

SVEUČILIŠTE U ZAGREBU  
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

**Bežični sustav za praćenje ljudskog  
pokreta u stvarnom vremenu korištenjem  
IMU senzora**

Lovro Šantek

Zagreb, srpanj 2024.



## Sadržaj

1.	Uvod .....	1
2.	Opis sustava.....	2
3.	Razvojni sustav za mikrokontroler ESP32 .....	4
3.1.	Osnovne karakteristike ESP32 mikrokontrolera i ESP32-WROOM-32 modula..	4
3.2.	Razvojna pločica ESP32-DevKit V1 .....	7
4.	Inercijski senzor.....	9
4.1.	Načelo rada rada inercijske mjerne jedinice.....	9
4.1.1.	Akcelerometar .....	9
4.1.2.	Žiroskop.....	10
4.1.3.	Magnetometar .....	10
4.2.	Senzor ICM-20948 .....	11
4.3.	Razvojna pločica 9DOF 2 Click.....	13
4.4.	Spajanje elektroničkih komponenti u sustav .....	14
5.	Programska potpora.....	16
5.1.	Program za ESP32 mikrokontroler.....	16
5.2.	Razvojni okvir ESP-IDF.....	17
5.3.	Implementacija SPI komunikacije na ESP32 mikrokontroleru .....	21
5.4.	AHRS algoritam i Fusion biblioteka .....	26
5.4.1.	AHRS algoritam .....	26
5.4.2.	Fusion biblioteka .....	27
5.5.	Wi-Fi komunikacija putem ESP32 mikrokontrolera .....	29
5.6.	Vizualizacija orijentacije korištenjem programskog okruženja Unity .....	31
6.	Zaključak .....	34
	Literatura .....	35
	Sažetak.....	37

Summary..... 38

# 1. Uvod

Sustavi za praćenje pokreta ljudskog tijela služe za određivanje položaja i orijentacije dijela tijela u prostoru. Općenito, kod raznih sustava koji imaju pokretne dijelove ili su cijeli pokretni u mnogim situacijama korisna je informacija o položaju i orijentaciji. To se može ilustrirati brojnim primjerima iz različitih područja, npr. od autoindustrije gdje sustavi GPS (engl. *Global Positioning System*) kontinuirano prate lokaciju automobila tijekom vožnje, preko industrije zabave koja uključuje filmsku produkciju, video igre i animacije, sve do robotike.

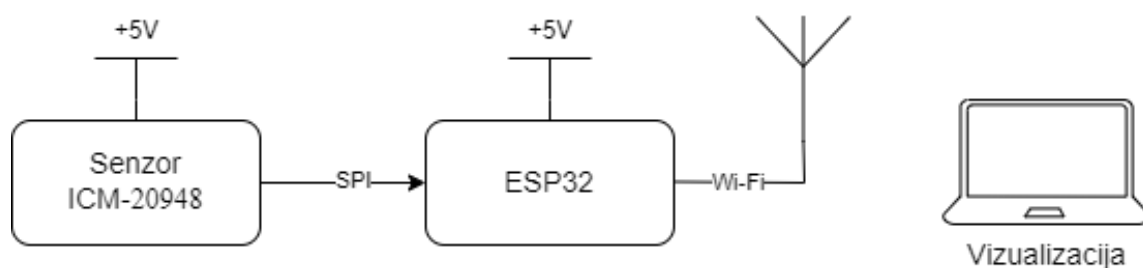
Jedna od praktičnih primjena, na kojoj je i primarni fokus ovog rada, je praćenje položaja i orijentacije ljudskog tijela u prostoru. Ta primjena sve je zastupljenija u profesionalnom sportu gdje se svakodnevno razvijaju novi uređaji za praćenje širokog spektra karakteristika sportaša. Osim sporta, koristi se i u medicini za praćenje položaja i kretanja pacijenata u svrhu prevencije i dijagnostike. Kako danas sve više raste svijest o potrebi i dobrobitima kretanja i redovite tjelesne aktivnosti, gotovo svaki moderni pametni sat u sebi sadrži senzore koji prate pokrete ljudskog tijela i na temelju dobivenih podataka daju korisne informacije o navikama i aktivnostima korisnika.

Jedan od tipičnih načina kako je moguće pratiti pokrete ljudskog tijela je korištenjem inercijskih senzora. U ovom radu fokus će biti na tzv. IMU (engl. *Inertial Measurement Unit*) grupi senzora i oni će se koristiti kao pristup rješenju problema određivanja položaja dijela ljudskog tijela. Cilj rada bio je implementirati vlastito rješenje bežičnog sustava koje može u stvarnom vremenu odrediti i vizualizirati orijentaciju predmeta ili dijela tijela na koji je postavljen senzorski čvor. Za implementaciju sustava odabran je mikrokontroler iz porodice ESP32 jer nudi ugrađenu mogućnost bežične komunikacije, a jednostavno ga je povezati s odabranim inercijskim senzorom i realizirati programsku potporu za obradu senzorskih podataka u stvarnom vremenu. Sustav se postavlja na ruku ili neki drugi dio tijela koji se želi pratiti, a korisnik je omogućeno promatranje orijentacije u stvarnom vremenu na svom osobnom računalu praćenjem orijentacije 3D modela, koji je sinkroniziran s postavljenim senzorom.

## 2. Opis sustava

Cilj ovog rada bio je razvoj sustava koji bežično prati i na osobnom računalu vizualizira orijentaciju tijela u prostoru. U konkretnoj primjeni koristit će se za mjerenje orijentacije ljudske ruke na koju je postavljen bežični senzorski čvor, ali će se moći jednostavno koristiti i za praćenje orijentacije bilo kojeg predmeta ili dijela tijela na koji je sustav postavljen.

Sustav za praćenje pokreta temelji se na razvojnoj pločici s ESP32 mikrokontrolerom koja obrađuje podatke i ostvaruje bežičnu komunikaciju. Za mjerenje je korišten *9DOF2 Click board* koji ima integriran senzor IMC-20948 koji sadrži akcelerometar, žiroskop i magnetometar. Ta 3 senzora mjere akceleraciju, kutnu brzinu i magnetsku indukciju. Iz vrijednosti tih triju fizikalnih veličina koje na različite i neovisne načine pružaju informaciju o orijentaciji senzora moguće je programski odrediti orijentaciju senzora u prostoru u odnosu na zemljinu površinu. Sustav je izveden u obliku prototipa na način da je ESP32 razvojna pločica jednostavno spojena s *9DOF2 Click* senzorom. Za napajanje sustava koristi se stabilizirani naponski izvor od 5 V. Za demonstraciju rada sustava potrebno je i osobno računalo koje vizualizira podatke dobivene od ESP32 mikrokontrolera.

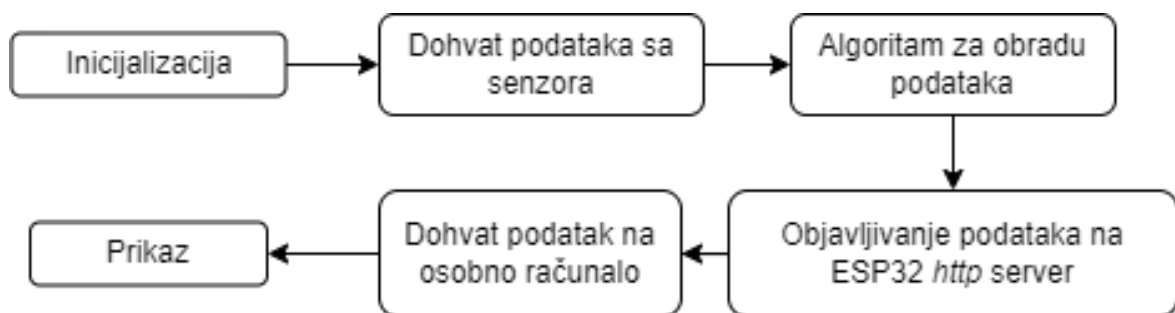


**Slika 2.1 Blok dijagram sustava**

Programska potpora za ESP32 mikrokontroler razvijena je u programskom jeziku C. Za potrebe komunikacije između senzora ICM-20948 i ESP32 mikrokontrolera bilo je potrebno implementirati komunikacijski protokol. U ovom radu korišten je SPI (engl. *Serial Peripheral Interface*) protokol koji je jednostavan za implementaciju, a odlikuje se velikim brzinama prijenosa. Ključan dio ispravnog rada sustava je način na koji se sirovi podaci iz senzora pretvaraju u Eulerove kuteve koji jednoznačno definiraju orijentaciju tijela u prostoru. Za te potrebe je u programskom jeziku C implementiran algoritam koji obavlja potrebne izračune. Izračunate vrijednosti potrebno je bežično poslati s ESP32 mikrokontrolera na osobno računalo. Bežičnu komunikaciju moguće je kod ESP32 mikrokontrolera izvesti putem Bluetooth ili Wi-Fi sučelja. U ovom radu odabrana je Wi-Fi

tehnologija u kojoj se ESP32 spaja na lokalnu Wi-Fi mrežu na koju je spojeno i osobno računalo. Nakon spajanja na lokalnu mrežu, ESP32 postaje *http* poslužitelj koji kontinuirano objavljuje podatke spremne za vizualizaciju. Na slici Slika 2.2 je blok dijagramom prikazan opisani tok izvođenja programa.

Programska potpora za vizualizaciju orijentacije izrađena je u programskom jeziku C# i *Unity* okolini. Taj dio koda služi isključivo za bežično primanje podataka s ESP32 *http* poslužitelja te njihovu vizualizaciju, a izvršava se u potpunosti na osobnom računalu. Podaci dolaze u paketima od tri varijable tipa *float*, koje predstavljaju valjanje, nagib i smjer (engl. *roll, pitch, yaw*).



**Slika 2.2 Blok dijagram izvođenja koda**

Konačan rezultat spajanja svih komponenti u sustav je simulirani model u okruženju *Unity* koji u stvarnom vremenu dinamički prati orijentaciju senzora *9DOF2 Click* u koji se nalazi na ruci u stvarnom svijetu.

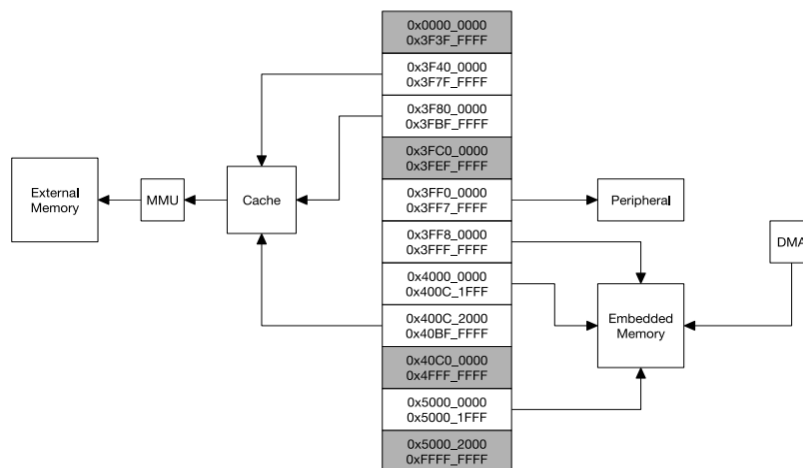
### 3. Razvojni sustav za mikrokontroler ESP32

#### 3.1. Osnovne karakteristike ESP32 mikrokontrolera i ESP32-WROOM-32 modula

U ovom radu korišten je mikrokontroler ESP32 jer predstavlja moderno i cjenovno pristupačno rješenje za razvoj IoT (engl. *Internet of Things*) uređaja te općenito aplikacija temeljenih na ugradbenim računalima koje zahtijevaju bežičnu komunikaciju putem Wi-Fi ili Bluetooth sučelja. Razvijen je od strane tvrtke Espressif Systems.

