
  int throttle = constrain(server.arg("Y").toInt(),0,1023);
  int yaw = constrain(server.arg("X").toInt(),-125,125);
  int forward = constrain(server.arg("F").toInt(),-125,125);
  int right = constrain(server.arg("R").toInt(),-125,125);;

  // Mapiraj vrijednosti od 0 do 255
  yaw = map(yaw, -125, 125, 0, 1023);
  forward = map(forward, -125, 125, 0, 1023);
  right = map(right, -125, 125, 0, 1023);
}
```

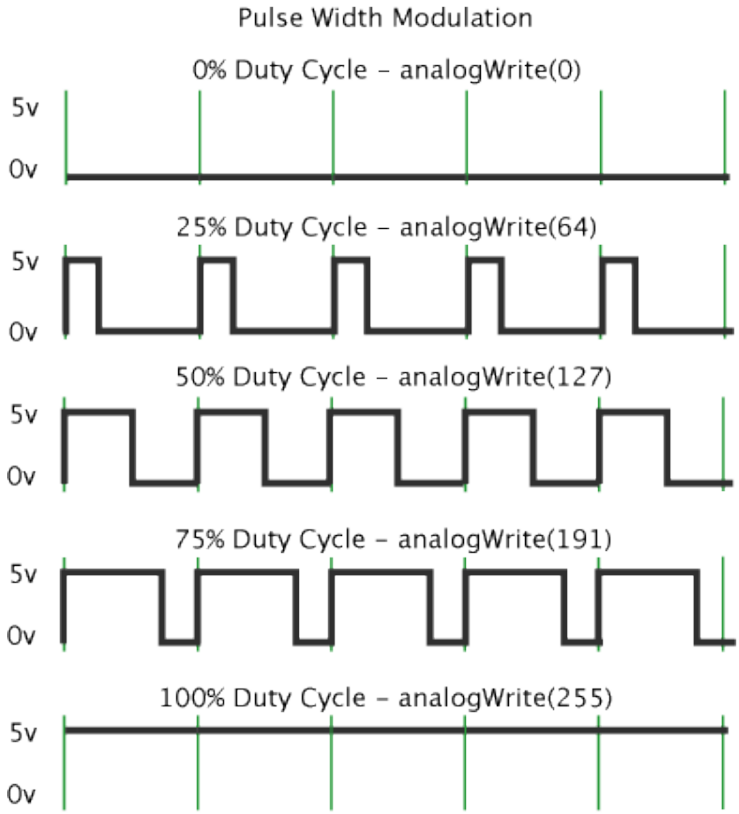
Nastavljajući dalje unutar iste funkcije, pridružujemo ove vrijednosti primjerenim varijablama. To radimo unutar *'constrain*()' funkcije koja prima tri argumenta. Prvi je naša vrijednost varijable, a iduće dvije je raspon unutar kojeg želimo primati vrijednosti. Na ovaj način odbacujemo vrijednosti koje su veće ili manje od definiranih granica, što nam daje konzistentan i predvidljiv raspon ulaza (na primjer, da nije moguće da dođe vrijednost -1 zbog greške zaokruživanja na strani kontrolera). Iako ovo nije prijeko nužno u našem slučaju jer je kontroler digitalan, predstavlja dobru praksu. Nakon toga radimo mapiranje pomoću map(), koje skalira vrijednosti od početnog raspona u željeni raspon. Naš kontroler na bazi Android aplikacije će raditi sa pozitivnim i negativnim vrijednostima od -125 do 125, centriranim u 0 (sredina), dok PWM izlaz iz Arduino pin-a ne može poprimiti negativne vrijednosti stoga će sve biti ponovno skalirano u 10-bitnu vrijednost digitalnog izlaza, što iznosi od 0 do 1023.

```
void runPID(){

  // [...] PID kod iz idućeg poglavlja
  // Postavi PWM-e pojedinih motora
  int motor1PWM = throttleOutput + yawOutput + forwardOutput + rightOutput;
  int motor2PWM = throttleOutput - yawOutput + forwardOutput - rightOutput;
  int motor3PWM = throttleOutput + yawOutput - forwardOutput - rightOutput;
  int motor4PWM = throttleOutput - yawOutput - forwardOutput + rightOutput;

  setPWM(motor1PWM,motor2PWM,motor3PWM,motor4PWM);
}
//Postavi izlaz pinova za motore
void setPWM(int PWM1, int PWM2, int PWM3, int PWM4){
analogWrite(motorPin, PWM1);
analogWrite(motorPin2, PWM2);
analogWrite(motorPin3, PWM3);
analogWrite(motorPin4, PWM4);
}
```

U funkciji runPID(), koja će biti dodatno proširena u idućem poglavlju, uzimamo vrijednosti iz kontrolera koje smo dobili i obradili pomoću map() i constrain(), te računamo konkretne brzine motora potrebne da se izvede manevar kojeg korisnik namjerava.

Naredba za vertikalnu translaciju (throttleOutput) se u slučaju za sva četiri motora zbraja, odnosno ako je 255 svi motori će raditi maksimalno da se dron uspinje, a ako je 0 svi će se zaustaviti da bi se dron spustio.

Za rotaciju oko osi potrebno je zbrajati i oduzimati dijametralne motore, što znači da će se u jednom smjeru rotacije „yawOutput“ zbrajati, a u drugom minusi poništiti što uzrokuje rotaciju u suprotnom smjeru. Isti slučaj je i za horizontalnu translaciju, odnosno kretanje naprijed nazad i lijevo desno. Za ići naprijed (forwardOutput) u jednom smjeru će se vrijednosti zbrojiti, a kretanje unatrag je postignuto tako što je će ulazna vrijednost biti negativna, što uzrokuje poništavanje minusa u kretanje u suprotnom smjeru. Isto je napravljeno i sa kretanjem lateralno (rightOutput), samo sa druga dva motora.



Slika 18. Prikaz PWM izlaza s obzirom na vrijednost (Izvor: docs.arduino.cc)

Funkcija *'setPWM*()' je ona koja radi izlaz na samim pinovima koji su spojeni na ESC, te se sastoji od četiri *'analogWrite*()' funkcije. U prvom argumentu uzima broj pina motora koji je definirali ranije u vrhu datoteke, a u drugom argumentu samu vrijednost izlaza koja je prethodno izračunata. Funkcija proizvodi niz pulsiranja koji variraju u vremenskoj domeni, odnosno razlikuju se u širini ali ne i intenzitetu.

Ovo omogućuje efikasnu operaciju ESC-a koji uvijek zahtjeva vrijednost iznad određene voltaže za potpunu efikasnost (u ovom slučaju 2.5V), a brzina rotacije se onda varira širinom, odnosno frekvencijom pulsiranja. Na slici vidimo konkretni primjer kako će izlaz izgledati za ulaz od 0, 64,127,191 i 255. Nakon ovih koraka motori se uspješno rotiraju na osnovi ulaznih vrijednosti primljenih putem HTTP servera.

7.2.4.Jedinica za inercijsko mjerenje (IMU)

U ovom poglavlju osvrnuti ćemo se na praktične aspekte i principe korištenja MPU-6050 modula, gdje ćemo prikazati inicijalizaciju i konfiguraciju, kao i kalibraciju. Samo korištenje modula za PID kontrolu biti će prikazano u sljedećem poglavlju.

```
#include <Wire.h>
#include "MPU6050_6Axis_MotionApps20.h"
MPU6050 mpu;
// MPU kontrolne/statusne varijable
uint8_t dmpStatus;      // povratni status nakon svake operacije
uint8_t fifoBuffer[64]; // FIFO spremnik za pohranu
// varijable za orijentaciju/kretanje
Quaternion q;           // [w, x, y, z]         spremnik za kvaternion
void setup() {
  Wire.begin();
  Serial.begin(9600);
  mpu.initialize();   // Inicijalizacija MPU6050
  if (mpu.testConnection()) {
    Serial.println("Uspješna veza s MPU6050!");
  } else {
    Serial.println("Veza s MPU6050 nije uspjela. Provjerite konekciju!");
  }
  Serial.println(F("Inicijalizacija DMP-a..."));
```

// [….] nastavak u sljedećem bloku

Nakon uključivanja potrebnih biblioteka, inicijaliziraju se varijable potrebne za spremanje podataka koji će se kasnije koristiti. Stvara se instanca klase MPU6050, varijabla za pohranu povratnog statusa nakon svake operacije uređaja, gdje 0 označava uspjeh, a ne-nula vrijednost označava grešku. Također se inicijalizira i spremnik za pohranu podataka iz FIFO (eng. First In, First Out) stack-a senzora. Na posljetku je i varijabla za pohranu kvaternion vrijednosti koje predstavljaju orijentaciju senzora. U *'setup*()' funkciji nalazi se kod koji se poziva jednom kada Arduino započne. Sadrži kod za inicijalizaciju i konfiguraciju MPU6050 senzora. Inicijalizira I2C komunikaciju, pokreće serijsku komunikaciju s brzinom prijenosa od 9600, omogućujući praćenje i otklanjanje pogrešaka putem serijskog monitora. Na posljetku poziva se funkcija za inicijalizaciju MPU6050 senzora sa '.*initialize*()', i zatim konekcija testira za uspješnost u idućem bloku sa '.*testConnection*()'.

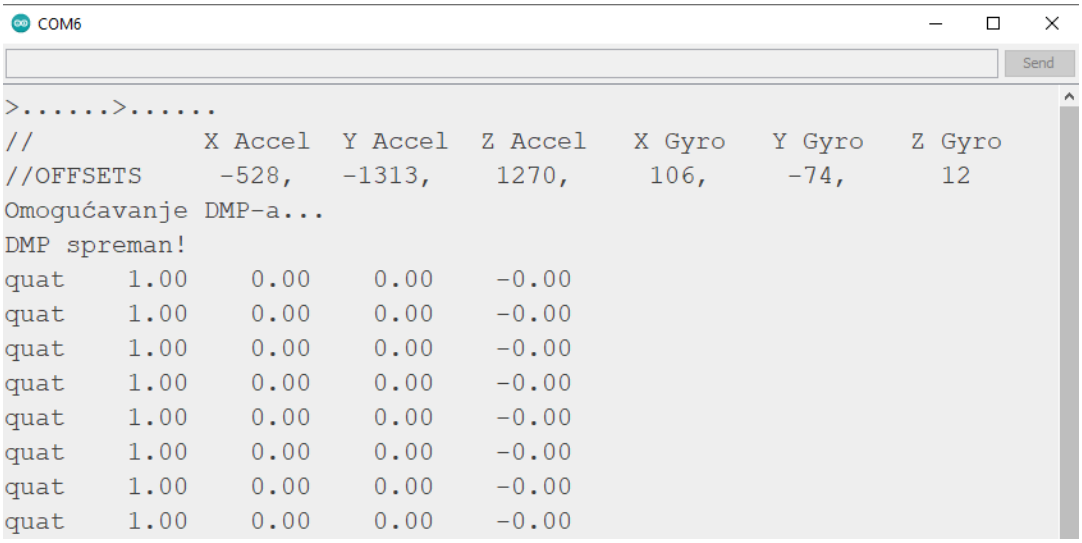
```
  dmpStatus = mpu.dmpInitialize();
// provjerite je li uspjelo (vraća 0 ako je)
  if (dmpStatus == 0) {
        // generiranje pomaka i kalibriranje
        mpu.CalibrateAccel(6);
        mpu.CalibrateGyro(6);
        mpu.PrintActiveOffsets();
        // uključite DMP, sada kada je spreman
        Serial.println(F("Omogućavanje DMP-a..."));
        mpu.setDMPEnabled(true);
        Serial.println(F("DMP spreman!"));
    } else { // greška!
        // 1 = početno učitavanje memorije nije uspjelo
        // 2 = ažuriranja konfiguracije DMP-a nisu uspjela
        Serial.print(F("Inicijalizacija DMP-a nije uspjela (kod "));
        Serial.print(dmpStatus);
        Serial.println(F(")"));
    }
}
```

Idući dio koda inicijalizira DMP (eng. Digital Movement Processor), koja je hardverska značajka MPU6050 koju smo objasnili u teorijskom dijelu. Prvo inicijaliziramo DMP i spremamo povratnu vrijednost ove operacije u varijablu *'dmpStatus'* koja pruža više informacija u uspješnosti inicijalizacije. Nakon toga se pozivaju funkcije za kalibraciju akcelerometra i žiroskopa. Broj 6 predstavlja broj uzoraka koji se koriste za kalibraciju. Funkcija *'PrintActiveOffsets*()' linija ispisuje trenutne kalibracijske pomake na serijski monitor, što može biti korisno za dijagnostiku i praćenje kalibracije. *'mpu.setDMPEnabled*()' pozivaj funkciju koja ga stvarno i omogućuje unutar senzora. Nakon toga ako inicijalizacija DMP-a nije uspjela, blok koda nakon *'else'* ispisuje poruku o grešci na serijski monitor, zajedno s kodom greške. To može pomoći u dijagnostici i otklanjanju problema.