Za temelj rješenja u ovom radu odabran je ESP32-WROOM-32 modul, koji sadrži *system-on-chip* (SoC) ESP32-D0WDQ6, a koji je projektiran da bude skalabilan i prilagodljiv, s dvije procesorske jezgre koje se mogu kontrolirati pojedinačno. Frekvencija takta procesora može se prilagoditi u rasponu od 80 MHz do 240 MHz. Modul ima 32-bitni procesor i sadrži kvarcni oscilator od 40 MHz. Osnovne tehničke karakteristike navedenog modula su sljedeće:

- napon napajanja: 3,0 V – 3,6 V,
- minimalna struja napajanja – 500 mA,
- dimenzije – 18 mm × 25,5 mm × 3,10 mm.



Slika 3.1 Struktura memorije ESP32 mikrokontrolera. Preuzeto iz [2].

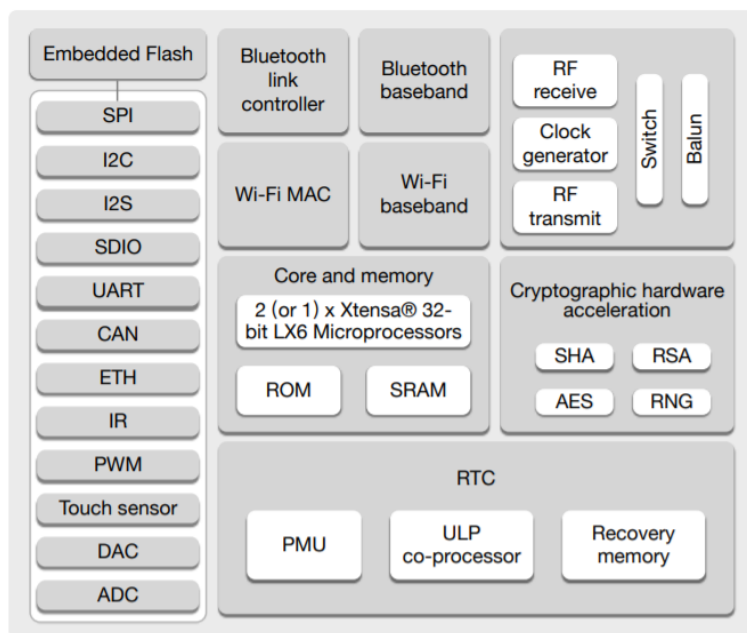
Modul sadrži integriranu memoriju tipa SPI flash od 4 MB, a kako koristi GPIO (engl. *General Purpose Input/Output*) pinove od 6 do 11, te pinove nije moguće koristiti kao



standardne GPIO ili SPI priključke. Također, odabrani modul implementira protokole za bežičnu komunikaciju:

- *Bluetooth*: podržava *Bluetooth* v4.2 BR/EDR i BLE specifikacije,
- *Wi-Fi*: podržava 802.11 b/g/n protokole, do 150 Mbps (2412 ~ 2484 MHz).

Za potrebe ovog projekta korišten je Wi-Fi protokol za povezivanje s osobnim računalom i SPI komunikacija za povezivanje s IMU senzorom.



**Slika 3.2 Blok shema ESP32 sustava. Preuzeto iz [2].**

Na slici Slika 3.2 prikazana je blok shema ESP32 modula. Osim navedenih, dodatne mogućnosti koje ovaj mikrokontroler nudi, a koje su korisne pri projektiranju bežičnih sustava s niskom potrošnjom, su razni režimi rada u načinu niske potrošnje:

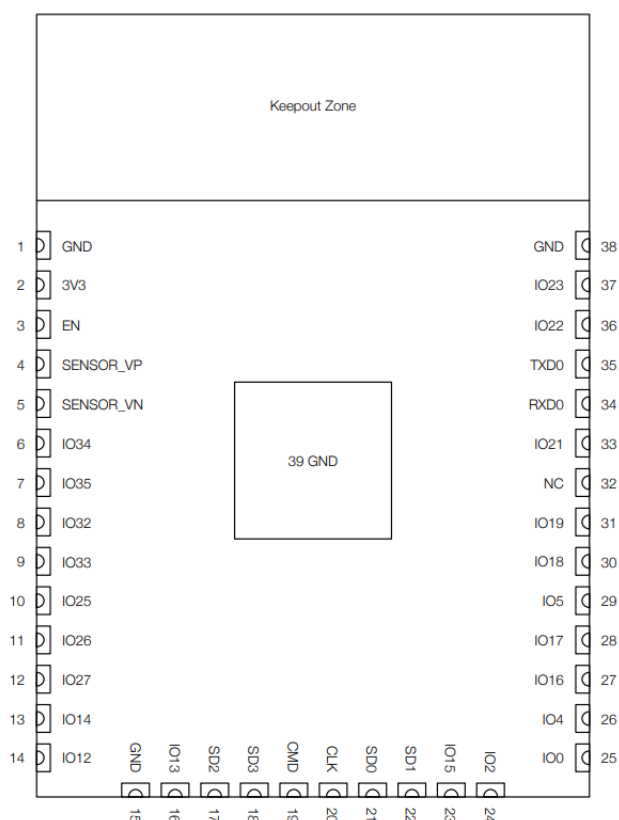
- *Active* – 180 mA ~ 240 mA za Wi-Fi, te oko 130 mA za Bluetooth LE,
- *Modem-sleep* - 20 mA ~ 31 mA,
- *Light-sleep* – 0,8 mA,
- *Deep-sleep* - 10  $\mu$ A ~ 150  $\mu$ A,
- *Hibernation* (RTC timer only) - 5  $\mu$ A.

Korištenjem ovih mogućnosti mogu se projektirati i sustavi vrlo niske potrošnje, što je ključno u uređajima napajanim baterijama ili akumulatorima.

ESP32-WROOM-32 modul od vanjskih komponenti uz sam mikrokontroler sadrži i još neke korisne sklopovske resurse kao što su:

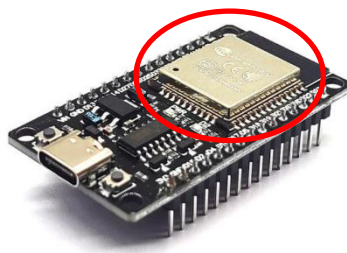
- kristalni oscilator 40 MHz,
- SPI *flash* memorija (GPIO6-GPIO11),
- antena i pripadajući filter<sup>1</sup>,
- integrirani sklop za JTAG, UART i *boot*.

Takav modul moguće je integrirati u razvojnu pločicu koja je spremna za daljnje korištenje.



**Slika 3.3 Izvodi ESP32-D0WDQ6. Preuzeto iz [3].**

<sup>1</sup> Vrijednosti induktiviteta L i kapaciteta C koji određuju filter ovise o izvedbi modula

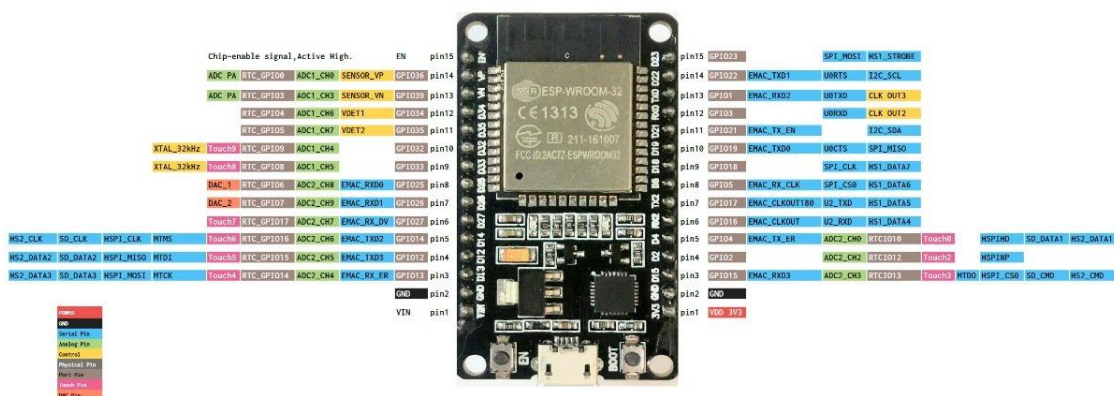


Slika 3.4 ESP32-WROOM-32 na razvojnoj pločici. Preuzeto iz [13].

## 3.2. Razvojna pločica ESP32-DevKit V1

Za implementaciju rješenja odabrana je razvojna pločica koja sadrži ESP32-WROOM-32 modul i koja je prikazana na slici Slika 3.5. Radi se o *ESP32-DevKit V1* pločici koja ima USB-C (engl. *Universal Serial Bus type C*) priključak za povezivanje s računalom.

Da bi se ESP32 mogao programirati, na razvojnoj pločici je između samog ESP32 modula i USB-C priključka ugrađen programator CH340C za koji je prethodno potrebno instalirati odgovarajući upravljački program.



Slika 3.5 Raspored priključaka na razvojnoj pločici. Preuzeto iz [4].

Na razvojnoj pločici postoji regulator napona koji služi za napajanje modula konstantnim naponom od 3,3 V te dvije tipke koje su označene s EN i BOOT. Ova razvojna pločica ima 30 izvoda od kojih je 25 moguće koristiti kao ulazno izlazne priključke, što je za ovaj rad bilo sasvim dovoljno. Priključci koji su zbog SPI komunikacije bitni imenovani su s prefiksom SPI\_\*. Po potrebi je između računala i vanjskog USB priključka na razvojnoj pločici moguće dodati i JTAG (engl. *Joint Test Action Group*) debugger koji služi pri detaljnoj analizi rada samog mikrokontrolera i otklanjanja problema u radu, ali u ovom radu

to nije bilo korišteno jer se programski kod prenosio s računala na pločicu preko UART (engl. *Universal Asynchronous Receiver / Transmitter*) sučelja.

## 4. Inercijski senzor

### 4.1. Načelo rada rada inercijske mjerne jedinice

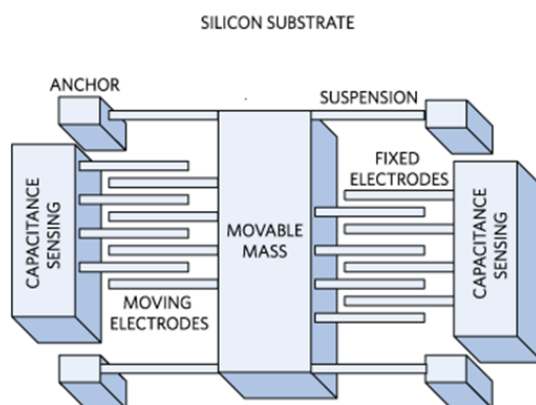
Za potrebe ovog rada odabran je IMU senzor ICM-20948 proizvođača TDK, a za njegovu lakšu primjenu korištena je razvojna pločica *9DOF 2 Click* tvrtke Mikroelektronika, koja na sebi ima već ugrađen spomenuti senzor. IMU jedinica tipično ima ugrađeno više inercijskih senzora, a u slučaju senzora ICM-20948 radi se o tri senzora (akcelerometar, žiroskop i magnetometar), koji će biti opisani u nastavku.

#### 4.1.1. Akcelerometar

Akcelerometar je senzor koji mjeri ukupnu akceleraciju (ubrzanje) u jednom smjeru. U većini primjena potrebno je odrediti orijentaciju u 3D prostoru tako da je potrebno mjeriti ubrzanje u 3 ortogonalne osi. Ukupna akceleracija sastoji se od dvije komponente: komponenta sile teže i inercijska komponenta zbog inercijskih sila koje djeluju na senzor. Na najnižoj razini, akcelerometar u korištenom senzoru izveden je s pomoću MEMS (engl. *Micro-Electromechanical System*) tehnologije.

Akcelerometar u senzoru ICM-20948 ima sljedeće karakteristike:

- digitalni izlazi za X, Y i Z osi, s programabilnim rasponom pune skale od  $\pm 2$  g,  $\pm 4$  g,  $\pm 8$  g i  $\pm 16$  g te integriranim 16-bitnim analogno digitalnim pretvornicima,
- mogućnost programskog izbora niskopropusnih filtara,
- prekid stanja spavanja na pokret uz nisku potrošnju senzora.



Slika 4.1 Princip rada MEMS akcelerometra. Preuzeto iz [12].

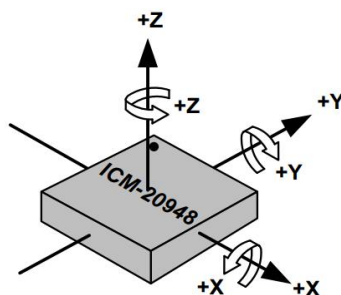
### 4.1.2. Žiroskop

Žiroskop je senzor koji mjeri komponente kutne brzine po pojedinim koordinatnim osima. Kutna brzina definirana je kao promjena kuta pri rotaciji tijela u jedinici vremena [1]. MEMS žiroskopi mjere kutnu brzinu primjenjujući Coriolisov efekt, koji se odnosi na silu inercije koja djeluje na objekte u pokretu u odnosu na rotirajući okvir [9].

Žiroskop senzora ICM-20948 ima sljedeće karakteristike:

- digitalni izlazi za kutne brzine za X, Y i Z osi, s programabilnim rasponom pune skale od  $\pm 250$  dps,  $\pm 500$  dps,  $\pm 1000$  dps i  $\pm 2000$  dps te integriranim 16-bitnim analogno digitalnim pretvornicima,
- mogućnost programskog odabira brzine uzorkovanja,
- mogućnost programskog odabira niskopropusnih filtara,
- samotestiranje.

U ovom radu korištena su mjerenja žiroskopa i akcelerometra za određivanje orijentacije u sve 3 osi za oba senzora.



Slika 4.2 Orijentacija osi osjetljivosti i polaritet rotacije. Preuzeto iz [10].

### 4.1.3. Magnetometar

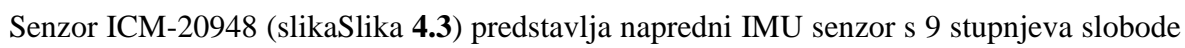
Magnetometar je senzor koji služi za mjerenje smjera i/ili iznosa magnetske indukcije prirodnog magnetskog polja okoline (Zemlje). Na Zemljinoj površini magnetska indukcija iznosi od 25 do 65 mT. Ako zanemarimo izvore magnetskih smetnji, tada se magnetometar može koristiti kao kompas za određivanje smjera u kojem je senzor okrenut u odnosu na Zemljin magnetski sjeverni pol [1]. U ovom radu nisu korištena mjerenja magnetometra, ali senzor ICM-20948 pruža mjerenja magnetskog polja u sve 3 osi i moguće je te podatke

uključiti u algoritam za računanje orijentacije u prostoru. Unutar ICM-20948 čipa nalazi se AK09916 magnetometar tvrtke *Asahi Kasei Microdevices Corporation*.

Magnetometar senzora ICM-20948 ima sljedeće karakteristike:

- 3-osni silicijski monolitni *Hall-efekt* magnetski senzor s magnetskim koncentратором,
- širok dinamički raspon mjerenja i visoka rezolucija uz nižu potrošnju struje,
- rezolucija izlaznih podataka od 16 bita,
- raspon mjerenja pune skale  $\pm 4900 \mu\text{T}$ ,
- funkcija samotestiranja s unutarnjim magnetskim izvorom za potvrdu rada senzora.

## 4.2. Senzor ICM-20948

Senzor ICM-20948 (slika ) predstavlja napredni IMU senzor s 9 stupnjeva slobode (9-DoF IMU senzor) i s niskom potrošnjom energije, a prilagođen je za primjene poput mobilnih telefona, pametnih satova i IoT uređaja. Ovaj senzor, koji proizvodi *InvenSense* i koji je dio TDK korporacije, ima kombinaciju troosnih žiroskopa, akcelerometara i magnetometara, pakiranih u 3 mm x 3 mm x 1 mm QFN (engl. *Quad Flat No-lead package*) kućište.

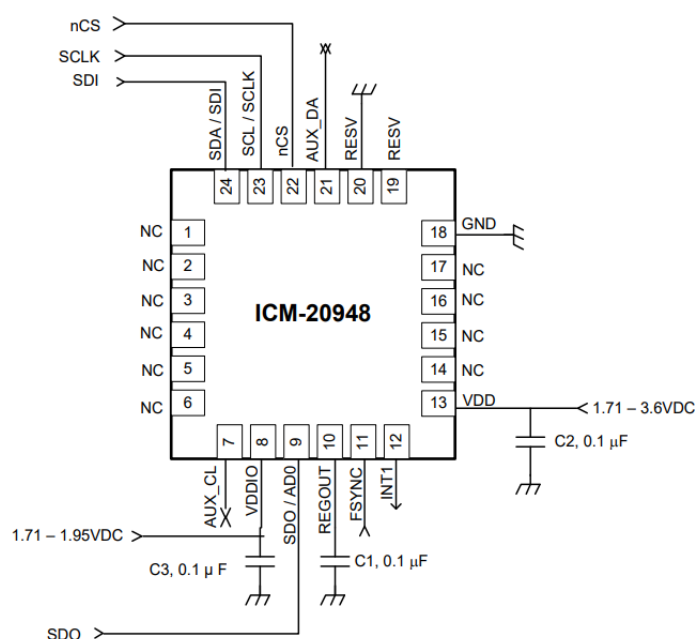
ICM-20948 integrira uz troosni žiroskop, troosni akcelerometar i troosni magnetometar i digitalnu obradu kretanja, što olakšava obradu podataka o kretanju bez značajnijeg opterećenja glavnog procesora uređaja. Ovo poboljšava performanse sustava u smislu potrošnje energije i omogućava složene algoritme za obradu kretanja koji su ključni za točnost i pouzdanost u stvarnom vremenu. Ova kombinacija čini ICM-20948 praktičnim za širok spektar aplikacija, pružajući ne samo standardne podatke o orijentaciji i kretanju, već i napredne mogućnosti poput navigacije i praćenja prostorne orijentacije [10].



**Slika 4.3 ICM-20948 MEMS Senzor za praćenje pokreta. Preuzeto iz [10].**

Jedna od najistaknutijih karakteristika ICM-20948 je njegova niska potrošnja energije, što ga čini idealnim za uređaje koji su ovisni o bateriji. S mogućnošću rada na naponima od 1,71 V do 3,6 V i integracijom visoko efikasnih 16-bitnih analogno digitalnih pretvornika. ICM-20948 također podržava i pomoćno I2C sučelje za povezivanje drugih vanjskih senzora, što omogućava proširenje funkcionalnosti bez potrebe za dodatnim mikrokontrolerom.

ICM-20948 pruža još i neke napredne funkcije, kao što su samotestiranje i kalibracija za vrijeme izvođenje programa, osiguravajući da uređaj ostane pouzdan i precizan tijekom rada. Samotestiranje omogućava uređaju da provjeri vlastite funkcionalnosti i performanse, dok kalibracija osigurava da senzori održavaju optimalnu preciznost unatoč mogućim vanjskim promjenama ili dugotrajnom korištenju.



**Slika 4.4 Izvodi senzora ICM-20948. Preuzeto iz [10].**



Sama mjerenja akceleracije izražavaju se kao udio akceleracije sile teže (npr. 0,5 g), dok se mjerenja kutne brzine iz žiroskopa iskazuju se u dps (engl. *degrees per second*), odnosno stupnjeva po sekundi. Inicijalna tolerancija mjerenja akceleracije iznosi  $\pm 25$  mg, a žiroskopa  $\pm 5$  dps. Senzor podržava brzu SPI komunikaciju koja može ići do 7 MHz, a otporan je na udarce do 20000 g. Radna temperatura mora biti u intervalu od  $-40$  °C do  $+85$  °C što zadovoljavajuće za sustave za praćenje ljudskog pokreta.

### 4.3. Razvojna pločica 9DOF 2 Click

*9DOF 2 Click* je kompaktna pločica tvrtke Mikroelektronika namijenjena aplikacijama koje zahtijevaju funkcionalnost praćenja pokreta i magnetometra s niskom potrošnjom energije. Pločica je odabrana za ovaj rad jer sadrži ugrađen opisani senzor ICM-20948. Pločica ima izvedene potrebne priključke za napajanje i za komunikaciju s mikrokontrolerom putem SPI i I2C protokola.

Iako je pločica izvorno napravljena kako bi se koristila kao dodatak većem razvojnom sustavu koji projektira ista tvrtka, ona se može koristiti i bez tog razvojnog sustava. U tom slučaju ne koriste se biblioteke proizvođača, već vlastito razvijeni programski kod.



**Slika 4.5 9DOF 2 Click, prednja i stražnja strana. Preuzeto iz [5].**

Pločica je mase 18 grama, a dimenzije su joj 28,6 mm x 25,4 mm x 12 mm, što ju čini prikladnom za prototipove, ali ne i krajnje proizvode zbog relativno velikih dimenzija.


*9DOF 2 Click* kompatibilna je s ESP32 mikrokontrolerom i zahtjeva napajanje od 3,3 V.

Pločica uz sam senzor sadrži još par dijelova koji su ključni za rad, a to su:

- regulator napona 1,8 V i zaštita od elektrostatičkog izboja (TXB0108),
- naponski pretvarač za usklađivanje naponskih razina,
- LED indikator rada,

- vanjski priključak za dodatni I2C senzor,
- izvode za jednostavno povezivanje s ostalim komponentama.

Proizvođač je definirao priključke kao što je prikazano na slici Slika 4.6, a postoje predviđeni priključci za napajanje, 4 priključka za SPI komunikaciju te 2 dodatna priključka (SYN i INT) koja nisu korištena u ovom radu. Crvenim pravokutnikom označeni su priključci relevantni za ovaj rad.