```
void loop() {
    // čitanje paketa iz FIFO-a
    if (mpu.dmpGetCurrentFIFOPacket(fifoBuffer)) { // Dohvatite najnoviji
paket
            // prikažite vrijednosti kvaterniona u jednostavnom matričnom
obliku: w x y z
            mpu.dmpGetQuaternion(&q, fifoBuffer);
            Serial.print("quat\t");
            Serial.print(q.w);
            Serial.print("\t");
            Serial.print(q.x);
            Serial.print("\t");
            Serial.print(q.y);
            Serial.print("\t");
            Serial.println(q.z);
    }
}
```

U glavnoj petlji programa prvo se čita koja čita najnoviji paket podataka iz FIFO stack-a senzora, ako je paket dostupan, funkcija vraća *'true'*, a podaci se pohranjuju u *'fifoBuffer'*. Funkcija *'dmpGetQuaternion*()' konvertira sirove podatke iz *'fifoBuffer'* u kvaternion format i pohranjuje ih u varijablu 'q'. Iduće linije ispisuju vrijednosti kvaterniona na serijski monitor u formatu "quat w x y z". Koriste se tabulatori (`\t`) za odvajanje vrijednosti, što olakšava čitanje i analizu podataka. Kvaternioni su kompleksni matematički objekti, koji se u kontekstu ovog koda koriste za predstavljanje orijentacije ili rotacije u 3D prostoru. Oni nude nekoliko prednosti:

- Manje kompleksni - Za razliku od nekih drugih metoda, kvaternioni su manje osjetljivi na određene matematičke probleme koji mogu uzrokovati neočekivano ponašanje.
- Učinkovitiji**:** Kvaternioni su brži za računanje, pogotovo u scenariju pri korištenju s DMP-om, što je važno u aplikacijama u stvarnom vremenu kao što je stabilizacija drona.



Slika 19. Prikaz kvaterniona nakon uspješne kalibracije (Izvor: Izrada autora)

7.2.5.PID kontroler

Iako smo dosad uspostavili osnovni kostur kontrole nad dronom, zbog činjenice da su dronovi intrinzično nestabilni zbog stalne varijacije vanjskih uvjeta poput vjetra i nesimetričnosti centra gravitacije potrebno je uvesti zatvorenu petlju kontrole, odnosno kontinuirano nadgledanje i prilagođavanje izlaza. U sljedećem tekstu će biti spomenut samo dio koda koji se nadodan za ostvarivanje funkcionalnosti, kako bi se izbjeglo ponavljanje već pređenog.

```
#include <Wire.h>       // Biblioteka za I2C komunikaciju
#include "MPU6050_6Axis_MotionApps20.h"

// PID parametri
float Kp = 1.5; // Proporcionalni koeficijent
float Ki = 0.1; // Integralni koeficijent
float Kd = 0.1; // Derivativni koeficijent
unsigned long previousTime = 0;
float euler[3]; // [x, y, z]
```

Sam kontroler je implementiran bez korištenja PID biblioteka. Prvo će se definirati tri PID parametra, odnosno koeficijenta.

- Kp - Proporcionalni koeficijent koji određuje koliko će kontrolni signal reagirati na trenutačnu grešku.
- Ki - Integralni koeficijent koji akumulira prethodne greške kako bi se eliminirala postojana pogreška.
- Kd - Derivativni koeficijent koji predviđa buduće greške i djeluje na temelju brzine promjene greške.

Kod izračuna derivacija vrlo je bitno pratiti koliko vremena je proteklo u izračunu derivativne komponente, kako bi mogli obračunati za razlike u vremenima izračuna. Definira se i polje Eulerovih kutova (x, y, z) za predstavljanje orijentacije objekta.

```
float errorPitch, errorRoll; // Greške za svaku os
float previousErrorPitch = 0, previousErrorRoll = 0; // Prethodne greške za derivativni izračun
float integralPitch = 0, integralRoll = 0; // Integralne greške
double rightInput, throttleInput, yawInput, forwardInput;
// Željeni kutovi
float desiredPitch = 0.0;
float desiredRoll = 0.0;

void setup(void){
  Serial.begin(74880);          // Pokretanje serijske konekcije za komunikaciju s računalom
  Wire.setClock(400000); // 400khz brzina I2C bus-a
  analogWriteFreq(5000); // 5khz PWM frekvencija
  analogWriteResolution(10); //10 bitova PWM rezolucija (0-1023)
  mpu.initialize();
}
```

Nakon tog definira se niz varijabli za greške i kontrolu:

- 'errorPitch, errorRoll' : Greške za svaku os (nagib i valjanje).
- 'previousErrorPitch , previousErrorRoll' : Prethodne greške za izračun derivativne komponente.
- 'integralPitch , integralRoll' : Akumulirane integralne greške za svaku os.
- 'rightInput, throttleInput, yawInput, forwardInput' : Varijable za unos kontrolnih signala.
- 'desiredPitch , desiredRoll ' : Željeni kutovi za nagib i valjanje.

Zatim se inicijalizira serijska komunikaciju s računalom na brzini od 74880 bauda, i postavlja brzinu I2C bus-a na 400 kHz. Ovo nam omogućuju maksimalnu brzinu komunikacije između kritičnog modula MPU-6050 i ESP8266, kako bi mikrokontroler mogao zaprimati podatke iz

akcelerometra i žiroskopa s najmanjom latencijom. *'analogWriteResolution*(10)' : Postavlja rezoluciju PWM signala na 10 bita (raspon od 0 do 1023), što omogućuje finiju gradaciju vrijednosti, i s time veću preciznost kontrole motora.

```
//Glavna petlja poslužitelja
void loop(void){
 // Čitanje kvaterniona iz MPU6050
  if (mpu.dmpGetCurrentFIFOPacket(fifoBuffer)) {
    mpu.dmpGetQuaternion(&q, fifoBuffer);
    mpu.dmpGetEuler(euler, &q); // Dobivanje Eulerovih kutova
  }
unsigned long currentTime = micros();
float dt = (currentTime - previousTime) / 1000000.0; // pretvori u sekunde
previousTime = currentTime;
// Izračunajte greške
errorPitch = desiredPitch - euler[2]; // theta za pitch
errorRoll = desiredRoll - euler[1]; // phi za roll
// Izračunajte integralne greške
integralPitch += errorPitch * dt;
integralRoll += errorRoll * dt;
float derivativePitch = (errorPitch - previousErrorPitch) / dt;
float derivativeRoll = (errorRoll - previousErrorRoll) / dt;
  // Izračunajte PID izlaze
float pitchOutput = Kp * errorPitch + Ki * integralPitch + Kd *
derivativePitch;
float rollOutput = Kp * errorRoll + Ki * integralRoll + Kd * derivativeRoll;
// Ažurirajte prethodne greške
previousErrorPitch = errorPitch;
previousErrorRoll = errorRoll;
// Izračunajte izlazne PWM signale za motore
runPID(pitchOutput, rollOutput);
}
```

U samoj *'loop*()' petlji (koja se izvršava kontinuirano) sadržana je logika za čitanja podataka sa senzora, izračun PID kontrolnih signala i ažuriranje izlaznih signala za motore. U prošlom potpoglavlju smo vidjeli čitanje kvaterniona iz FIFO spremnika, što je prošireno u ovoj iteraciji sa pretvaranjem istih u Eulerove kuteve koji su pohranjeni u polju *'euler'*.

Iako kvaternioni imaju broj matematičkih prednosti, Eulerovi kutevi imaju i svoje prednosti. U ovom slučaju, oni su puno intuitivniji za izrađivanje stabilizacijskog kontrolera u tri dimenzije i

često su primijenjeni kada se dizajniraju formule za takve algoritme. Također su često i jednostavniji za korištenje u primjenama koje zahtijevaju ljudsku interpretaciju. Koriste samo tri parametra (za razliku od 4 za kvaternione) što pojednostavljuje probleme i kalkulacije. Situacija je dodatno olakšana činjenicom da MPU-6050 podržava konverziju u Eulerove kuteve putem DMP procesora, što smo iskoristili s funkcijom *'dmpGetEuler(euler, &q)*'.

Nakon toga dohvaća se trenutno vrijeme u mikrosekundama, i izračunava vremenski interval *'dt'* u sekundama, koji iznosi razliku između prethodne takve kalkulacije i trenutnog trenutka. Nakon toga ažurira se prethodno vrijeme za sljedeći ciklus.

U idućoj sekciji izračunavaju se greške za nagib i valjanje koristeći primjerene vektore iz varijable eulerovih kuteva, tako što se oduzimaju od našeg željenog kuta nagiba (u ovom slučaju 0.0), kako bi se dobilo odstupanje. Nakon tog ova greška se integrira koristeći prethodno izračunate vremenske segmente *'dt'*, što predstavlja akumulaciju greške kroz vrijeme.

Derivativna greška za ove dvije osi se računa kao razlika trenutne greške i prethodne, u razmjeru s veličinom vremenskog segmenta *'dt'* za kojeg gledamo grešku.

Na poslijetku moguće je izračunati PID izlaz koristeći klasičnu formulu:

$$pidAxis(t) = Kp * (errorAxis) + Ki * (\int_0^t errorAxis * dt) + Kd * \frac{errorAxis - prevErrorAxis}{dt}$$

Gdje „Axis“ predstavlja os koju želimo kontrolirati (roll, pitch, itd.), a „t“ vrijeme. Kp, Ki i Kd predstavljaju konstante definirane na početku, i ukoliko su postavljene na 0.0 taj dio formule postaje 0 i izbacuje se iz izračuna, a ukoliko je velika vrijednost povećava se u važnosti. Iako postoje preporučene početne vrijednosti, za svaku aplikaciju moraju se eksperimentalno odrediti najbolje vrijednosti koje rezultiraju stabilnosti.

U *'previousError'* pohranjuje trenutačnu grešku za nagib i valjanje kao prethodnu grešku za sljedeći ciklus izračuna, nakon čega se rezultati PID izlaza za te dvije osi šalju funkciji 'runPID()' koja će dalje generirati PWM signale za upravljanje motorima. U sljedećem bloku je prikazana ova funkcija:

```
//Procesiranje PID-a
void runPID(float pitchOutput, float rollOutput){


  // Scale the PID outputs to match the expected input range of the motors
  float scaledPitchOutput = pitchOutput * 100; // Adjust the scaling factor as needed
  float scaledRollOutput = rollOutput * 100; // Adjust the scaling factor as needed


  // Postavi PWM-e pojedinih motora
  int motor1PWM = constrain(throttleInput + yawInput + forwardInput + rightInput - scaledPitchOutput + scaledRollOutput, 0, 1023);
  int motor2PWM = constrain(throttleInput - yawInput + forwardInput - rightInput - scaledPitchOutput - scaledRollOutput, 0, 1023);
  int motor3PWM = constrain(throttleInput + yawInput - forwardInput - rightInput + scaledPitchOutput - scaledRollOutput, 0, 1023);
  int motor4PWM = constrain(throttleInput - yawInput - forwardInput + rightInput + scaledPitchOutput + scaledRollOutput, 0, 1023);


  setPWM(motor1PWM,motor2PWM,motor3PWM,motor4PWM);
}
//Postavi izlaz pinova za motore
void setPWM(int PWM1, int PWM2, int PWM3, int PWM4){
analogWrite(motorPin, PWM1);
analogWrite(motorPin2, PWM2);
analogWrite(motorPin3, PWM3);
analogWrite(motorPin4, PWM4);
}
```

Unutar ove funkcije prvo skaliramo PID izlaze iz vrijednosti od 0.00 do 1.00, u vrijednosti do 100 (može se promijeniti po potrebi), kako bi se prilagodio očekivanom rasponu ulaznih signala za motore. Izračunavaju se PWM signali za motore, te smo ovu funkciju pobliže prikazali u podpoglavlju „PWM“, ali je sad modificirana da primi i nove signale.

Nadodaju se *'scaledPitchOutput'* i *'scaledRollOutput'* kako bi se dron automatski uvijek vratio u horizontalni položaj, te nadalje cijela linija je stavljena u *'constrain*()' metodu kako bi se ograničio maksimalni izlaz unutar 10-bitnih vrijednosti mikrokontrolera, odnosno od 0 do 1023. S ovim dron poprima zatvorenu kontrolnu petlju koja kontinuirano osigurava stabilnost.

7.3. Izrada android aplikacije

Sljedeći kod je pisan modularno u funkcijskim cjelinama, te će biti prezentiran kroz različite funkcionalne module. Unutar svake funkcionalne cjeline, prvo će biti prikazana klasa, te onda implementacije same klase u glavnoj izvršnoj metodi *'MainActivity*()'. Ona je početna točka izvršavanja svake Android aplikacije, to je podklasa klase '*Activity*' koja je odgovorna za stvaranje prozora u kojem se korisničko sučelje može vidjeti i s kojim se može upravljati.
Životni ciklus *'MainActivity*()' se sastoji od nekoliko metoda koje se pozivaju u različitim fazama izvršenja aplikacije. Ove metode uključuju:

- onCreate(): Metoda koja se poziva kada se aktivnost prvi put stvara.
- onStart(): Poziva se kada aktivnost postane vidljiva korisniku.
- onResume(): Poziva se kada aktivnost postane interaktivna s korisnikom.
- onPause(): Poziva se kada se trenutna aktivnost prekida.
- onStop(): Poziva se kada aktivnost više nije vidljiva korisniku.
- onDestroy(): Poziva se prije nego što se aktivnost uništi.

Ovaj životni ciklus detaljnije je opisan u službenoj dokumentaciji Android-a (Android, 2021.). Samo korisničko sučelje je definirano pomoću XML jezika u zasebnim datotekama. XML datoteke za raspored pohranjene su u direktoriju *'res/layout'* Android projekta. One definiraju strukturu, izgled i hijerarhiju UI komponenti, kao što su gumbi, tekstualni prikazi, prikazi slika i rasporedi koji organiziraju ove komponente. Definirana su im ograničenja, nazivi i ostali atributi kao što su njihova veličina, položaj, boja i ponašanje (Android, 2019.). Ovakav pristup odvaja prezentacijski sloj od logike i funkcionalnosti, čineći razvojni proces organiziranijim i održivijim. Ove datoteke se onda mogu referencirati u *'MainActivity*()', i pojedini elementi dobaviti i manipulirati pomoću različitih funkcija (Android, 2019.).

Slika 20. Prikaz korisničkog sučelja Android aplikacije (Izvor: Izrada autora)

7.3.1. Telemetrija

U ovom poglavlju, fokusirat ćemo se na specifičnu implementaciju telemetrije između drona i Android aplikacije. U kontekstu drona, telemetrija (ili „nadziranje na udaljenost“) omogućuje praćenje i analizu informacija o letjelici. Konkretne informacije koje dron pruža su bile definirane u prethodnom poglavlju, u sekciji programiranja mikrokontrolera. Prikazan je bio i način na koje su one učinjene dostupne pomoću servera, te je sad potrebno implementirati funkcionalnost koja će ih kontinuirano vraćati i prikazivati unutar tekstualnih elemenata. Zadatak je prvo započet implementacijom klase *'RetrieveTextTask'*:

```
public class RetrieveTextTask extends AsyncTask<String, Void, String> {
    private OnTextRetrieveListener listener;  // Slušatelj za događaj dohvaćanja teksta
    private int textViewIndex; // Indeks TextView-a koji će prikazati dohvaćeni tekst

    // Konstruktor koji postavlja indeks TextView-a i slušatelja
    public RetrieveTextTask(int textViewIndex, OnTextRetrieveListener listener) {
        this.textViewIndex = textViewIndex;
        this.listener = listener;
    }
    @Override
    protected void onPostExecute(String result) {
        // Pozivanje slušatelja s rezultatom
        if (listener != null) {
            listener.onTextRetrieved(textViewIndex, result);
        }
    }
    // Sučelje za slušatelja dohvaćanja teksta
    public interface OnTextRetrieveListener {
        void onTextRetrieved(int textViewIndex, String text);
    }
}
```

Ona nasljeđuje *'AsyncTask'* što znači da je dizajnirana za asinkrono izvršavanje, omogućuje izvršavanje operacija u pozadini, neovisno o glavnoj dretvi koja upravlja korisničkim sučeljem. Ovo je ključno za održavanje glatkog i odzivnog korisničkog iskustva, posebno u situacijama gdje se obavljaju dugoročne, kontinuirane operacije. Generički tipovi '<String, Void, String>

označavaju da je ulazni parametar pozadinskog zadatka tipa *'String'*, ne postoji ažuriranje o napretku (*'Void'*), i rezultat pozadinskog zadatke je također tipa *'String'*.

U konstruktoru inicijaliziramo parametre slušatelja i postavljamo varijablu člana *'textViewIndex'*. To omogućuje pozivatelju da navede koji će 'TextView' biti ažuriran i tko će biti obavješten kada je dohvaćanje teksta završeno.

Nakon što se pozadinski zadatak dovrši, rezultat se prosljeđuje metodi *'onPostExecute'*, koja se izvršava u glavnoj niti. Ovo omogućuje sigurno ažuriranje korisničkog sučelja s rezultatom, jer se korisničko sučelje može izravno mijenjati samo iz glavne dretve.

U njoj se slušatelj obavještava o rezultatu putem metode *'onTextRetrieved'*. Ona mora biti definirana od strane klase koja implementira sučelje, i bit će pozvana kada se tekst dohvati. To omogućuje modularnost i fleksibilnost, jer različite komponente mogu reagirati na rezultat asinkronog zadatka.

Na posljetku je definirano sučelje ' OnTextRetrieveListener' unutar klase. Svaka klasa koja želi biti obaviještena kada je dohvaćanje teksta završeno mora implementirati ovo sučelje.

U kontekstu ovog koda, metoda koja se izvršava u zasebnoj dretvi će biti *'doInBackground'* koju ćemo prikazati u idućem odjeljku.

```
    @Override
    protected String doInBackground(String... urls) {
        // URL za dohvaćanje teksta
        String urlString = urls[0];
        StringBuilder stringBuilder = new StringBuilder();  // Graditelj niza
za sastavljanje odgovora
        HttpURLConnection urlConnection = null; // Veza za HTTP zahtjev


// [..] nastavak u sljedećem bloku
```

Prvo se inicijalizira prvi URL iz proslijeđenog niza URL-ova koji će se koristiti za dohvaćanje teksta, i instanca klase *'StringBuilder'*, koja će se koristiti za sastavljanje odgovora kao niza znakova. Instanca klase *'HttpURLConnection'* je inicijalno je postavljena na *'null'*, koristiti će se za izvršavanje HTTP zahtjeva.

```
        try {
            URL url = new URL(urlString);
            urlConnection = (HttpURLConnection) url.openConnection();
            // Provjera je li veza uspješna
            if (urlConnection.getResponseCode() == HttpURLConnection.HTTP_OK)
{
                InputStream inputStream = urlConnection.getInputStream();
                BufferedReader bufferedReader = new BufferedReader(new
InputStreamReader(inputStream));
                String line;
                // Čitanje svake linije odgovora
                while ((line = bufferedReader.readLine()) != null) {
                    stringBuilder.append(line);
                }
                bufferedReader.close(); // Zatvaranje čitača
            } else {
                // Obrada slučaja kada veza nije uspješna
                return "Error: " + urlConnection.getResponseMessage();
            }
        } catch (IOException e) {
            e.printStackTrace();
            return "Error: " + e.getMessage();
        } finally {
            if (urlConnection != null) {
                urlConnection.disconnect(); // Prekidanje veze
            }
        }
        // Povratak dohvaćenog teksta
        return stringBuilder.toString();
    }
```

Zatim u *'try*()' bloku se stvara objekt klase *'URL'* stvorene iz ranije definiranog *'string'*-a koji predstavlja URL, te otvaramo vezu s njim što nam vraća objekt koji predstavlja tu vezu. Dalje se provjerava uspješnost te veze tako što se HTTP odgovor uspoređuje s konstantom *'HTTP_OK' (*odgovor 200*),* te se tek tada može ući u unutrašnju petlju.
Dohvaćamo ulazni tok podataka iz uspostavljene veze i stvaramo čitač tipa *'BufferedReader'* koji omogućuje čitanje teksta iz ulaznog toka. Deklarira se varijabla 'line' koja će čuvati svaku pročitanu liniju teksta. Čita liniju po liniju iz 'bufferedReader'-a i ako ona nije prazna (*null*), dodaje se u graditelj odgovora ' *stringBuilder'*, te na posljetku zatvaramo čitač, oslobađajući resurse. Ako veza nije bila uspješna, vraća se poruka o grešci s odgovarajućom porukom iz

odgovora, a ako se dogodi iznimka (npr. problem s mrežom), ispisuje se exception trace i vraća se poruka o grešci. U bloku *'finally'*, koji se uvijek izvršava, veza se prekida ako je bila otvorena i cijeli odgovor se pretvara u niz znakova i vraća kao rezultat metode. S tim, cijela klasa je definirana i preostaje samo pogledati implementaciju ove klase u glavnoj klasi *'MainActivity'*.