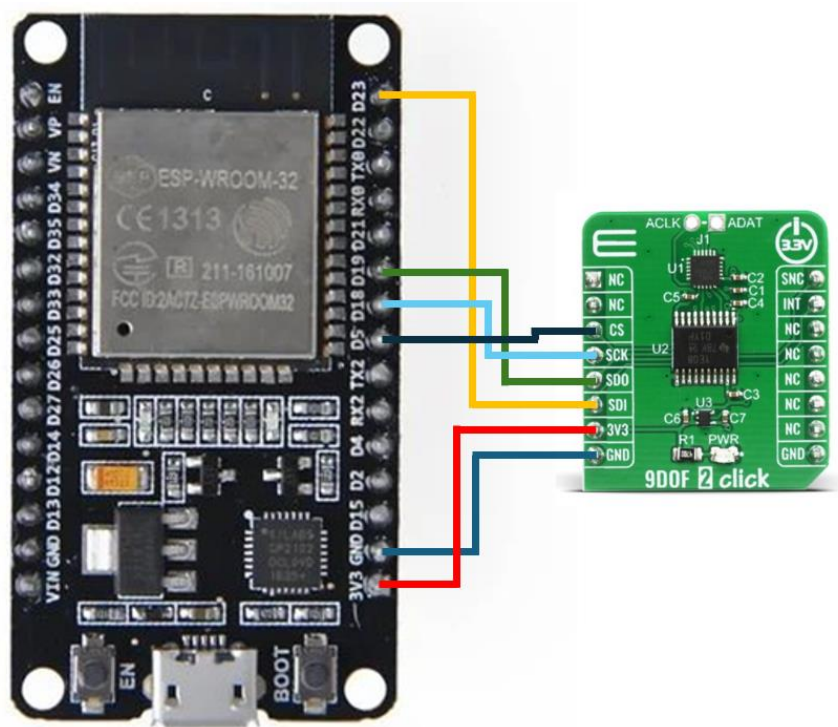
Notes	Pin					Pin	Notes
	NC	1	AN	PWM	16	<b>SYN</b>	External sync
	NC	2	RST	INT	15	<b>INT</b>	Interrupt
SPI Chip Select	<b>CS</b>	3	CS	RX	14	NC	
SPI Clock	<b>SCK</b>	4	SCK	TX	13	NC	
SPI Data OUT	<b>SDO</b>	5	MISO	SCL	12	NC	
SPI Data IN	<b>SDI</b>	6	MOSI	SDA	11	NC	
Power Supply	<b>3.3V</b>	7	3.3V	5V	10	NC	
Ground	<b>GND</b>	8	GND	GND	9	<b>GND</b>	Ground

Slika 4.6 Vanjski priključci 9DOF 2 Clicka. Preuzeto iz [5].

## 4.4. Spajanje elektroničkih komponenti u sustav

U ovom radu bilo je potrebno povezati senzor ICM-20948 i razvojnu pločicu mikrokontrolera ESP32 kako bi se ostvarila SPI komunikacija. Za to je potrebno spojiti 6 izvoda (MOSI, MISO, SCLK, CS, VCC, GND). Iz slike Slika 3.5 može se vidjeti da su izvodi

5, 18, 19 i 23 označeni kao SPI izvodi i njih je potrebno spojiti sa senzorom. Također, senzoru je potrebno spojiti 3V3 i GND priključke kako bi mu bio osiguran napon napajanja.



**Slika 4.7** Način povezivanja ESP32 razvojne pločice i senzora. Preuzeto iz [11] i [5]

## 5. Programska potpora

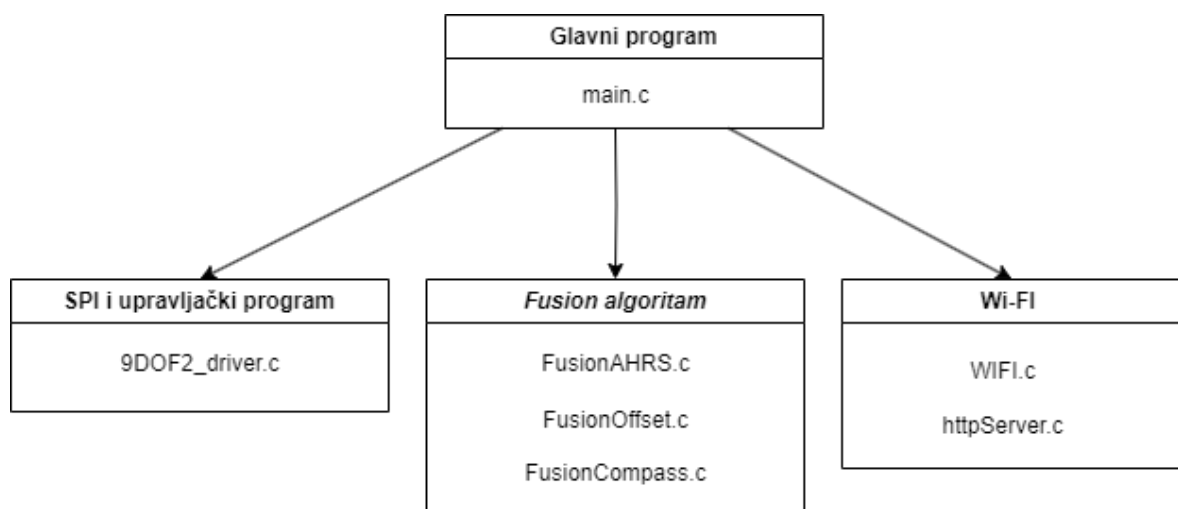
Programska potpora razvijena je za dvije platforme:

- program za ESP32 mikrokontroler (programski jezik C),
- program za osobno računalo za vizualizaciju mjerenja (programski jezik C# i razvojno okruženje Unity).

U ovom radu primarni fokus bio je na programskom rješenju koje se izvršava na mikrokontroleru, dok je C# program služio za programiranje okruženja za vizualizaciju rezultata.

### 5.1. Program za ESP32 mikrokontroler

Programski kod razvijen je u okruženju *Visual Studio Code* koristeći razvojni okvir ESP-IDF (engl. *Espressif IoT Development Framework*). Cjelokupni kod može se prikazati pojednostavljenim blok dijagramom na slici Slika 5.1.



Slika 5.1 Pojednostavljena struktura koda

Kod je razdvojen u 4 cjeline, a to su:

- glavni program koji poziva sve potrebne funkcije,
- upravljački program za senzor *9DOF 2 Click* koji implementira i SPI komunikaciju,

- *Fusion* algoritam koji iz senzorskih podataka računa orijentaciju tijela u prostoru,
- Wi-Fi dio koda koji uspostavlja HTTP poslužitelj i šalje podatke na osobno računalo.

Detaljna analiza programske podrške za mikrokontroler bit će izložena u nastavku.

## 5.2. Razvojni okvir ESP-IDF

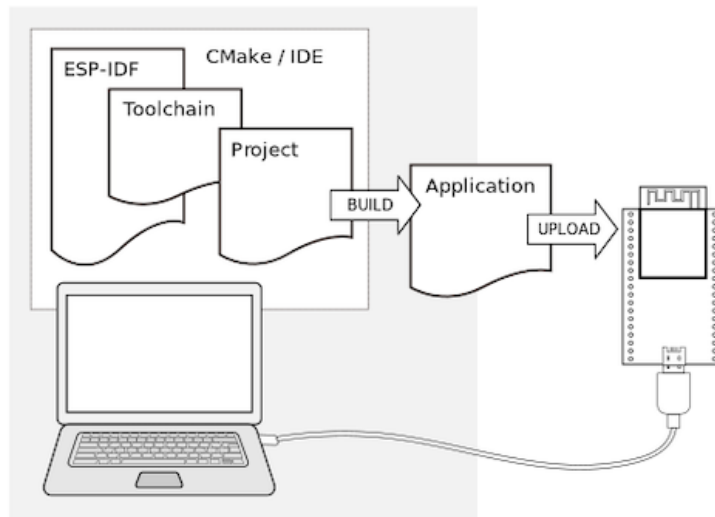
ESP-IDF je skup HAL (engl. *Hardware Abstraction Layer*) biblioteka tvrtke Espressif za razvoj programske potpore za ESP32 mikrokontrolere. U ovom radu korištena je najnovija verzija ESP-IDF ekstenzije 1.8.0 za VSCode i verzija 5.2.1 samog ESP-IDF okruženja.

ESP-IDF okruženje ubrzava i olakšava kreiranje i razvoj projekata za ESP32 mikrokontrolere. Prije početka izrade projekta bilo je potrebno instalirati ESP-IDF ekstenziju unutar razvojnog okruženja VSCode. Nakon instalacije, unutar `esp-idf` direktorija, mogu se pronaći sljedeći poddirektoriji i datoteke:

- `components` - direktorij koji u sebi sadrži izvorni kod za ESP-IDF API (engl. *Application Programming Interface*),
- `examples` - direktorij u kojem se nalaze gotovi primjeri tipičnih primjena za ESP32 mikrokontroler,
- `make` - direktorij koji sadrži datoteke potrebne za izgradnju *Cmake-a*,
- `tools` - direktorij koji sadrži *Python* skripte za upravljanje alatnim lancem,
- `docs` - direktorij u kojem su tekstualne datoteke i dokumentacija,
- nerazvrstane datoteke vezane za *git*, licence, instalacijske datoteke i skripte itd.

Za povezivanje računala na kojem je razvijen kod i mikrokontrolera korišten je USB-C kabel. Na računalu, korišten je serijski priključak COM8 (engl. *Communication Port 8*).

Koraci razvoja ESP-IDF alatom prikazani su na slici Slika 5.2.



**Slika 5.2** Dijagram toka razvojnog procesa za ESP-IDF. Preuzeto iz [14].

U prozoru kroz koji se postavljaju osnovne konfiguracije projekta potrebno je odabrati odgovarajuću razvojnu pločicu te serijski priključak računala preko kojeg je razvojna pločica spojena na računao.

Za kreiranje novog projekta moguće je koristiti postojeće predloške, a moguće je i stvoriti prazan projekt. U ovom radu korišten je predložak *sample\_project* koji kreira strukturu direktorija i datoteke na sljedeći način:

```

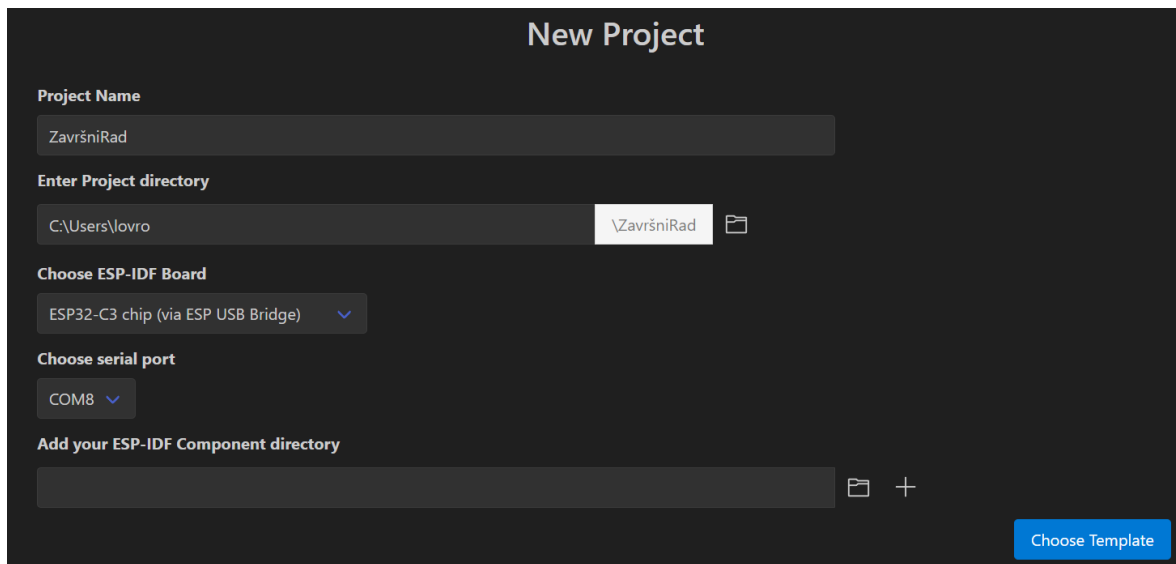
├─ CMakeLists.txt
├─ main
│   └─ CMakeLists.txt
│       └─ main.c
└─ README.md
  
```

Datoteku `CMakeLists.txt` koja se nalazi u korijenskom direktoriju projekta nakon što je jednom kreirana nije potrebno mijenjati.

```

cmake_minimum_required(VERSION 3.16)
include($ENV{IDF_PATH}/tools/cmake/project.cmake)
project(Završni_rad)
  
```

#### **Odsječak koda 1. *CMakeLists.txt* u korijenskom direktoriju**



**Slika 5.3 ESP-IDF izbornik za kreiranje novog projekta**

Datoteka `main.c` u ovom trenutku sadrži samo praznu definiciju funkcije `app_main`, a datoteka `CmakeLists.txt` koja se nalazi unutar `main` poddirektorija izgleda ovako:

```
idf_component_register(SRCS "main.c" INCLUDE_DIRS ".").
```

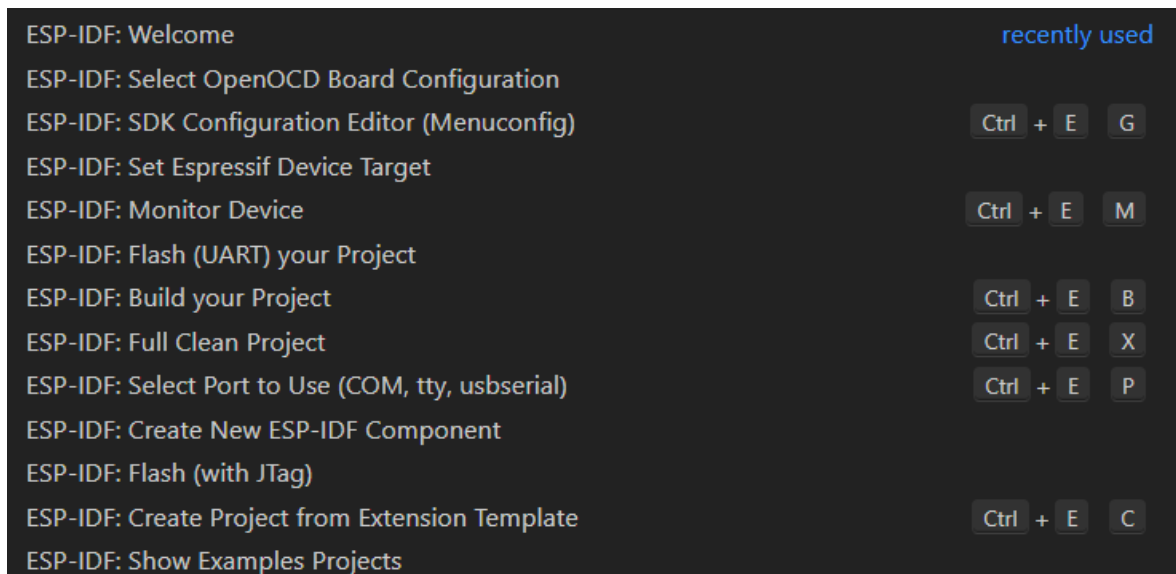
Ovu datoteku `CmakeLists.txt` potrebno je nadopunjavati kroz razvoj projekta.

Sada je postavljen kostur projekta, a prije početka razvoja programske podrške potrebno je upoznati se sa skriptama koje služe za rad s ESP-IDF alatom. Skriptama se moguće složiti na dva načina. Prvi je taj da korisnik u terminalu ručno upisuje naredbe tipa `idf.py [OPTIONS] COMMAND1 [ARGS]... [COMMAND2 [ARGS]...]...` Taj pristup je fleksibilniji jer omogućuje prilagodbu svih postavki koje je kao korisnik moguće birati.

Drugi pristup su već pripremljene naredbe koje se pozivaju pomoću aliasa. ESP-IDF već ima pripremljene naredbe koje se najčešće koriste pri razvoju projekata pa su posložene u oblik koji je praktičniji za krajnjeg korisnika i pokriva većinu slučajeva upotrebe.

Svi aliasi imaju prefiks „ESP-IDF:“.

Padajući izbornik sa slike Slika 5.4 otvaramo kombinacijom tipki (`ctrl+shift+p`).



**Slika 5.4 ESP-IDF aliasi korišteni u projektu**

Prvi bitniji alias je svakako *ESP-IDF: SDK Configuration Editor (Menuconfig)*. Njegovim izvršavanjem otvara se izbornik u kojem je moguće postaviti postavke konfiguracije cijelog projekta. Sve od GPIO priključaka preko SPI i UART protokola pa da Bluetootha i Wi-Fi ja potrebno je ovdje prethodno uključiti i namjestiti željene postavke. Kada su sve željene postavke uređene, promjene se spremaju pritiskom na tipku *Save*. Ono što se zapravo u pozadini događa je to da ESP-IDF uredi datoteku `sdkconfig` koja u sebi ima postavljene konfiguracije. Naravno, kao i uvijek, korisnik može direktno sam uređivati `sdkconfig` datoteku kako bi imao maksimalnu moguću fleksibilnost.

Prije *buildanja* projekta, potrebno je s pomoću aliasa *ESP-IDF: Select port to use (COM, tty, serial)* odabrati COM na koji je spojen USB-C kabel, a također je potrebno s pomoću aliasa *ESP-IDF: Set Espressif Devise Target* odabrati razvojnu pločicu koja se koristi.

Sljedeći često korišten alias je *ESP-IDF: Build your Project*. Ovaj alias po završetku svog izvršavanja kreira `build` direktorij u kojem je spremna `.bin` datoteka za programiranje mikrokontrolera. Po završetku *builda* u terminalu se pojavljuju informacije kao na slici Slika 5.5.

Sadržaj `build` direktorija moguće je izbrisati aliasom *ESP-IDF: Full Clean Project*.



```

Total sizes:
Used static DRAM: 32392 bytes ( 148344 remain, 17.9% used)
    .data size: 15664 bytes
    .bss size: 16728 bytes
Used static IRAM: 99346 bytes ( 31726 remain, 75.8% used)
    .text size: 98319 bytes
    .vectors size: 1027 bytes
Used Flash size : 747047 bytes
    .text: 589727 bytes
    .rodata: 157064 bytes
Total image size: 862057 bytes (.bin may be padded larger)

```

**Slika 5.5 Ispis nakon izvođenja ESP-IDF: Build your Project aliasa**

Nakon što se *build* alias uspješno izvrši do kraja odnosno nakon što korisnik ukloni sve potencijalne greške u kodu, vrijeme je za prebacivanje .bin datoteke na mikrokontroler. To se, uz pretpostavku da je ispravno odabran COM te razvojni kit, može učiniti pomoću *ESP-IDF: Flash (UART) your Project*.

Posljednji alias koji je bio često korišten u ovom projektu je *ESP-IDF: Monitor Device* koji otvara serijski monitor na kojem je moguće pratiti poruke koje ESP32 *loggira*.

### 5.3. Implementacija SPI komunikacije na ESP32 mikrokontroleru

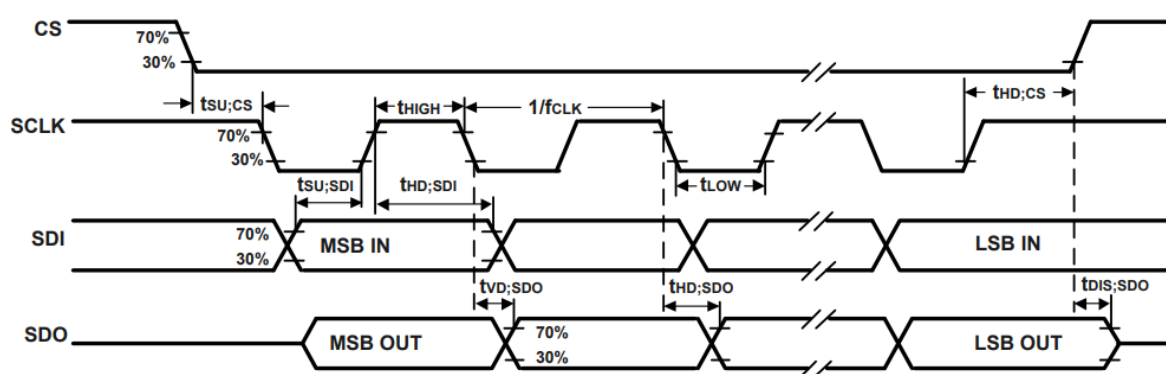
SPI (engl. *Serial Peripheral Interface*) je sinkroni serijski komunikacijski protokol koji se koristi za brzu komunikaciju s vanjskim sklopovima. Razvijen je početkom 1980-ih od strane tvrtke *Motorola* kako bi omogućio komunikaciju između mikrokontrolera i njegovih perifernih uređaja. SPI protokol omogućuje jednostavno povezivanje jednog glavnog uređaja (*master*) s jednim ili više podređenih uređaja (*slave*).

Zbog svoje jednostavnosti i efikasnosti, SPI je postao široko prihvaćen i koristi se u raznim aplikacijama, uključujući senzore, SD kartice i mrežna sučelja. Jedna od ključnih prednosti SPI protokola je njegova brzina; podržava brzine prijenosa podataka koje znatno premašuju one koje pruža *I<sup>2</sup>C* protokol, koji je također popularni serijski komunikacijski protokol za komunikaciju s vanjskim perifernim jedinicama [15].

Već u poglavlju 4.4 spomenuti su priključci koji se koriste u SPI protokolu, a to su:

- CS (engl. *Chip Select*) s pomoću kojeg mikrokontroler odabire s kojom periferijom želi ostvariti komunikaciju,
- SCLK (engl. *Serial Clock*) koji služi kako bi glavni uređaj (mikrokontroler) zadao takt kojim će se obavlja prijenos podataka,
- MOSI (engl. *Master Out Slave In*) za slanje podataka iz glavnog uređaja prema perifernom uređaju,
- MISO (engl. *Master In Slave Out*) za slanje podataka iz perifernog uređaja prema glavnom uređaju.