```
import androidx.appcompat.app.AppCompatActivity;
public class MainActivity extends AppCompatActivity implements
RetrieveTextTask.OnTextRetrieveListener {
    // TextView komponente za prikaz različitih podataka
    private TextView textTemp, textPres, textAlt, textDroneAlt;

    // Handler i Timer za planiranje zadataka
    private Handler handler;
    private Timer timer;
    private Runnable runnable;
```

Glavna klasa nasljeđuje *'AppCompatActivity'*, što je standardna klasa za podršku modernih Android značajki na različitim verzijama Androida, poput akcijske trake i fragmenta. Osim toga, *'MainActivity'* implementira sučelje *'RetrieveTextTask.OnTextRetrieveListener'*, što znači da će ova klasa morati implementirati metode definirane u tom sučelju.

Deklariraju se različite varijable za pohranu referenci koji će prikazivati različite podatke (*'TextView'*). Ove komponente koristit će se za prikaz različitih podataka kao što su temperatura (textTemp), tlak (textPres), nadmorska visina (textAlt) i visina drona (textDroneAlt).

'Handler' je Androidova klasa koja omogućuje zakazivanje izvršavanja koda koji će se pokrenuti u budućnosti. U ovom kontekstu, *'handler'* će biti korišten za ažuriranje *'TextView'* komponenti na glavnoj dretvi.

'Timer' je Java klasa koja se koristi za zakazivanje zadataka koji se trebaju izvršiti nakon određenog vremenskog perioda. U ovom slučaju, *'timer'* će biti korišten za periodično dohvaćanje podataka.

'Runnable' je Java sučelje koje se koristi za predstavljanje koda koji će se izvršiti. U ovom kontekstu, *'runnable'* će biti korišten za definiranje koda koji će *'handler'* izvršiti.

```
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        // Inicijalizacija TextView komponenti za prikaz podataka
            textTemp = findViewById(R.id.textTemp);
            textPres = findViewById(R.id.textPres);
            textAlt = findViewById(R.id.textAltitude);
            textDroneAlt = findViewById(R.id.textDroneAlt);

        // Inicijalizacija timera i handlera
        timer = new Timer();
        handler = new Handler();

        // Početak dohvaćanja teksta s navedenih URL-ova
        startRetrievingText("http://42.42.42.42/temp", 0);
        startRetrievingText("http://42.42.42.42/pres", 1);
        startRetrievingText("http://42.42.42.42/altitude", 2);
        startRetrievingText("http://42.42.42.42/droneAlt", 3);
```

U metodi *'onCreate*()' inicijaliziramo aktivnosti, i postavljamo izgled aktivnosti na definirani XML resurs. Pomoću *'findViewById'* se dohvaćaju reference na *'TextView'* komponente i izgled komponenti iz tog XML dokumenta.

Pozivanjem metode *'startRetrievingText'* (koja je prikazana u idućem odlomku) s različitim URL-ovima i indeksima, pokrećemo asinkroni zadatak za dohvaćanje teksta s navedenih URL-ova.

```
 // Metoda za početak dohvaćanja teksta s URL-a u fiksnom intervalu
    private void startRetrievingText(String url, int textViewIndex) {
        TimerTask task = new TimerTask() {
            @Override
            public void run() {
                RetrieveTextTask textTask = new
RetrieveTextTask(textViewIndex, MainActivity.this);
                textTask.execute(url);
            }
        };
        // Planiranje zadatka da se izvršava svake sekunde
        timer.scheduleAtFixedRate(task, 0, 1000);
    }
```

Metoda *'startRetrievingText'* prima parametre koji predstavljaju URL za dohvaćanje teksta i indeks *'TextView'* komponente koja će prikazati tekst. Stvara se anonimna unutarnja klasa koja nasljeđuje *'TimerTask'*, u kojoj se definira zadatak koji će se izvršavati u redovitim intervalima. Zatim se stvara instanca klase *'RetrieveTextTask'*, koja je prethodno definirana za asinkrono dohvaćanje teksta. U idućoj liniji se pokreće asinkroni zadatak s navedenim, proslijeđenim URL-om. *'timer'* planira zadatak da se izvršava svake sekunde (1000 milisekundi), i počinje odmah (kašnjenje od 0 milisekundi).

```
// Metoda za ažuriranje odgovarajuće TextView komponente s dohvaćenim tekstom
    public void onTextRetrieved(int textViewIndex, String text) {
        handler.post(new Runnable() {
            @Override
            public void run() {
                switch (textViewIndex) {
                    case 0:
                        textTemp.setText(text);
                        break;
                    case 1:
                        textPres.setText(text);
                        break;
                    case 2:
                        textAlt.setText(text);
                        break;
                    case 3:
                        textDroneAlt.setText(text);
                        break;
                }
            });}
```

U metodu *'onTextRetrieved'* se prosljeđuju parametri koji predstavljaju indeks *'TextView'* komponente i dohvaćeni tekst. *'new Runnable()*' postavlja izvršivi objekt u red glavne dretve. Ovo osigurava da se ažuriranje korisničkog sučelja izvršava u glavnoj dretvi, što je obavezno u Androidu. Unutar nje radimo operacije ažuriranja, odnosno ovisno o indeksu postavlja se tekst na odgovarajuću *'TextView'* komponentu. Ove dvije metode zajedno omogućuju redovito dohvaćanje teksta s određenih URL-ova i ažuriranje korisničkog sučelja s dohvaćenim tekstom. Ukratko, metoda *'startRetrievingText'* planira zadatak koji se izvršava svake sekunde, dok metoda *'onTextRetrieved'* ažurira odgovarajuću *'TextView'* komponentu u glavnoj dretvi. Ovaj kod služi kao most između asinkronog dohvaćanja podataka i njihovog prikaza u korisničkom sučelju, omogućujući dinamičko ažuriranje sučelja s vanjskim podacima.

7.3.2. Joystick element

U ovom poglavlju definirati ćemo klasu klase *'JoystickView'*, koja nasljeđuje Androidovu klasu *'View'*. Ova klasa služi za stvaranje prilagođenog joysticka koji se može koristiti kao upravljačke palice za interakciju s korisnikom. Idući odlomak koda definira osnovne postavke i metode za konfiguraciju joysticka, uključujući boje, stilove, središta, polumjere i raspon osi.

```
public class JoystickView extends View {
    // Boje i postavke za vanjski i unutarnji krug joysticka
    private Paint outerCirclePaint, innerCirclePaint;
    private PointF outerCircleCenter, innerCircleCenter; // Središte krugova
    private float outerCircleRadius, innerCircleRadius; // Polumjeri krugova
    private float normalizedX, normalizedY; //Normalizirane vrijednosti kruga
    private int maxYValue, minYValue, maxXValue, minXValue; // Raspon osi
    private int defaultCenterX, defaultCenterY; //Koordinate središta krugova
    // Slušatelji za događaje pokreta i otpuštanja joysticka
    private OnJoystickMoveListener onJoystickMoveListener;
    private OnJoystickReleaseListener onJoystickReleaseListener;
```

Unutar same klase definiraju se objekti klase *'Paint'* koji definiraju boju i stil za vanjski i unutarnji krug joysticka, i klase *'PointF'* koji predstavljaju središte vanjskog i unutarnjeg kruga i sadrže dvije '*float*' varijable. Definiraju se ostale pomoćne varijable, i na posljetku su sučelja koja omogućuju reagiranje na pokret i otpuštanje joysticka (*release* i *move* listeneri).

```
    // Postavljanje raspona osi joysticka
    public void setAxisRange(int maxXValue, int minXValue, int maxYValue, int
minYValue) {
        this.maxXValue = maxXValue;
        this.minXValue = minXValue;
        this.maxYValue = maxYValue;
        this.minYValue = minYValue;
    }
    // Postavljanje zadanih koordinata središta unutarnjeg kruga
    public void setDefaultCenter(int x, int y) {
        this.defaultCenterX = x;
        this.defaultCenterY = y;
        innerCircleCenter.set(outerCircleCenter.x+x,outerCircleCenter.y+ y);
}
```

Metoda *'setAxisRange'* postavlja raspon vrijednosti za X i Y os joysticka. Ovo omogućuje prilagodbu joysticka vrijednostima s kojima je najlakše raditi, odnosno koje su najprimjerenije za slanje mikrokontroleru. Unutar *'setDefaultCenter'* postavljaju se zadane koordinate središta unutarnjeg kruga. Ovo omogućuje prilagodbu početne pozicije unutarnjeg kruga.

```
// Inicijalizacija postavki joysticka
private void init() {
    // Postavljanje boja i stilova za krugove
    outerCirclePaint = new Paint();
    outerCirclePaint.setColor(Color.GRAY);
    outerCirclePaint.setStyle(Paint.Style.FILL);
    innerCirclePaint = new Paint();
    innerCirclePaint.setColor(Color.RED);
    innerCirclePaint.setStyle(Paint.Style.FILL);
    // Inicijalizacija središta i polumjera krugova
    outerCircleCenter = new PointF();
    outerCircleRadius = 200;
    innerCircleCenter = new PointF();
    innerCircleRadius = outerCircleRadius / 2;
    setDefaultCenter(0, 0);
}
```

Metoda *'init'* Inicijalizira postavke joysticka, uključujući boje, stilove, središta i polumjere krugova. Stvara novi objekt klase *'Paint'* koji će se koristiti za crtanje vanjskog kruga joysticka, te postavlja boju vanjskog kruga na sivu. Postavlja stil crtanja vanjskog kruga na „FILL“ što nači da će krug biti ispunjen bojom.

Također stvara novi objekt klase *'Paint'* i za unutrašnji krug, koji je ovog puta obojan crveno, te također ima „FILL“ svojstvo.

Postavlja vanjski krug na zadanu središnju poziciju tako što inicijalizira novi objekt klase *'PointF'* koji sadrži dvije vrijednosti *'float'* koje predstavljaju kordinatu u dvodimenzionalnom prostoru. Radijus vanjskog kruga se postavlja na 200 jedinica, te se zatim isto radi i sa unutrašnjim krugom, samo što je radijus unutrašnjeg kruga postavljen na polovicu vanjskog.

Korištenjem prethodno spomenute metode *'setDefaultCenter*(0,0)' se postavljaju zadane kordinate središta za oba kruga.

```
public boolean onTouchEvent(MotionEvent event) {
  int action = event.getAction();
  if (action ==MotionEvent.ACTION_MOVE || action == MotionEvent.ACTION_DOWN){
        // Izračun udaljenosti između točke dodira i središta vanjskog kruga
            float dx = event.getX() - outerCircleCenter.x;
            float dy = event.getY() - outerCircleCenter.y;
            float distance = (float) Math.sqrt(dx * dx + dy * dy);
            if (distance > outerCircleRadius) {
                float angle = (float) Math.atan2(dy, dx);
                innerCircleCenter.x = outerCircleCenter.x + outerCircleRadius
* (float) Math.cos(angle);
                innerCircleCenter.y = outerCircleCenter.y + outerCircleRadius
* (float) Math.sin(angle);
            } else {
                innerCircleCenter.x = event.getX();
                innerCircleCenter.y = event.getY();
            } invalidate(); // Ponovno iscrtavanje pogleda
     normalizedX=(innerCircleCenter.x-outerCircleCenter.x)/outerCircleRadius;
     normalizedY=(innerCircleCenter.y-outerCircleCenter.y)/outerCircleRadius;
     // Preslikavanje normaliziranih X i Y vrijednosti na željeni raspon
     int mappedX = (int) (normalizedX * (maxXValue - minXValue) + minXValue);
     int mappedY = (int) (normalizedY * (maxYValue - minYValue) + minYValue);
            // Obavijesti slušatelja o pokretu joysticka
            if (onJoystickMoveListener != null) {
                onJoystickMoveListener.onJoystickMove(mappedX, mappedY);
            } return true;
        } else if (action == MotionEvent.ACTION_UP) {
            innerCircleCenter.set(outerCircleCenter.x + defaultCenterX,
outerCircleCenter.y + defaultCenterY);
            invalidate();  // Ponovno iscrtavanje pogleda
            if (onJoystickMoveListener != null) {
                onJoystickMoveListener.onJoystickMove(0, 0);
            } // Obavijesti slušatelja da je joystick otpušten
            if (onJoystickReleaseListener != null) {
                onJoystickReleaseListener.onJoystickRelease();
            } return true;
}
        return super.onTouchEvent(event); }
```

Ovaj odlomak koda definira metodu *'onTouchEvent'* unutar klase *'JoystickView'*, koja se koristi za obradu događaja dodira na prilagođenom joysticku. Metoda se poziva svaki put kada korisnik dodirne ili pomakne joystick. Prvo dohvaća akciju dodira iz objekta *'MotionEvent'*, koji sadrži informacije o događaju dodira. Zatim provjerava je li akcija dodira "pomicanje" (*ACTION_MOVE*) ili "pritisak" (*ACTION_DOWN*), što znači da korisnik dodiruje ili pomiče joystick. Sljedeće je potrebno izračunati udaljenost između točke dodira i središta vanjskog kruga joysticka, što se koristi za određivanje je li točka dodira unutar ili izvan vanjskog kruga. Ako je točka dodira izvan vanjskog kruga, izračunava se kut i postavlja pozicija unutarnjeg kruga na rub vanjskog kruga. Ako je unutar, postavlja se pozicija unutarnjeg kruga na točku dodira. Pozivanjem metode *'invalidate'* da zatraži ponovno iscrtavanje pogleda, će ažurirati prikaz joysticka. Sljedeće normalizira X i Y vrijednosti unutarnjeg kruga u rasponu [-1, 1], i Preslikava normalizirane vrijednosti na željeni raspon, koji je postavljen prethodno spomenutom metodom *'setAxisRange'*.

Provjerava je li postavljen slušatelj za pokret joysticka, te ako je obavještava ga o novim preslikanim X i Y vrijednostima. Idući dio koda obrađuje slučaj kada korisnik otpusti dodir na joysticku, što je predstavljeno akcijom *'MotionEvent.ACTION_UP'*. Kada se dodir otpusti, pozicija unutarnjeg kruga vraća se na zadano središte. To se postiže postavljanjem koordinata unutarnjeg kruga na zadanu poziciju koja je prethodno postavljena metodom *'setDefaultCenter'*. Naredba *'invalidate*()' poništava prikaz kako bi se zatražilo ponovno iscrtavanje.

Ako je postavljen slušatelj za pokret, obavještava se s nultim vrijednostima za X i Y, što ukazuje na to da se joystick vratio u središnji položaj, a ako je postavljen slušatelj za otpuštanje, obavještava se da je joystick otpušten. To se može koristiti za pokretanje specifičnih akcija u aplikaciji kada se joystick otpusti. Na kraju metoda vraća *'true'*, što ukazuje da je događaj dodira obrađen. Ukratko ova metoda *'onTouchEvent'* osigurava da kada korisnik otpusti joystick, vrati se u zadani položaj, a svi slušatelji budu obaviješteni o otpuštanju. To omogućuje prirodnu i intuitivnu interakciju, gdje joystick reagira na dodir korisnika pomicanjem, a zatim se vraća u neutralni položaj kada se otpusti.

```
@Override
    protected void onSizeChanged(int w, int h, int oldw, int oldh) {
        // Izračun središta pogleda i ažuriranje pozicija krugova
        outerCircleCenter.set(w / 2f, h / 2f);
        innerCircleCenter.set(outerCircleCenter);
    }
    @Override
    protected void onDraw(Canvas canvas) {
        // Crtanje vanjskog kruga
        canvas.drawCircle(outerCircleCenter.x, outerCircleCenter.y,
outerCircleRadius, outerCirclePaint);

        // Crtanje unutarnjeg kruga
        canvas.drawCircle(innerCircleCenter.x, innerCircleCenter.y,
innerCircleRadius, innerCirclePaint);
    }
    public void setOnJoystickMoveListener(OnJoystickMoveListener listener) {
        this.onJoystickMoveListener = listener;
    }
    public void setOnJoystickReleaseListener(OnJoystickReleaseListener
listener) {
        this.onJoystickReleaseListener = listener;
    }
    public interface OnJoystickMoveListener {
        void onJoystickMove(int x, int y);
    }
    public interface OnJoystickReleaseListener {
        void onJoystickRelease();
    }
}
```

Metoda *'onSizeChanged'* se poziva kada se promjeni veličina pogleda (*view*), parametri w i h predstavljaju novu širinu i visinu pogleda, dok *'oldw'* i *'oldh'* predstavljaju staru širinu i visinu. Središte vanjskog kruga postavlja se na središte pogleda, a zatim se pozicija unutarnjeg kruga postavlja na istu poziciju kao i vanjski krug. Ovo osigurava da se joystick pravilno pozicionira unutar pogleda, čak i ako se veličina pogleda promijeni.

'onDraw' se poziva kada je potrebno iscrtati pogled. Koristi objekt tipa *'Canvas'* za crtanje dvaju krugova koji predstavljaju joystick: vanjski krug i unutarnji krug. Pozicije, polumjeri i boje krugova određeni su prethodno postavljenim svojstvima.

Iduće dvije metode omogućuju vanjskim klasama da postave slušatelje za događaje pokreta i otpuštanja joysticka. Slušatelji omogućuju vanjskim klasama da reagiraju na promjene u

joysticku, kao što su pokreti i otpuštanje. Nadalje su definirana dva sučelja koja definiraju metode koje moraju implementirati slušatelji. Sad kada su definirani svi elementi potrebni za izradu joystick-a, pogledajmo kako je ona implementirana u glavnoj aktivnosti.

```
import androidx.appcompat.app.AppCompatActivity;
public class MainActivity extends AppCompatActivity implements
RetrieveTextTask.OnTextRetrieveListener {

    // Deklaracija joystick komponenti za upravljanje dronom
    private JoystickView joystickViewLeft;
    private JoystickView joystickViewRight;

    // Varijable za pohranu vrijednosti joysticka
    private int throttle,yaw,forward,right=0;
    private boolean leftTouched;
    private Runnable runnable;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
     // Inicijalizacija joystick komponenti i postavljanje njihovih svojstava
        joystickViewLeft = findViewById(R.id.throttle_yaw_joystick);
        joystickViewRight = findViewById(R.id.spatial_joystick);
        joystickViewLeft.setAxisRange(125, 0, 0, 125);
        joystickViewRight.setAxisRange(125, 0, 125, 0);
        joystickViewLeft.setDefaultCenter(0,100);
        joystickViewRight.setDefaultCenter(0,0);
```

Prvo se deklariraju dva objekta koji predstavljaju dvije instance naše joystick klase koju smo prethodno definirali. Definiraju se 4 varijable koje služe kako bi pohranile vrijednosti osi koje joysticki generiraju. '*leftTouched*' je zastava (eng. *flag*) koja može biti korištena za praćenje stanja lijevog joysticka, kako se kasnije zahtjev za slanjem stanja dronu ne bi duplicirao, što će kasnije biti još objašnjeno. *'runnable'* je referenca na objekt koji se može koristiti za izvršavanje koda u pozadinskoj dretvi.

Kao što smo već spomenuli metoda *'onCreate'* poziva se kada se aktivnost prvi put stvori. U njoj se postavlja izgled aktivnosti prema definiranom XML resursu, a joystick komponente se inicijaliziraju pomoću metode *'findViewById'*, koja ih povezuje s odgovarajućim elementima u

XML dokumentu. Metode *'setAxisRange'* i *'setDefaultCenter'* konfiguriraju joystick komponente, postavljajući raspon osi i zadane centar koordinate.

```
// Postavljanje slušača za pokrete i otpuštanja joysticka
        joystickViewLeft.setOnJoystickMoveListener(new
JoystickView.OnJoystickMoveListener() {
            public void onJoystickMove(int x, int y) {
                setLeftJoystick(x,y);
                sendPWMcommand();
                leftTouched=true;
            }});
        joystickViewLeft.setOnJoystickReleaseListener(new
JoystickView.OnJoystickReleaseListener() {
            public void onJoystickRelease() {
                setLeftJoystick(0,100);
                sendPWMcommand();
                leftTouched=false;
            }});
        joystickViewRight.setOnJoystickMoveListener(new
JoystickView.OnJoystickMoveListener() {
            public void onJoystickMove(int x, int y) {
                    setRightJoystick(x,y);
                    if(!leftTouched){
                        sendPWMcommand();
                    }}});
        joystickViewRight.setOnJoystickReleaseListener(new
JoystickView.OnJoystickReleaseListener() {
            public void onJoystickRelease() {
                setRightJoystick(0,0);
                if(!leftTouched){
                    sendPWMcommand();
                }}});
    // Metode za postavljanje vrijednosti joysticka
    public void setLeftJoystick(int x, int y){
        this.throttle=y;
        this.yaw=x;
    }
    public void setRightJoystick(int x, int y){
        this.forward=y;
        this.right=x;
    }
}
```

Ovaj dio koda postavlja slušatelje (eng. *listeners*) za događaje pokreta i otpuštanja dva joysticka (*joystickViewLeft* i *joystickViewRight*) koji omogućuju aplikaciji da reagira na korisnikove interakcije s joystickima, interpretirajući pokrete i otpuštanja i pretvarajući ih u odgovarajuće naredbe.

Prvo se postavljaju slušatelji za lijevi joystick, kada se pomakne, metoda *'setLeftJoystick*(x,y)' postavlja vrijednosti *'throttle'* i *'yaw'* na nove koordinate. Zatim se poziva *'sendPWMcommand*()' za slanje odgovarajuće naredbe, a zastava *'leftTouched'* postavlja se na *'true'*. Kada se lijevi joystick otpusti, postavlja se na zadane vrijednosti, šalje se naredba, a zastava *'leftTouched'* postavlja se na *'false'*.

Na isti način postavlja se i desni joystick, s jednom razlikom. Naime kako će kasnije biti prikazano, pri slanju vrijednosti palica šalju se sve 4 varijable u jednom zahtjevu, a ne dva zahtjeva (svaki za jedan palicu), kako bi se smanjio ukupni broj zahtjeva napravljenih dronu, i tako smanjila latencija procesiranja.

Rezultat toga je da u jednom trenutku samo jedna palica mora pozivati metodu za slanje kako bi se sve vrijednosti poslale, što onda reguliramo pomoću zastave *'leftTouched'*, koja funkcionira kao jednostavni semafor. Na posljetku dvije metode postavljaju vrijednosti varijabli koje predstavljaju položaj joysticka, lijevi kontrolira gas i rotaciju oko osi, dok desni kontrolira horizontalnu translaciju drona kroz prostor. Preostaje slanje ovih vrijednosti dronu pomoću zahtjeva, za što nam treba HTTP klijent kojeg ćemo opisati u idućem poglavlju.

7.3.3. HTTP klijent

Sljedeći kod definira klasu *'HttpGetRequest'*, koja nasljeđuje *'AsyncTask'* i služi za izvršavanje asinkronih HTTP GET zahtjeva. Ova klasa omogućuje slanje HTTP zahtjeva u pozadinskoj dretvi, bez blokiranja glavne dretve koja upravlja korisničkim sučeljem.