U ovom projektu glavni uređaj je mikrokontroler ESP32, a jedini periferni uređaj je senzor ICM-20948.



**Slika 5.6 Vremenski odnosi signala za SPI protokol na ICM-20948 senzoru. Preuzeto iz [5].**

Kao što se vidi na dijagramu Slika 5.6, periferni uređaj je odabran kada se na priključak CS dovede logička '0'. SCLK zadaje brzinu prijenosa, a podaci se prenose kroz MOSI i MISO linije. Osim toga definirana su i dozvoljena trajanja porasta signala što igra ulogu na visokim frekvencijama rada.

Kako bi se unutar programske podrške postavili preduvjeti za SPI komunikaciju funkcija `app_main()` poziva funkciju `spi_init()` koja je definirana unutar `9DOF2_driver.c` datoteke. Funkcija `spi_init()` postavlja vrijednosti dvije varijable tipa `spi_bus_config_t` te `spi_device_interface_config_t` koje u sebi sadrže sve parametre SPI komunikacije.

Kroz cijeli projekt na prikladnim mjestima umetnuti su kontrolni ispisi koji s pomoću funkcije `ESP_LOG()` na serijski monitor ispisuju potencijalne greške ili poruke potvrde u slučaju kada sve ispravno radi.

U ovom projektu, korištene su dvije razine `ESP_LOG()` funkcije:

- `ESP_LOGE()` za ispis grešaka,
- `ESP_LOGI()` za ispis informativnog tipa.

Sljedeće funkcije koja se tiču SPI komunikacije, a pozivaju se u `app_main()` funkciji su deklarirane kao:

- `uint8_t read_register(uint8_t address),`
- `bool write_register(uint8_t address, uint8_t value).`

Kao što i samo ime govori, funkcije služe za čitanje i pisanje 8-bitnih registara sa željene 8-bitne adrese. Ulazni parametar je adresa s koje je potrebno pročitati vrijednost ako se radi o čitanju iz registra. Ako se radi o pisanju u registar tada je potrebno zadati adresu te vrijednost koja se upisuje na zadanu adresu.

```
uint8_t read_register(uint8_t address)
{
    uint8_t send_buf[2] = {SPI_READ_MASK | address, 0x00};
    uint8_t recv_buf[2] = {0};

    spi_transaction_t t;
    memset(&t, 0, sizeof(t)); // Zero out the transaction
    t.length = 16;             // Total bits to transfer
    t.tx_buffer = send_buf;    // Transmit buffer
    t.rx_buffer = recv_buf;    // Receive buffer
    t.flags = 0;

    esp_err_t ret = spi_device_transmit(spi_handle, &t);
    if (ret != ESP_OK) {
        ESP_LOGE("SPI", "Failed to transmit: %s", esp_err_to_name(ret));
    } else {
        ESP_LOGI("SPI", "Received: 0x%02X", recv_buf[1]);
    }
    return recv_buf[1];
}
```

**Odsječak koda 2 Definicija funkcije `read_register()` unutar datoteke `9DOF2_driver.c`**

```

bool write_register(uint8_t address, uint8_t value)
{
    uint8_t tx_data[2];
    tx_data[0] = SPI_WRITE_MASK & address; // Clear MSB to indicate a
write operation
    tx_data[1] = value;

    spi_transaction_t transaction;
    memset(&transaction, 0, sizeof(spi_transaction_t)); // Zero out the
transaction struct
    transaction.length = 16;
    transaction.tx_buffer = tx_data;
    transaction.rx_buffer = NULL;

    esp_err_t result = spi_device_transmit(spi_handle, &transaction);
    if (result != ESP_OK)
    {
        ESP_LOGE("SPI", "Failed to write to sensor: %s",
esp_err_to_name(result));
        return false;
    }
    else
    {
        ESP_LOGI("SPI", "Data 0x%X written to register 0x%X
successfully", value, address);
        return true;
    }
}

```

### **Odsječak koda 3 Definicija funkcije write\_register() unutar datoteke 9DOF2\_driver.c**

Posljednja bitna funkcija unutar *9DOF2\_driver.c* datoteke je funkcija za očitavanje rezultata mjerenja akceleracije i žiroskopa. Funkcije su vrlo slične, a razlikuju se tek u tome s kojom konstantom na kraju množe ili dijele pročitane vrijednosti. Ime funkcije koja služi za dohvat rezultata mjerenja žiroskopa naziva se `read_gyro()`. Funkcije vraćaju mjerenja akceleracije/kutne brzine u samo jednoj osi, a os se bira ulaznom varijablom `uint8_t accel_out_adress`.

```

float read_accel(uint8_t accel_out_adress)
{
    uint8_t tx_data_high[2] = {SPI_READ_MASK | accel_out_adress, 0x00};
    uint8_t rx_data_high[2] = {0};
    spi_transaction_t trans_high = {
        .length = 16,
        .tx_buffer = tx_data_high,
        .rx_buffer = rx_data_high
    };
    spi_device_transmit(spi_handle, &trans_high);

    uint8_t tx_data_low[2] = {SPI_READ_MASK | (accel_out_adress + 1),
0x00};
    uint8_t rx_data_low[2] = {0};
    spi_transaction_t trans_low = {
        .length = 16,
        .tx_buffer = tx_data_low,
        .rx_buffer = rx_data_low
    };
    spi_device_transmit(spi_handle, &trans_low);

    int16_t accel = ((int16_t)rx_data_high[1] << 8) | rx_data_low[1];
    return accel / ACCEL_SENSITIVITY;
}

```

#### Odsječak koda 4 Implementacija funkcije read\_accel()

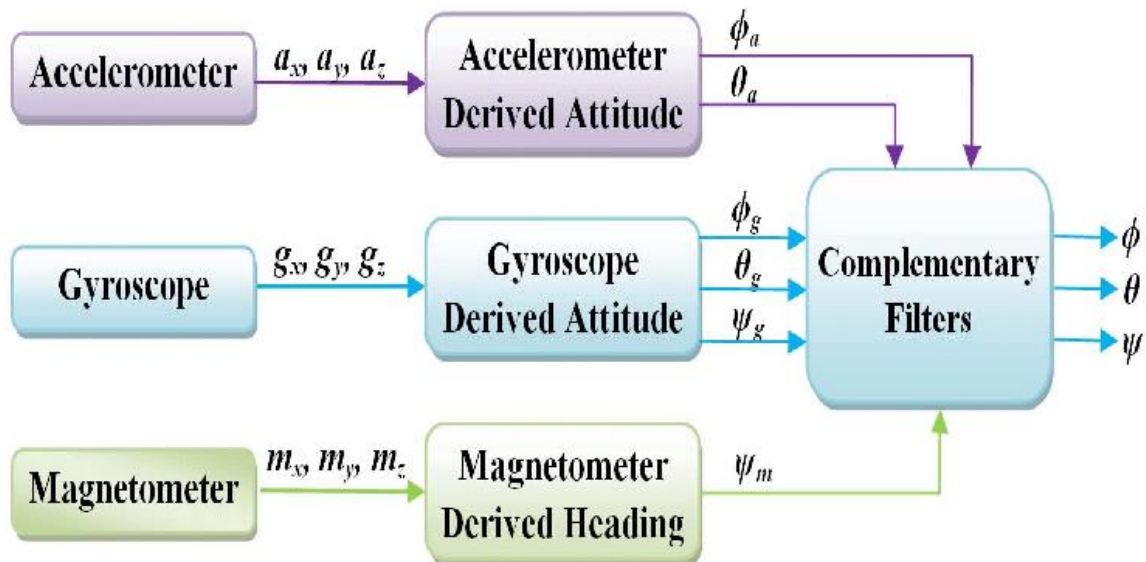
Način na koji radi Odsječak koda 4 najlakše je pokazati na jednom primjeru. U slučaju da se želi pročitati vrijednost akceleracije u smjeru x-osi, potrebno je kao ulazni parametar u funkciju `read_accel()` poslati adresu višeg bajta (*ACCEL\_XOUT\_H* definiran u *9DOF2\_driver.h*) zapisa rezultata. Funkcija čita viši i niži bajt rezultata, spaja ih u jednu 16-bitnu varijablu, dijeli s faktorom *ACCEL\_SENSITIVITY* i glavnom programu vraća dobivenu *float* vrijednost.

Svaki put kada je potrebno ažurirati orijentaciju tijela u prostoru potrebno je pozvati funkcije `read_accel()` o `read_gyro()` u sve 3 osi kako bi ostale funkcije radile s ispravnim vrijednostima.

## 5.4. AHRS algoritam i Fusion biblioteka

### 5.4.1. AHRS algoritam

AHRS (engl. *Attitude and Heading Reference System*) je algoritam koji kombinira podatke iz žiroskopa, akcelerometra i magnetometra kako bi dobio orijentaciju senzora u prostoru. Algoritam također može podržavati sustave koji koriste samo žiroskop i akcelerometar, kao i sustave koji kombiniraju žiroskop i akcelerometar s vanjskim izvorom mjerenja smjera, kao što je GPS. Među AHRS algoritmima postoji više implementacija, kao što su Madgwickov algoritam, Mahonyjev algoritam, *Kalman filter*, *Extended Kalman filter* itd. Algoritam korišten u ovom radu bazira se na Madgwickovom algoritmu [17].



Slika 5.7 Blok dijagram AHRS algoritma. Preuzeto iz [19].

Kao što se vidi iz blok dijagrama na slici Slika 5.7, ulazni parametri algoritma su akceleracija, kutna brzina i magnetska indukcija, a to je upravo ono što senzor IMC-20948 [10] mjeri. Algoritam je moguće implementirati samo uz mjerenja jedne od te 3 veličine, ali svako dodatno mjerenje druge fizičke veličine može povećati točnost rezultata. Svi ulazni parametri ulaze u filter koji na svom izlazu daje Eulerove kuteve ili kvaternione koji jednoznačno definiraju orijentaciju tijela u prostoru.

### 5.4.2. Fusion biblioteka

*Fusion* je biblioteka za obradu podataka s IMU senzora koja je optimirana za ugradbene računalne sustave [16]. *Fusion* je implementirana kao C biblioteka prenosivog koda, ali je dostupna i kao *Python* paket pod nazivom *imufusion*. Implementacija algoritma unutar biblioteke temelji se na AHRS algoritmu izloženom u doktorskoj disertaciji Seana Madgwicka [17].

*Fusion* izračunava orijentaciju integrirajući podatke iz žiroskopa s dodatkom povratne veze. Povratna informacija jednaka je greški u trenutnom mjerenju orijentacije, kako je određeno drugim sensorima, i pomnožena je s pojačanjem. Algoritam stoga funkcionira kao komplementarni filter koji kombinira mjerenja žiroskopa filtrirana visokopropusnim filtrom s mjerenjima drugih senzora filtriranim niskopropusnim filtrom. Nisko pojačanje više "vjeruje" žiroskopu dok visoko pojačanje povećava utjecaj drugih senzora. Pojačanje jednako nula zanemaruje ostale senzore tako da se mjerenje orijentacije određuje samo na temelju žiroskopa [16].

*Fusion* kod normalizacije vektora koristi funkciju za brzo računanje drugog korijena [18] kako bi ubrzao izvođenje koda na mikrokontroleru uz minimalnu pogrešku. To je česta praksa u sustavima koja nemaju veliku procesorsku moć, a potreban je izračun drugog korijena broja.