```
public class HttpGetRequest extends AsyncTask<String, Void, String> {

    // Povratni poziv koji će se izvršiti nakon dovršetka zahtjeva
    private OnHttpRequestComplete callback;
    // Sučelje koje definira metodu za obradu odgovora na HTTP zahtjev
    public interface OnHttpRequestComplete {
        void onRequestComplete(String response);
    }
    // Metoda za postavljanje objekta koji implementira povratni poziv
    public void setCallback(OnHttpRequestComplete callback) {
        this.callback = callback;
    }
    @Override
    protected void onPostExecute(String result) {
        // Pozivanje povratnog poziva s rezultatom
        // Ova metoda se izvodi u glavnoj dretvi i može ažurirati korisničko
sučelje
        if (callback != null) {
            callback.onRequestComplete(result); // Pozivanje metode povratnog
poziva s rezultatom
        }
    }
}
```

Prvo se definira varijabla *'callback'* koja služi za pohranu objekta koji implementira povratni poziv. Povratni poziv će se izvršiti nakon što se HTTP zahtjev dovrši. Zatim se definira metoda koja će se pozvati kada se HTTP zahtjev dovrši, i Klasa koja koristi *'HttpGetRequest'* mora implementirati ovo sučelje kako bi mogla reagirati na dovršetak zahtjeva.
Metoda *'setCallback'* omogućuje postavljanje objekta koji implementira povratni poziv. *'onPostExecute'* se poziva nakon što se asinkroni zadatak dovrši. Ako je povratni poziv postavljen (nije 'null'), metoda *'onRequestComplete'* sučelja se poziva s rezultatom zahtjeva. Budući da se *'onPostExecute'* izvodi u glavnoj dretvi, može se koristiti za ažuriranje korisničkog sučelja.

```
    @Override
    protected String doInBackground(String... params) {
        String urlString = params[0];  // URL za HTTP zahtjev
        String parameter1 = params[1]; // Parametri za HTTP zahtjev

        // Ručno kodiranje znaka "=" kao "%3D" i zamjena "%26" s "&"
        String encodedParam1 = URLEncoder.encode(parameter1)
                .replace("%3D", "=")
                .replace("%26", "&");
        // Spajanje URL-a i parametara
        String urlWithParams = urlString + "?" + encodedParam1;
        System.out.println(urlWithParams); // Ispisivanje punog URL-a s
parametrima
        try {
            URL url = new URL(urlWithParams); // Kreiranje URL objekta
            HttpURLConnection connection = (HttpURLConnection)
url.openConnection();
            // Postavljanje metode zahtjeva na GET
            connection.setRequestMethod("GET");
            // Čitanje odgovora s pomoću BufferedReadera
            BufferedReader reader = new BufferedReader(new
InputStreamReader(connection.getInputStream()));
            StringBuilder response = new StringBuilder();
            String line;
            while ((line = reader.readLine()) != null) {
                response.append(line); // Dodavanje svake linije u odgovor
            }
            reader.close(); // Zatvaranje čitača

            return response.toString(); // Vraćanje odgovora kao stringa

        } catch (IOException e) {
            e.printStackTrace(); // Ispisivanje greške ako se dogodi
        }

        return null; // Vraćanje null ako se dogodi iznimka
    }
```

Metoda prvo prima niz gdje je prvi element URL, a drugi element su parametri za HTTP zahtjev. Parametri se moraju kodirati kako bi se osiguralo da su sigurni za uključivanje u URL. Ovdje se koristi *'URLEncoder.encode*()' za kodiranje parametara, a zatim se neki specifični znakovi zamjenjuju kako bi se postigao željeni format. Parametri se zatim dodaju URL-u s upitnikom (?) koji ih odvaja od osnovnog URL-a. Rezultirajući URL s parametrima ispisuje se u konzolu kako bi provjerili ispravnost. Kreira se *'URL'* objekt s punim URL-om i otvara se *'HttpURLConnection'*, te se metoda zahtjeva postavlja na "*GET*".

Odgovor se čita iz ulaznog toka pomoću *'BufferedReader'*, te se svaka linija odgovora dodaje *'StringBuilder'* objektu koji će sadržavati cijeli odgovor. Nakon tog zatvara se čitač i vraća cijeli odgovor kao *'string'*.

Ako se slučajem dogodila iznimka *'IOException'*, ispisuje se trag greške. Ako se dogodi iznimka, metoda vraća *'null'*. Pogledajmo kako je ova klasa implementirana u glavnoj aktivnosti:

```
import androidx.appcompat.app.AppCompatActivity;
public class MainActivity extends AppCompatActivity implements
RetrieveTextTask.OnTextRetrieveListener {

    // Varijable za pohranu vrijednosti joysticka
    private int throttle,yaw,forward,right=0;
    private boolean leftTouched;
    private Runnable runnable;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);

    // Metoda za slanje naredbe dronu
    private void sendPWMcommand(){
        HttpGetRequest getRequest = new HttpGetRequest();
        getRequest.execute("http://42.42.42.42/motors", "Y=" + throttle +
"&X=" + yaw + "&F="+ forward + "&R=" + right);
    }
}
```

Prvo se deklariraju potrebne varijable, prva četiri *'int'*-a su dijelovi joystick funkcionalnosti te se koriste za spremanje vrijednosti koje želimo poslati, pa su uključene i ovdje. Kao što smo već upoznati, Metoda *'onCreate'* poziva se pri stvaranju aktivnosti i služi za inicijalizaciju komponenti, i postavlja se sadržaj pogleda na *'activity_main'* XML datoteku koja definira izgled korisničkog sučelja.
Privatna metoda *'sendPWMcommand'* služi za slanje naredbe dronu putem HTTP GET zahtjeva. URL i parametri za zahtjev formiraju se na temelju trenutnih vrijednosti joysticka. Sama naredba '.*execute*()' je dio *'AsyncTask'* klase koju smo implementirali u *'HttpGetRequest'* klasi, pa je ona povezana sa *'doInBackground*()' metodom u njoj.

7.3.4. Snaga signala

RSSI (eng. *Received Signal Strength Indicator*) je pokazatelj koji prikazuje snagu signala u decibelima (*dBm*) i može pružiti uvid u kvalitetu i stabilnost Wi-Fi veze. RSSI vrijednost može varirati od -30 dBm do -100 dBm. Vrijednost bliža 0 ukazuje na jači signal, dok niža vrijednost ukazuje na slabiji signal.

- Izvrsna veza: -30 do -50 dBm
- Dobra veza: -50 do -70 dBm
- Slaba veza: -70 do -90 dBm
- Blizu prekida veze: -90 do -100 dBm

Kada RSSI vrijednost padne ispod -90 dBm, veza postaje nestabilna i blizu je prekida. Ovo može rezultirati sporim brzinama prijenosa podataka, prekidima u vezi i potpunim gubitkom veze.
Kako bi pilot drona mogao procijeniti rizik gubitka kontrole nad dronom, ova informacija se prikazuje kodirana bojom i postotkom na korisničkom sučelju. Pogledajmo implementaciju ovog modula:

```
public class WifiSignalStrengthUpdater {
    private Context context; // Kontekst aplikacije ili aktivnosti
    private Handler handler; // Handler za izvršavanje koda u glavnoj niti
    private OnSignalStrengthChangeListener listener; // Slušatelj za promjene
jačine signala

    // Konstruktor koji postavlja kontekst i slušatelja
    public WifiSignalStrengthUpdater(Context context,
OnSignalStrengthChangeListener listener) {
        this.context = context;
        this.listener = listener;
        // Inicijalizacija handlera za glavnu nit
        this.handler = new Handler(Looper.getMainLooper()); }

    // Metoda za pokretanje ažuriranja jačine signala
    public void startUpdates() {
        handler.post(new Runnable() {
            @Override
            public void run() {
                // Dohvaćanje prilagođene jačine signala
                int signalStrength = getRescaledWifiSignalStrength();
                if (listener != null) {
                    // Obavještavanje slušatelja o promjeni
                    listener.onSignalStrengthChanged(signalStrength);
                }
                // Ponovno postavljanje koda za izvršavanje nakon 1 sekunde
                handler.postDelayed(this, 1000);
}});}}
```

U glavnoj definiciji klase *'WifiSignalStrenghtUpdater'* prvo definiramo *'context'* , što je ključna klasa koja pruža pristup informacijama o okolini aplikacije i omogućuje interakciju s raznim sustavnim servisima. Inicijaliziramo *'handler'* koji se koristi za zakazivanje i izvršavanje koda koji se treba pokrenuti u određenoj niti, često u glavnoj dretvi koja upravlja korisničkim sučeljem. Na posljetku definiramo *'listener'* koji omogućuje definiranje ponašanja koje će se dogoditi kada se promijeni jačina signala. Sve to prosljeđujemo u konstruktor klase *'WifiSignalStrengthUpdater'* koji postavlja varijable člana na vrijednost koja je proslijeđena konstruktoru. To omogućuje ostalim metodama unutar klase da pristupe kontekstu i slušatelju koji je proslijeđen prilikom stvaranja objekta. U sljedećoj liniji varijabla člana *'handler'* je povezana s novostvorenim objektom tipa *'Handler'*, što omogućuje klasi da zakazuje i izvršava kod unutar glavne niti.

Metoda *'startUpdates'* služi za pokretanje ažuriranja jačine Wi-Fi signala. U varijablu *'signalStrength'* se pohranjuje rezultat metode mjerenja i skaliranja snage signala, koja će biti opisana u sljedećem dijelu koda. Zatim se provjerava je li slušatelj postavljen, i ako je poziva metodu *'onSignalStrengthChanged'* na slušatelju, prosljeđujući joj trenutnu jačinu signala. Ovo omogućuje objektu koji je implementirao sučelje *'OnSignalStrengthChangeListener'* da reagira na promjenu signala. Na posljetku, ovaj cijeli kod se izvršava svakih 1000 milisekundi, što omogućuje periodično ažuriranje jačine signala svake sekunde.