U funkciji `app_main()` potrebno je inicijalizirati algoritam i to se radi na sljedeći način:

```
FusionAhrs ahrs;  
  
FusionAhrsInitialise(&ahrs);
```

Sve što radi `FusionAhrsInitialise` funkcija je to da resetira sve vrijednosti algoritma (to je potrebno ako se algoritam prethodno izvodio za vrijeme trenutnog izvršavanja programa) i postavlja postavke zadane u `FusionAhrsSettings` strukturi.

U slučaju da korisnik želi promijeniti parametre algoritma, to je moguće učiniti u strukturi koja opisuje postavke algoritma, a njena implementacija i postavke koje su otvorene za prilagodbu moguće je vidjeti u odsječku koda Odsječak **koda 5**.

```
const FusionAhrsSettings settings = {  
    .convention = FusionConventionNwu,  
    .gain = 0.5f,  
    .gyroscopeRange = 0.0f,  
    .accelerationRejection = 90.0f,
```

```

        .magneticRejection = 90.0f,
        .recoveryTriggerPeriod = 0,
};

```

### Odsječak koda 5 Struktura koja zadaje parametre AHRS algoritma

U ovom radu korištene su zadane vrijednosti.

Nakon inicijalizacije algoritma, u dva vektora spremaju se trenutne vrijednosti akceleracije i kutne brzine po osima:

```

const FusionVector gyro = {.axis = {gx, gy, gz}},
const FusionVector accel = {.axis = {ax, ay, az}}.

```

Varijable  $ax$ ,  $ay$  i  $az$  predstavljaju zadnje dohvaćene vrijednosti akceleracije po osima dok varijable  $gx$ ,  $gy$  i  $gz$  predstavljaju zadnje dohvaćene vrijednosti kutne brzine po osima. Sada još preostaje pozvati funkciju koja će izvršiti algoritam s aktualnim vrijednostima:

```

FusionAhRsUpdateNoMagnetometer(&ahrs, gyro, accel, SAMPLE_PERIOD);

```

Algoritam pri izvođenju u obzir uzima trenutno stanje sustava te odstupanje novoizmjerenih vrijednosti od tog stanja. Nakon što izračuna razliku između te dvije veličine, spreman je izračunati novi rezultat. Na ovaj način u sustav je umetnuta povratna veza koja ne postoji u inicijalnom Madgwickovom algoritmu. Kao rezultat poziva funkcije `FusionAhRsUpdateNoMagnetometer`, komponente kvaterniona ( $q_1$ ,  $q_2$ ,  $q_3$ ,  $q_4$ ) poprimaju nove vrijednosti. Na kraju je moguće kvaternione koji su dobiveni izvršavanjem algoritma pretvoriti u Eulerove kuteve. U ovom radu korišteni su Eulerovi kutevi kako bi se za vrijeme razvoja intuitivnije mogli validirati dobiveni rezultati. Formule za pretvorbu iz kvaterniona u Eulerove kuteve su:

$$\psi = \text{Atan2} (2q_2q_3 - 2q_1q_4, 2q_1^2 + 2q_2^2 - 1)$$

$$\theta = -\sin^{-1} (2q_2q_4 + 2q_1q_3)$$

$$\phi = \text{Atan2} (2q_3q_4 - 2q_1q_2, 2q_1^2 + 2q_4^2 - 1)$$

```

const FusionEuler euler =
FusionQuaternionToEuler(FusionAhRsGetQuaternion(&ahrs));

```

Vrijednostima pojedinih kuteva može se pristupiti preko `euler.angle.roll`.



## 5.5. Wi-Fi komunikacija putem ESP32 mikrokontrolera

ESP32 podržava 802.11 b/g/n Wi-Fi protokole, a zauzima frekvencijski spektar [2412 - 2484] MHz. Podržani načini rada su pristupna točka, stanica ili pristupna točka i stanica u isto vrijeme.

Kako bi se inicijalizirala Wi-Fi komunikacija na ESP32 mikrokontroleru, unutar funkcije `app_main()` poziva se funkcija `wifi_init()`.

```
void wifi_init(void) {
    esp_err_t ret = nvs_flash_init();
    if (ret == ESP_ERR_NVS_NO_FREE_PAGES || ret ==
ESP_ERR_NVS_NEW_VERSION_FOUND) {
        ESP_ERROR_CHECK(nvs_flash_erase());
        ret = nvs_flash_init();
    }
    ESP_ERROR_CHECK(ret);

    wifi_event_group = xEventGroupCreate();

    ESP_ERROR_CHECK(esp_netif_init());
    ESP_ERROR_CHECK(esp_event_loop_create_default());
    esp_netif_create_default_wifi_sta();

    wifi_init_config_t cfg = WIFI_INIT_CONFIG_DEFAULT();
    ESP_ERROR_CHECK(esp_wifi_init(&cfg));
    ESP_ERROR_CHECK(esp_wifi_set_storage(WIFI_STORAGE_RAM));
    ESP_ERROR_CHECK(esp_wifi_set_mode(WIFI_MODE_STA));

    wifi_config_t wifi_config = {
        .sta = {
            .ssid = WIFI_SSID,
            .password = WIFI_PASS
        },
    };

    ESP_ERROR_CHECK(esp_wifi_set_config(ESP_IF_WIFI_STA, &wifi_config));
    ESP_ERROR_CHECK(esp_event_handler_register(WIFI_EVENT,
ESP_EVENT_ANY_ID, &event_handler, NULL)
    ESP_ERROR_CHECK(esp_event_handler_register(IP_EVENT,
IP_EVENT_STA_GOT_IP, &event_handler, NULL));
    ESP_ERROR_CHECK(esp_wifi_start());
```

```
ESP_LOGI("wifi_manager", "wifi_init_sta finished.");
}
```

### Odsječak koda 6 Implementacija funkcije wifi\_init()

U odsječku koda Odsječak **koda 6** jasno se vidi na koji način funkcija inicijalizira Wi-Fi komunikaciju. Na početku se inicijalizira NVS (engl. *Non-Volatile Storage*) *flash*. To je dio memorije koji se neće brisati nakon isključenja napajanja. Nakon toga odrađuju se i sve ostale potrebne konfiguracije. Valja skrenuti pažnju na strukturu `wifi_config` u kojoj su pohranjeni SSID (engl. *Service Set Identifier*) te lozinka za pristup mreži. To su vrijednosti koje korisnik ručno mora upisati u program kako bi se ESP32 mogao spojiti na lokalnu mrežu.

Još jedan parametar koji korisnik ručno može zadati je koliko će se puta ESP32 pokušati spojiti na mreži prije nego li odustane. U ovom projektu ta vrijednost definirana je kao:

```
#define MAX_RETRY 10
```

Nakon što je Wi-Fi uspješno postavljen, sljedeći poziv iz funkcije `app_main()` je `http_server_start()`.

To je funkcija koja pokreće http poslužitelj, a pozornost valja skrenuti na dvije stvari.

Prva je format u kojem se podaci šalju na server. Ovo je bitno zbog toga što program za vizualizaciju mora znat kako će ih *parsirati*. Podaci se šalju u csv (engl. *comma-separated values*) obliku.

```
asprintf(&resp_str, "%.2f, %.2f, %.2f", global_yaw, global_pitch,
global_roll);
```

Također, valja skrenuti pozornost `httpd_uri_t` strukturu koja specificira kako ESP32 HTTP server treba reagirati na određene zahtjeve koje primi.

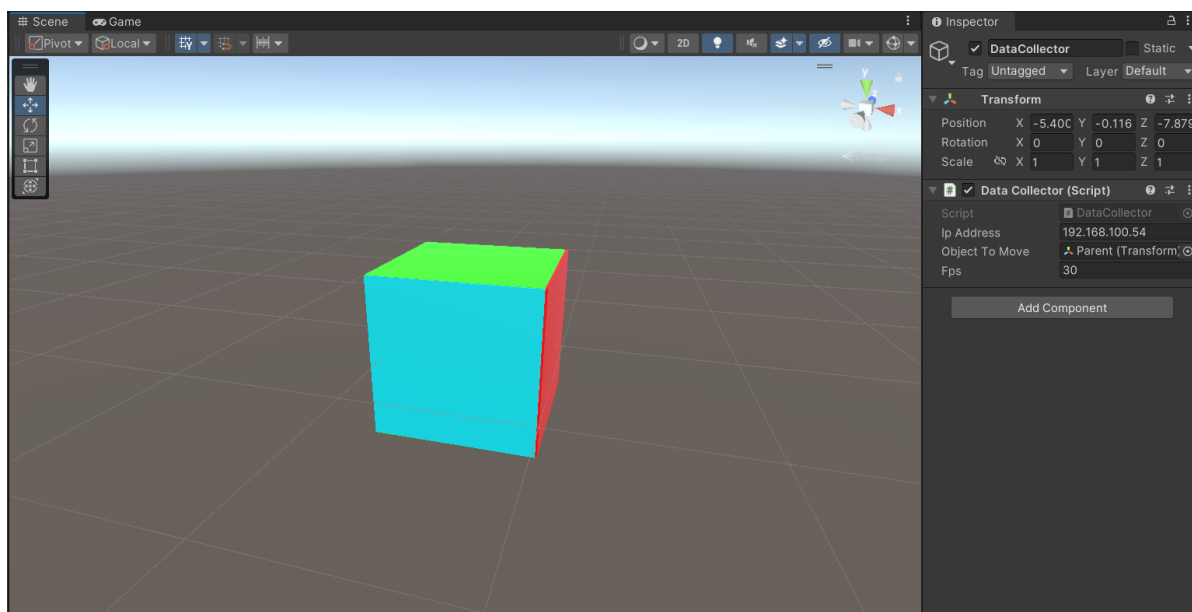
```
httpd_uri_t angles = {
    .uri      = "/angles",
    .method   = HTTP_GET,
    .handler   = angles_get_handler,
    .user_ctx = NULL
};
```

### Odsječak koda 7 Struktura koja određuje odziv na HTTP zahtjev

## 5.6. Vizualizacija orijentacije korištenjem programskog okruženja Unity

U ovoj fazi projekta algoritam ispravno radi i podaci se bežično šalju *http* protokolom preko Wi-Fi mreže na centralni server. Ti podaci su sada spremni za vizualizaciju na osobnom računalu u okruženju Unity.

Radi jednostavnosti prikaza korišten je model kocke s obojenim stranicama. Umjesto kocke moguće je koristiti bilo koji drugi 3D model i rezultat će biti isti i sve će se ispravno vizualizirati.



Slika 5.8 Vizualizacija unutar Unity okruženja

U Unity okruženju bilo je potrebno definirati IP adresu *http* poslužitelja ESP32 mikrokontrolera na kojem se kontinuirano objavljuju podaci. Potrebno je definirati i brzinu slanja zahtjeva za novim vrijednostima kuteva koja je za ovaj sustav odabrana da bude 30 fps (engl. *frames per second*).