```
    // Privatna metoda za dohvaćanje prilagođene jačine Wi-Fi signala
    private int getRescaledWifiSignalStrength() {
        WifiManager wifiManager = (WifiManager)
context.getSystemService(Context.WIFI_SERVICE);
        // Dohvaćanje informacija o Wi-Fi vezi
        WifiInfo wifiInfo = wifiManager.getConnectionInfo();

        if (wifiInfo != null) {
            // Dohvaćanje sirove jačine signala
            int signalStrength = wifiInfo.getRssi();
            // Prilagodba jačine signala na raspon od 0 do 100
            int rescaledSignalStrength = 100 - ((Math.abs(signalStrength) -
30) * 100) / 50;
            // Ograničavanje prilagođene jačine signala na raspon od 0 do 100
            if (rescaledSignalStrength < 0) {
                rescaledSignalStrength = 0;
            } else if (rescaledSignalStrength > 100) {
                rescaledSignalStrength = 100;
            }
            // Povratak prilagođene jačine signala
            return rescaledSignalStrength;
        }
        // Povratak 0 ako informacije o Wi-Fi vezi nisu dostupne
        return 0;
    }
```

Funkcija *'getRescaledWifiSignalStrength*()' pretvara snagu signala između vrijednosti. Zbog potencijalnog manjka znanja pilota o značenjima vrijednosti u jedinici dBm, ova vrijednost je prevedena u postotak i boju koja poboljšava korisničko iskustvo. Inicijalizira se novi objekt standardne klase *'WifiManager'* koja pruža primarni ulaz za upravljanje svim aspektima Wi-Fi povezivanja. To uključuje pregled dostupnih Wi-Fi mreža, povezivanje s mrežama, promjenu Wi-Fi stanja i mnoge druge operacije. Za dobivanje instance te klase, pozivamo metodu *'getSystemService'* s argumentom koji identificira željenu uslugu, u ovom slučaju, Wi-Fi uslugu.

'context' u ovoj liniji koda predstavlja kontekst aplikacije ili aktivnosti što pruža pristup resursima, postavkama ili uslugama sustava. Rezultat poziva na *'getSystemService'* je tipa *'Object'*, pa ga je potrebno pretvoriti (eng. *cast*) u *'WifiManager'*. Ovo se postiže korištenjem sintakse „(WifiManager)“, što osigurava da objekt ima pravi tip za daljnje operacije.
Metoda *'getConnectionInfo'* je članska metoda klase *'WifiManager'* i koristi se za dohvaćanje objekta *'wifiInfo'* koji predstavlja trenutnu Wi-Fi vezu. Ova metoda ne zahtijeva nikakve argumente i vraća objekt koji sadrži sve relevantne informacije o trenutnoj Wi-Fi vezi. *'wifiManager'* u ovoj liniji koda predstavlja instancu klase *'WifiManager'*, koja je prethodno dohvaćena (kao što je objašnjeno u prethodnom dijelu).
Rezultat poziva pohranjuje se u varijablu *'wifiInfo'*, koja je tipa *'WifiInfo'*. Ova varijabla sada sadrži objekt koji pruža pristup raznim informacijama o trenutnoj Wi-Fi vezi, uključujući, ali ne ograničavajući se na, snagu signala, brzinu veze i druge tehničke detalje.
Korištenjem metode *'getRssi*()' dohvaćamo informacije o snazi signala trenutno povezane Wi-Fi mreže. Vrijednost je pohranjena u varijablu *'signalStrength'* koja se u idućem koraku skalira kako bi dobili reprezentaciju snage od 0 do 100, gdje je 100 najjači signal. Iduće dvije petlje osiguravaju da je vrijednost ograničena unutar ovih vrijednosti za bilo koju skaliranu vrijednosti vraćenu s *'getRssi*()'. Naime, iako tipično vrijednosti variraju od -30 do -100 dBm, moguće je da je vraćena vrijednosti i izvan tog raspona, što stavlja i skaliran rezultat izvan raspona od 0 do 100. Nakon toga, ova vrijednost se može vratiti ukoliko je uspješno izračunata.

```
    // Sučelje za slušatelja promjene jačine signala
    public interface OnSignalStrengthChangeListener {
        void onSignalStrengthChanged(int signalStrength);
    }
// Metoda za zaustavljanje ažuriranja jačine signala
    public void stopUpdates() {
        // Uklanjanje svih zakazanih izvršavanja iz handlera
        handler.removeCallbacksAndMessages(null);
    }
```

Posljednje dvije funkcije predstavljaju dio koda koji se može koristiti za praćenje (slušanje) i upravljanje (zaustavljanjem) ažuriranja jačine Wi-Fi signala unutar glavne aktivnosti.

```
import androidx.appcompat.app.AppCompatActivity;
public class MainActivity extends AppCompatActivity implements
RetrieveTextTask.OnTextRetrieveListener {
    // Handler i Timer za planiranje zadataka
    private Handler handler;
    private Timer timer;
    // TextView i ažuriranje za prikaz jačine WiFi signala
    private TextView signalStrengthTextView;
    private WifiSignalStrengthUpdater signalStrengthUpdater;
    private Runnable runnable;
    @Override

    protected void onCreate(Bundle savedInstanceState) {
// Korištenje „AppCompatActivity“, pruža akcijsku traku i podršku fragmenta
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        // Inicijalizacija timera i handlera
        timer = new Timer();
        handler = new Handler();
        // Inicijalizacija komponenti za jačinu WiFi signala
        signalStrengthTextView = findViewById(R.id.signalStrengthTextView);
        signalStrengthUpdater = new
WifiSignalStrengthUpdater(getApplicationContext(), new
WifiSignalStrengthUpdater.OnSignalStrengthChangeListener() {
            @Override
            public void onSignalStrengthChanged(int signalStrength) {
                updateSignalStrengthTextView(signalStrength);
            }
        });
        signalStrengthUpdater.startUpdates();
}
```

U glavnoj klasi prikazano je korištenje modula za prikaz snage signala. Inicijaliziraju se potrebne varijable, nakon čega pozivamo metodu *'onCreate*()' koja se poziva kada se aktivnost prvi put stvori. Ovdje je prvo postavljena aktivnost prosljeđujući *'savedInstanceState'* i postavlja se izgled aktivnosti koristeći XML resurs definiran u *'activity_main'*. Povezuje se varijabla ' *signalStrengthTextView*' s ID-om definiranim u navedenom XML resursu. Inicijaliziranjem objekt za praćenje jačine signala ' *signalStrengthUpdater*', postavlja slušatelja koji će ažurirati *'TextView'* koji je prikazan na sučelju. Na posljetku pokreće se ažuriranje snage signala pomoću metode *'startUpdates*()'. Pogledajmo kako se samo ažuriranje vrijednosti UI elementa izvršava kroz iduće tri metode:

```
    // Metoda za ažuriranje TextView komponente jačine signala
    private void updateSignalStrengthTextView(int signalStrength) {
        signalStrengthTextView.setText("Snaga signala: "+signalStrength+"%");
        // Izračunaj boju na osnovi snage signala
        int color = getColorForSignalStrength(signalStrength);
        signalStrengthTextView.setTextColor(color);
    }
    // Metoda za dobivanje boje na temelju vrijednosti jačine signala
    private int getColorForSignalStrength(int signalStrength) {
        float hue = (signalStrength / 100f) * 120; // Izračunaj nijansu (od
crvene do zelene)
        return Color.HSVToColor(new float[]{hue, 1f, 1f});
    }