Također, bilo je potrebno napisati kratki kod u C# koji kontinuirano dohvaća trenutne vrijednosti kuteva s poslužitelja.

```
public class DataCollector : MonoBehaviour
{
    [SerializeField] private string ipAddress;
    [SerializeField] private Transform objectToMove;
    [SerializeField] private float fps = 30;
```

```

private void Start()
{
    InvokeRepeating("StartCoroutineGetAngles", 0f, 1 / fps);
}
private void StartCoroutineGetAngles()
{
    StartCoroutine(GetAngles());
}
private IEnumerator GetAngles()
{
    string url = $"http://{ipAddress}/angles";
    using (UnityWebRequest request = UnityWebRequest.Get(url))
    {
        yield return request.SendWebRequest();

        if (request.result != UnityWebRequest.Result.Success)
        {
            Debug.LogError("Error: " + request.error);
        }
        else
        {
            UseData(request.downloadHandler.text);
        }
    }
}
private void UseData(string data)
{
    string[] angles = data.Split(',');
    float roll = float.Parse(angles[0]);
    float pitch = float.Parse(angles[1]);
    float yaw = float.Parse(angles[2]);

    objectToMove.localEulerAngles = new Vector3(-yaw, -roll, -pitch);
}
}

```

### **Odsječak koda 8 C# kod za dohvat podataka s ESP32 http poslužitelja**

Ovaj C# kod koristi Unity za vizualiziranje podataka o kutevima prikupljenih s ESP32 *http* poslužitelja. Pošto je C# objektni jezik, cijeli kod je jedna klasa imena `DataCollector`. Na početku klase definiraju se korištene varijable. Također se implementira i `Start()`

metoda koja se poziva pri pokretanju simulacije. U `Start()` metodi definira se fps vizualizacije, a moguće je zadati i vrijeme čekanja od pokretanja simulacije do početka dohvaćanja vrijednosti s poslužitelja. Implementirana je i metoda koja kontinuirano dohvaća podatke s *http* poslužitelja koji je uspostavio ESP32. Za kraj, potrebno je iz *stringa* koji je dohvaćen s poslužitelja izvući vrijednosti kuteva koje su međusobno odvojene zarezom (csv) te ažurirati prikaz s najnovijim vrijednostima.

Vektor u kojem su spremljene vrijednosti Eulerovih kuteva `Vector3(-yaw, -roll, -pitch)` ispred svakog elementa ima negativan predznak jer tako nulti položaj senzora u stvarnom svijetu odgovara nultom položaju vizualiziranog objekta.

## 6. Zaključak

Rad je demonstrirao razvoj i implementaciju bežičnog sustava za praćenje ljudskog pokreta koristeći IMU senzor. Kroz rad je uspješno ostvaren sustav koji mjeri i vizualizira orijentaciju predmeta u stvarnom vremenu.

ESP32 mikrokontroler pokazao se kao odličan izbor za ovu primjenu, omogućavajući obradu podataka u stvarnom vremenu i jednostavnu bežičnu komunikaciju. Komunikacijski protokol SPI bio je jednostavan za implementaciju i zadovoljio je sve potrebe komunikacije između senzora i mikrokontrolera. Štoviše, SPI može raditi na još većim brzinama tako da njegov potencijal u ovoj primjeni nije iskorišten do kraja.

Korišteni AHRS *Fusion* algoritam za računanje orijentacije ispunio je svoju svrhu. Zbog jednostavnosti, algoritam je radio samo s vrijednostima kutne brzine i akceleracije, dok su mjerenja magnetometra zanemarena. Uz ovo pojednostavljenje *Fusion* algoritam mogao je s lakoćom kontinuirano slati rezultate dovoljne točnosti za potrebe ovog rada. U budućim inačicama projekta točnost je moguće povećati uzimanjem magnetske indukcije u obzir pri izračunu Eulerovih kuteva.

Wi-Fi tehnologija omogućila je bežično povezivanje mikrokontrolera i osobnog računala, ali se pokazala i kao dio projekta koji uvodi najveća ograničenja. Prije svega, *http* protokol koji je korišten uvodi kašnjenje kod vizualizacije što je u nekim trenucima vidljivo i golim okom. Ovo je moguće riješiti korištenjem nekog drugačijeg protokola s manjem latencijom, npr. UDP. Nadalje, Wi-Fi sa sobom povlači visoku potrošnju što je velika mana u sustavu koji je baterijski napajan. Jedno od rješenja ovog problema je prelazak na tehnologiju bežične komunikacije koja ima manju potrošnju kao što je BLE (engl. *Bluetooth Low Energy*).

Vizualizacijom u Unityju pokazano je kako se kompleksni podaci mogu prezentirati korisniku na razumljiv i intuitivan način, te tako dodatno validirati rezultat.

# Literatura

- [1] Tehnologija u medicini, FER intranet, Stranica predmeta, Materijali. Poveznica: [https://www.fer.unizg.hr/download/repository/TUM\\_2022\\_LAB1\\_priprema.pdf](https://www.fer.unizg.hr/download/repository/TUM_2022_LAB1_priprema.pdf); svibanj 2024.
- [2] Espressif, ESP32 *Datasheet*. Poveznica: [https://www.espressif.com/sites/default/files/documentation/esp32\\_datasheet\\_en.pdf](https://www.espressif.com/sites/default/files/documentation/esp32_datasheet_en.pdf); lipanj 2024.
- [3] Espressif, ESP32-WROOM32 *Datasheet*. Poveznica: [https://www.espressif.com/sites/default/files/documentation/esp32-wroom-32\\_datasheet\\_en.pdf](https://www.espressif.com/sites/default/files/documentation/esp32-wroom-32_datasheet_en.pdf); lipanj 2024.
- [4] Sudo.is, ESP32 WROOM *Pinout*. Poveznica: <https://www.sudo.is/docs/esphome/boards/esp32wroom/>; lipanj 2024.
- [5] Mikroelektronika, *Click Board 9DOF 2*. Poveznica: <https://www.mikroe.com/9dof-2-click>; travanj 2024.
- [6] Mikroelektronika, *9DOF 2 Click Schematic*. Poveznica: <https://download.mikroe.com/documents/add-on-boards/click/9dof-2-click/9DOF-2-click-schematic-v101.pdf>; travanj 2024.
- [7] TDK InvenSense, ICM-20948 *Datasheet*. Poveznica: <https://download.mikroe.com/documents/datasheets/ICM-20948-datasheet.pdf>; travanj 2024.
- [8] K. Petanjek, „Wi-Fi senzorski čvor temeljen na ESP32 mikrokontroleru“, završni rad, Fakultet elektrotehnike i računarstva, Sveučilište u Zagrebu, 2020.
- [9] Vectornav, Akcelerometar, Žiroskop, Magnetometar. Poveznica: <https://www.vectornav.com/resources/inertial-navigation-primer/theory-of-operation/theory-mems>; lipanj 2024.
- [10] TDK InvenSense, ICM-20948. Poveznica: <https://invensense.tdk.com/products/motion-tracking/9-axis/icm-20948/>; srpanj 2024.
- [11] Indiamart, ESP32 30 pinova. Poveznica: <https://www.indiamart.com/proddetail/esp-wroom-32-esp32-esp-32s-development-board-2850945170097.html>; srpanj 2024.
- [12] Analog Devices, prikaz MEMS akcelerometra. Poveznica: <https://www.analog.com/en/resources/technical-articles/accelerometer-and-gyroscopes-sensors-operation-sensing-and-applications.html>; srpanj 2024.
- [13] AliExpress, ESP32 razvojna pločica, USB-C, 30 izvoda. Poveznica: <https://www.aliexpress.com/i/32964479255.html>; srpanj 2024.
- [14] Espressif, ESP-IDF. Poveznica: <https://docs.espressif.com/projects/esp-idf/en/stable/esp32/get-started/index.html>; srpanj 2024.
- [15] Wikipedia, SPI protokol. Poveznica: [https://en.wikipedia.org/wiki/Serial\\_Peripheral\\_Interface](https://en.wikipedia.org/wiki/Serial_Peripheral_Interface); srpanj 2024.
- [16] GitHub, x-io Technologies, Fusion AHRS. Poveznica: <https://github.com/xioTechnologies/Fusion/tree/main>; lipanj 2024.

- [17] x-io Technologies, Madwickov doktorski rad. Poveznica: <https://x-io.co.uk/open-source-imu-and-ahrs-algorithms/>; svibanj 2024.
- [18] Wikipedia, brzi drugi korijen. Poveznica: [https://en.wikipedia.org/wiki/Fast\\_inverse\\_square\\_root](https://en.wikipedia.org/wiki/Fast_inverse_square_root); srpanj 2024.
- [19] ResearchGate, Blok dijagram AHRS algoritma. Poveznica: [https://www.researchgate.net/figure/Block-diagram-of-the-fusion-algorithm-for-AHRS\\_fig1\\_286581952](https://www.researchgate.net/figure/Block-diagram-of-the-fusion-algorithm-for-AHRS_fig1_286581952);



# Sažetak

## Bežični sustav za praćenje ljudskog pokreta u stvarnom vremenu korištenjem IMU senzora

U radu je implementiran bežični sustav za praćenje ljudskog pokreta u stvarnom vremenu koristeći IMU senzor. Cilj rada bio je razviti sustav koji omogućava precizno praćenje i vizualizaciju orijentacije ljudske ruke, s potencijalom za primjenu na bilo koji pokretni objekt ili drugi dio tijela. Sustav koristi ESP32 mikrokontroler, koji se zbog svojih mogućnosti bežične komunikacije i integracije s različitim senzorima pokazao kao idealan odabir za ovu primjenu. Korišten je IMU senzor ICM-20948 ugrađen na *9DOF 2 Click* razvojnoj pločicu, a sadrži troosni akcelerometar, žiroskop i magnetometar. Programska potpora razvijena je u programskim jezicima C (ESP32 mikrokontroler) i C# (za 3D vizualizaciju na osobnom računalu koristeći Unity okruženje). Ova integracija omogućava korisniku da u stvarnom vremenu prati 3D model čija se orijentacija u prostoru ažurira na temelju podataka sa senzora. Rad detaljno opisuje teorijske osnove potrebne za razumijevanje funkcija svih komponenti sustava, od mikrokontrolera do algoritama za obradu podataka, te pokazuje kako se pojedini dijelovi integriraju u funkcionalni prototip.

**Ključne riječi:** ESP32, IMU, Wi-Fi, SPI, AHRS, praćenje pokreta

# Summary

## **Wireless system for real-time human motion tracking based on IMU sensor**

The paper describes an implementation of a wireless system for human motion real-time tracking using an IMU sensor. The goal was to develop a system that enables precise monitoring and visualization of the orientation of the human hand, with the potential for application to any moving object or other part of the body. The system is based on ESP32 microcontroller, which proved to be the ideal choice for this application due to its wireless communication capabilities and integration with various sensors. The IMU sensor ICM-20948 installed on a 9DOF 2 Click development board was used, and it contains a three-axis accelerometer, a gyroscope and a magnetometer. The software for the system was developed in the programming languages C (for ESP32 microcontroller) and C# (for 3D visualization on a personal computer using the Unity environment). This integration allows the user to monitor a 3D model in a real time, whose orientation in space is updated based on received sensor data. The paper describes in detail the theoretical foundations necessary to understand the functions of all system components, from microcontroller to data processing algorithms, and shows how individual parts are integrated into a functional prototype.

**Keywords:** ESP32, IMU, Wi-Fi, SPI, AHRS, motion tracking