// Prepisivanje metode onDestroy kako bi se zaustavila ažuriranja jačine
signala
    @Override
    protected void onDestroy() {
        super.onDestroy();
        signalStrengthUpdater.stopUpdates();
    }
   }
```

Prva metoda postavlja tekst *'string'* koristeći proslijeđenu jačinu signala (u postotcima). Izračunava se boja na temelju jačine signala i pohranjuje u varijabli *'color'*, koja se zatim dodjeljuje samom tekstualnom elementu u idućoj liniji pomoću ugrađene metode *'setTextColor'*.

Ova boja se izračunava tako dobijemo nijansu na osnovi mapiranja jačine signala na raspon nijansi od crvene do zelene (0-120 u HSV boji). I idućoj liniji *'HSVtoColor()'* pretvara izračunatu nijansu u boju koja se može koristiti u Androidu, koristeći punu zasićenost i vrijednost (*1f*).

'onDestroy()' metoda je jedna od spomenutih metoda životnog ciklusa koja se poziva kada se aktivnost uništi. U njoj je pozvana metoda *'stopUpdates*()' što sigurno zaustavlja ažuriranje jačine signala i de-alocira memorijske resurse.

8. Zaključak

Ovaj rad se bavio složenim i multidisciplinarnim zadatkom projektiranja i implementacije kvadkopter drona, uključujući i sustave za upravljanje istim putem Wi-Fi-a, uz uporabu Android aplikacije i različitih Arduino modula. Rad se protezao od razmatranja teoretskih osnova leta kvadkoptera, preko hardverskih i softverskih aspekata, sve do praktične implementacije i analize sustava.
Izloženi su teoretski koncepti dinamike leta kvadkoptera, uključujući aerodinamičke sile i momente, te način na koji različite komponente integriraju kako bi se postigao kontroliran let. Proučeni su i implementirani različiti Arduino moduli, IMU i barometrički senzor, koristeći specifične komunikacijske protokole kao što su I2C i Wi-Fi. Dizajnirana je inačica elektroničkog brzinskog kontrolera (ESC-a) za četkaste motore bez jezgre prilagođena za uporabu u ovom dronu. Dizajniran je i prilagođena tiskana pločica (PCB) za dron, što je omogućilo integraciju svih potrebnih komponenti na kompaktni i efikasan način. Razvijena je i Android aplikacija koja omogućuje bežično upravljanje dronom putem pametnog telefona ili tableta. Kako bi se ovo ostvarilo, korišteni su brojni alati i aplikacije, uključujući Arduino IDE za programiranje mikrokontrolera, Android Studio za razvoj mobilne aplikacije, te KICAD za dizajniranje PCB-a.

Rezultati ovog istraživanja demonstrirali su uspješnu integraciju različitih tehnologija i disciplina u stvaranju funkcionirajućeg kvadkoptera. Postoji potencijal za daljnji razvoj i primjenu u različitim kontekstima. Zbog prilagodljivog i modularnog dizajna, buduća istraživanja mogu se fokusirati na dodatak dodatnih modula za specifične primjene, kao što su kamere i LiDAR, te razvoj algoritama za realno-vremensku obradu podataka može omogućiti bogatije interakcije s okolinom i precizniju navigaciju. Nadalje zbog niske cijene samog drona, postoji potencijal za praktičnu implementaciju automatizirane kontrole pomoću umjetne inteligencije rojeva. Kako bi se ove mogućnosti implementirane, niska snaga mikroprocesora se može nadoknaditi visokim performansama današnjih Android mobilnih uređaja, koje smo koristili kao upravljački uređaj. Nadalje ovakav uređaj predstavlja i potencijalni edukacijski alat u upoznavanju učenika s različitim disciplinama inženjerstva, kroz praktičan proces izgradnje ovakvog drona. Jedan od dodatnih funkcionalnosti koje se mogu implementirati u ovakvom kontekstu je recimo i automatsko namještanje PID konstanti kroz razne optimizacijske algoritme, i evaluacija istih.

Popis literature

Chao, H., Cao, Y., & Chen, Y. Q. (2010). Autopilots for small unmanned aerial vehicles: A survey. *International Journal of Control, Automation and Systems,.*

Miličević, Zoran & Bojković, Zoran. (2021). „From the early days of 962 unmanned aerial vehicles (UAVs) to their integration into wireless networks.“ *Vojnotehnički glasnik.*

Swopes, B. (2023) : „L'hélicoptère N°2“ This Day In Aviation
Preuzeto sa: https://www.thisdayinaviation.com/tag/lhelicoptere-n2/ Dostupno 22.7.2023

National Air and Space Museum (2019) " Gyrodyne QH-50C Drone Anti-Submarine Helicopter (DASH)." *National Air and Space Museum.*
Preuzeto sa https://airandspace.si.edu/collection-objects/gyrodyne-qh-50c-drone-anti-submarine-helicopter-dash/nasm_A20090023000 Dostupno 21.7.2023.

Elmenreich, W. (2002). Sensor Fusion in Time-Triggered Systems, *Vienna University of Technology.*

Piszcz, A. (2004). „Operational experiences with high-power lithium-ion batteries.“
23rd International Communications Satellite Systems Conference and Exhibit.

Svensson, P., (2013). "Review: Phantom quadcopter a fun consumer drone" *Phys.org*
Preuzeto sa: https://phys.org/news/2013-08-phantom-quadcopter-fun-consumer-drone.html Dostupno 22.7.2023

Makar, K., Shehata, M., Hafez, M., & Dessouky, M. (2018). "Towards Fully Autonomous Quadcopters: A Survey." *International Journal of Advanced Robotic Systems.*

Anderson, K., & Gaston, K. J. (2013).“Lightweight unmanned aerial vehicles will revolutionize spatial ecology.“ *Frontiers in Ecology and the Environment*

Nitha, M & Syeeda, S. (2022.). „Applications of drone technology in construction projects: A systematic literature review.“ *International Journal of Research – granthaalayah*

Torres-Sánchez, J., López-Granados, F., & Peña, J. M. (2014). „An automatic object-based method for optimal thresholding in UAV images: Application for vegetation detection in herbaceous crops.“ *Computers and Electronics in Agriculture*

Bork, Edward & Su, Jason. (2007). „Integrating LIDAR data and multispectral imagery for enhanced classification of rangeland vegetation: A meta analysis.“ *Remote Sensing of Environment.* 111. 11-24.

Clarke, R. (2014). „Understanding the Drone Epidemic.“ *Computer Law & Security Review, 230-246.*

European Union Regulations (2019). „Commission Implementing Regulation (EU) 2019/947. „ *Official Journal of the European Union.*

CCAA (2023.). „Često postavljana pitanja“. *Croatian Civil Aviation Agency.*

Bouabdallah, S. (2007). „Design and Control of Quadrotors with Application to Autonomous Flying.“, *École Polytechnique Fédérale de Lausanne.*

Kendoul, F. (2012). „Survey of Advances in Guidance, Navigation, and Control of Unmanned Rotorcraft Systems.“ *Journal of Field Robotics*, 29(2), 315-378.

Lupashin, S., et al. (2010). „A Simple Learning Strategy for High-Speed Quadrocopter Multi-Flips.“ *In Proceedings of the IEEE International Conference on Robotics and Automation.*

Zufferey, J. C., Klaptocz, A., & Floreano, D. (2013). „Energy-efficient flight control for a robotic bird.“ *International Journal of Micro Air Vehicles.*

Krajník, T., et al. (2011). „AR-Drone as a Platform for Robotic Research and Education.“ *In Proceedings of the International Conference on Research and Education in Robotics - EUROBOT.*

Espressif, (2023.) "ESP8266 Non-OS SDK API Reference". *Espressif Systems.* Preuzeto sa : https://www.espressif.com/sites/default/files/documentation/2c-esp8266_non_os_sdk_api_reference_en.pdf Dostupno dana 24.7.2023

Kolban, N. (2016). „Kolban's Book on the ESP8266 & ESP32.“ *Leanpub.com* Preuzeto sa: https://ullisroboterseite.de/esp8266-faq/ESP8266_ESP32-oct2016.pdf Dostupno dana 24.7.2023

Mishra, M., Dubey, V., Mishra, P., & Khan, I. (2019). „MEMS Technology: A Review.“ *Journal of Engineering Research and Reports, 4.*

Yang, H., Zhou, B., Wang, L., Xing, H., & Zhang, R. (2018). „A Novel Tri-Axial MEMS Gyroscope Calibration Method over a Full Temperature Range.“ *Sensors*.

Brossard, M., & Bonnabel, S. (2019). „Learning Wheel Odometry and IMU Errors for Localization.“ *Proceedings of the 2019 International Conference on Robotics and Automation (ICRA).*

Su, H., Iordachita, I., Tokuda, J., Hata, N., Liu, X., Seifabadi, R., Xu, S., Wood, B., & Fischer, G. (2017). „Fiber-Optic Force Sensors for MRI-Guided Interventions and Rehabilitation: A Review.“ *IEEE Sensors Journal.*

Poulose, A., & Han, D. (2019). „Hybrid Indoor Localization Using IMU Sensors and Smartphone Camera.“ *IEEE Sensors Journal.*

Zhao, S., Zhang, H., Wang, P., Nogueira, L., & Scherer, S. (2021). „Super Odometry: IMU-centric LiDAR-Visual-Inertial Estimator for Challenging Environments.“ *Proceedings of the 2021 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS).*

Shahzad, W., Ayaz, Y., Khan, M. J., Naseer, N., & Khan, M. (2019). „Enhanced Performance for Multi-Forearm Movement Decoding Using Hybrid IMU–sEMG Interface.“ *Frontiers in Neurorobotics, 43.*

Lawhorn, D., Han, P., Lewis, D. D., Chulaee, Y., & Ionel, D. (2021). „On the Design of Coreless Permanent Magnet Machines for Electric Aircraft“ *IEEE Propulsion.*

InvenSense (2023.) „MPU-6050 Six-Axis (Gyro + Accelerometer) MEMS MotionTracking™ Devices“ *TDK Invensense.* Preuzeto sa*: https://invensense.tdk.com/products/motion-tracking/6-axis/mpu-6050/* Dostupno 25.7.2023

Rafiq, A., Rohman, W. N., & Riyanto, S. (2020). „Development of a Simple and Low-cost Smartphone Gimbal with MPU-6050 Sensor.“ Journal of Robotics and Control (JRC)

Bosch Sensortec. (2020). „BME280: Combined humidity and pressure sensor.“ Preuzeto sa: https://www.bosch-sensortec.com/media/boschsensortec/downloads/datasheets/bst-bme280-ds002.pdf Dostupno 27.7.2023.

Beard, R.W., & McLain, T.W. (2012). „Small Unmanned Aircraft: Theory and Practice.„ *Princeton University Press.*

NXP Semiconductors. (2021). „UM10204: I2C-bus specification and user manual.“ *NXP Semiconductors.* Preuzeto sa: https://www.nxp.com/docs/en/user-guide/UM10204.pdf Dostupno dana 28.7.2023.

IEEE. (2020). „IEEE 802.11-2020 - IEEE Standard for Information technology—Telecommunications and information exchange between systems Local and metropolitan area networks--Specific requirements - Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications.“ *IEEE Journal.*

Qin, Z., Denkilkian, N., Wang, Y., & Chuah, M. C. (2017). „Securing UAV Communication over 5G Network.“ *2017 IEEE 7th Annual Computing and Communication Workshop and Conference (CCWC)*, 1-6.

Alazab, A., Awajan, A., Mesleh, A., & Sarra, S. (2020). „Mobile wireless sensor networks overview.“ *Mobile Sensor Networks: Architecture and Applications. CRC Press.*

Anan, S. R., Hossain, M. A., Milky, M. Z., Khan, M. M., Masud, M., & Aljahdali, S. (2021). „Research and Development of an IoT-Based Remote Asthma Patient Monitoring System.“ *Journal of Healthcare Engineering*

Monk, S. (2013). „Programming Arduino: Getting Started with Sketches.“ McGraw Hill Professional.

McRoberts, M. (2011). „Arduino Starter Kit Manual.“ Earthshine Electronics.

Kicad, (2023.) „Kicad.“ *Kicad.org* Preuzeto sa: https://docs.kicad.org/7.0/en/kicad/kicad.pdf dostupno dana 28.7.2023

Hammoodi, S. J., Flayyih, K. S., & Hamad, A. R. (2020). „Design and implementation speed control system of DC Motor based on PID control and matlab simulink.“ *IAES Journal.*

Winslow, J., Benedict, M., Hrishikeshavan, V., & Chopra, I. (2016). „Design, development, and flight testing of a high endurance micro quadrotor helicopter.“ *International Journal of Micro Air Vehicles*

Green, C. R. (2015). „Modeling and test of the efficiency of electronic speed controllers for brushless DC motors.“ *California State Polytechnic University.*

Yazdanpanah, R., & Mirsalim, M. (2015). „Design of robust speed and slip controllers for a hybrid electromagnetic brake system.“ *IET Electric Power Applications.*

T. Porselvi, S. Y. Aouthithiye Barathwaj, S. G. CS, S. V. Tresa Sangeetha and J. Shalini Priya, „Deep Learning Based Predictive Analysis of BLDC Motor Control“ *2022 IEEE 3rd Global Conference for Advancement in Technology (GCAT)*

Johnson, H. W. (2012). „*High-speed digital design: A handbook of black magic*.“ *Prentice Hall.*

Android (2021.) „The activity lifecycle“ *developer.android.com*
preuzeto sa : https://developer.android.com/guide/components/activities/activity-lifecycle dostupno dana 18.8.2023

Android (2019.) „Layouts in Views“ *developer.android.com*
preuzeto sa : https://developer.android.com/develop/ui/views/layout/declaring-layout
dostupno dana 18.8.2023

Popis slika

Slika 1. Oehmichen no.2 (Izvor: Swopes, B., 2023.) 3
Slika 2. Tri osnovne kontrolne osi kvadkoptera (Izvor: Izrada autora) 5
Slika 3. LIDAR snimka šume, s algoritamski odvojenom krošnjom od tla (Izvor: Anderson & Gaston, 2013.) 8
Slika 4. Prikaz korisničnog sučelja KiCad EDA (Izvor: kicad.org) 12
Slika 5 Prikaz ESP8266 razvojne pločice (Izvor: Kolban, 2016.) 16
Slika 6: Prikaz MPU-6050 modula (Izvor: InvenSense, 2023.) 23
Slika 7. Prikaz BME280 modula (Izvor: Amazon.com) 24
Slika 8. I2C pinovi na ESP8266 (Izvor: ElectronicsHub.com) 26
Slika 9. Shematski prikaz I2C sabirnice (Izvor: Medium.com, 2022) 26
Slika 10. Shematski prikaz glavnih komponenti drona (Izvor: Izrada autora) 28
Slika 11. Shematski prikaz ESC-a i asociranih komponenti (Izvor: Izrada autora) 30
Slika 12. Simulacija ESC-a u LTspice (Izvor: Izrada autora) 31
Slika 13. Fizička realizacija tiskane pločice drona (Izvor: Izrada autora) 33
Slika 14. Računalno generirani 3D prikaz izgleda drona (Izvor: Izrada autora) 34
Slika 15. Struktura funkcije u Arduinu (Izvor: izrada autora) 35
Slika 16. Prikaz pristupa poslužitelju (Izvor: Izrada autora) 39
Slika 17. Prikaz funkcionalnosti BME280 modula (Izvor: Izrada autora) 42
Slika 18. Prikaz kvaterniona nakon uspješne kalibracije (Izvor: Izrada autora) 51
Slika 20. Prikaz korisničkog sučelja Android aplikacije (Izvor: Izrada autora) 56

Popis tablica

Tablica 1. Prednosti i nedostatci različitih tipova FCU 14
Tablica 2. Wi-Fi parametri ESP8266 17
Tablica 3. Prednosti i nedostatci motora korištenih za kvadkoptere 19
Tablica 4. Prednosti i nedostatci različitih tipova IMU 22