Električni kompas temeljen na Arduino platformi

Bionda, Darko

SVEUČILIŠTE U RIJECI

TEHNIČKI FAKULTET

Preddiplomski sveučilišni studij elektrotehnike

Završni rad

Električni kompas temeljen na Arduino platformi

Rijeka, rujan 2023.

Darko Bionda

0069086033

SVEUČILIŠTE U RIJECI

TEHNIČKI FAKULTET

Preddiplomski sveučilišni studij elektrotehnike

Završni rad

Električni kompas temeljen na Arduino platformi

Mentor: doc. dr. sc. Ivan Volarić

Rijeka, rujan 2023.

Darko Bionda

0069086033

SVEUČILIŠTE U RIJECI
TEHNIČKI FAKULTET
POVJERENSTVO ZA ZAVRŠNE ISPITE

Rijeka, 21. ožujka 2022.

Zavod: **Zavod za automatiku i elektroniku**
Predmet: **Elementi automatizacije postrojenja**
Grana: **2.03.03 elektronika**

ZADATAK ZA ZAVRŠNI RAD

Pristupnik: **Darko Bionda (0069086033)**
Studij: Preddiplomski sveučilišni studij elektrotehnike

Zadatak: **Električni kompas temeljen na Arduino platformi / Arduino based electric compas**

Opis zadatka:

U sklopu završnog rada potrebno je izraditi električni kompas koji se sastoji od sljedećih elemenata: Arduino razvojne pločice, magnetometra, GPS modula, te LCD zaslona. U prvoj fazi pomoću Arduino razvojne pločice na LCD zaslonu je potrebno iscrtati kompas (krug s naznačenim stranama svijeta i podjelom po stupnjevima). Zatim na temelju podataka iz magnetometra, potrebno je pronaći kut pod kojim se nalazi sjever, te sukladno tome rotirati iscrtani kompas da pokazuje prema sjeveru. Zatim je potrebno modificirati program, tako da umjesto sjevera, pokazuje prema predefiniranim geografskim koordinatama na temelju koordinata dobivenih iz GPS modula i orijentaciji sklopa prema sjeveru. Uz to, na LCD zaslonu je potrebno ispisati udaljenost do te točke.

Rad mora biti napisan prema Uputama za pisanje diplomskih / završnih radova koje su objavljene na mrežnim stranicama studija.

Zadatak uručen pristupniku: 21. ožujka 2022.

Mentor:

Doc. dr. sc. Ivan Volarić

Predsjednik povjerenstva za završni ispit:

Prof. dr. sc. Viktor Sučić

SVEUČILIŠTE U RIJECI

TEHNIČKI FAKULTET

Preddiplomski sveučilišni studij elektrotehnike

IZJAVA

U skladu s člankom 10. Pravilnika o završnom radu i završnom ispitu na preddiplomskim stručnim studijima Tehničkog fakulteta u Rijeci, izjavljujem da sam samostalno izradio završni rad za rujan 2023. godine.

Rijeka, rujan 2023.

_____

Darko Bionda

0069086033

ZAHVALA

Prvenstveno bih se zahvalio mentoru profesoru doc. dr. sc. Ivanu Volariću na prihvaćanju mentorstva i pomoći koja mi je bila potrebna prilikom izrade završnog rada. Također, zahvaljujem se svojoj obitelji koja je uvijek bila uz mene u lijepim i manje lijepim trenucima te naposljetku bi se zahvalio kolegama koji su uvijek bili spremni pomoći i koji su bili dio ovog nezaboravnog puta.

SADRŽAJ

1. UVOD

Tema završnog rada je izrada električnog kompasa s GPS modulom pomoću Arduino platforme. Potrebno je napraviti kompas koji će iz očitanih podataka s magnetometra na LCD zaslonu iscrtavati sjever. Također na temelju unesenih geografskih koordinata te koordinata korisnika dobivenih pomoću GPS modula na LCD zaslonu se iscrtava smjer i udaljenost do željenog odredišta.

U završnom radu je objašnjeno kako svaka komponenta radi zasebno, objašnjene su njihove karakteristike i kako su spojene u jednu funkcionalnu cjelinu. Također je opisan proces izrade kompasa i navedeni problemi koji su se pojavili te kako su riješeni.

Rad je realiziran u tri faze. U prvoj fazi realizirano je iscrtavanje kompasa na LCD zaslonu koji se sastoji od kruga s naznačenim stranama svijeta i podjelom na stupnjeve. U drugoj fazi rada na temelju očitanih podataka iz magnetometra potrebno je pronaći kut pod kojim se nalazi sjever te sukladno tome rotirati iscrtani kompas prema sjeveru. Treća faza se sastoji od modificiranja programa da umjesto sjevera, pokazuje prema prethodno definiranim geografskim koordinatama dobivenim iz GPS modula i orijentaciji sklopa prema sjeveru te ispisati udaljenost do te točke.

2. ARDUINO

Arduino je *open-source* platforma koja je skup elektroničkih i softverskih komponenti koje služe za jednostavnije povezivanje složenijih cjelina. Sama baza su hardver i softver koji su jednostavni i fleksibilni za korištenje. Prednost Arduina su male dimenzije koje omogućavaju realizaciju jednostavnih, ali i složenih kompaktnih sustava.

2.1. Povijest Arduina

Arduino projekt započet je od strane Instituta za dizajn interakcija Ivera u Italiji. Arduino je zapravo nadogradnja rada Wiring mikrokontrolera koji je dizajniran od strane Hernarda Barragana 2004. godine.

Massimo Banzi, Dave Mellis, Nicholas Zambetti, David Cuartielles i Gianluca Martino su originalni tvorci Arduino platforme. Prvenstveni cilj Arduino tima je bio pojednostavljenje Wiring platforme, što je rezultiralo brojnom primjenom Arduino razvojnih pločica. Pojednostavljivanjem same platforme Arduino je postao jako zastupljen i izvan tehničkih grana zbog jednostavnog učenja i brojne dokumentacije što omogućuje relativno jednostavnu realizaciju željenih ideja.

Banzi, jedan od originalnih tvoraca Arduino platforme, naveo je pet razloga po čemu je Arduino razvojna pločica izdvaja od ostalih mikrokontrolera:

1. Financijski pristupač širem broju ljudi.
2. Besplatno Arduino razvojno okruženje (IDE).
3. Jednostavnost prilikom korištenja, jednostavno spajanje s računalom (pomoću USB-a) te komunikacija se vrši pomoću standardnih serijskih protokola.
4. Velika baza izvornih kodova koji su dostupni svima.
5. Fleksibilnost. Dovoljan broj analognih i digitalnih ulaza, serijsko sučelje, SPI, I2C, digitalne i PWM izlaze.

2.2. Arduino hardver

Kao i softver koji je otvorenog koda i Arduino hardverske komponente su *open-source* karaktera, što znači da su sve njihove karakteristike i informacije dostupne svima te su dostupne za izmjenjivanje. Što se tiče licence Arduina, tvrtka je samo licencirala ime Arduino što onemogućuje kopijama da koriste ime Arduino u svom nazivu. Arduino zajednica se svakog dana nadopunjuje s novim projektima koji su dostupni svima. Iz tih razloga, Arduino razvojnu pločicu je na relativno jednostavan način moguće koristiti s brojnim komponentama drugih proizvođača.

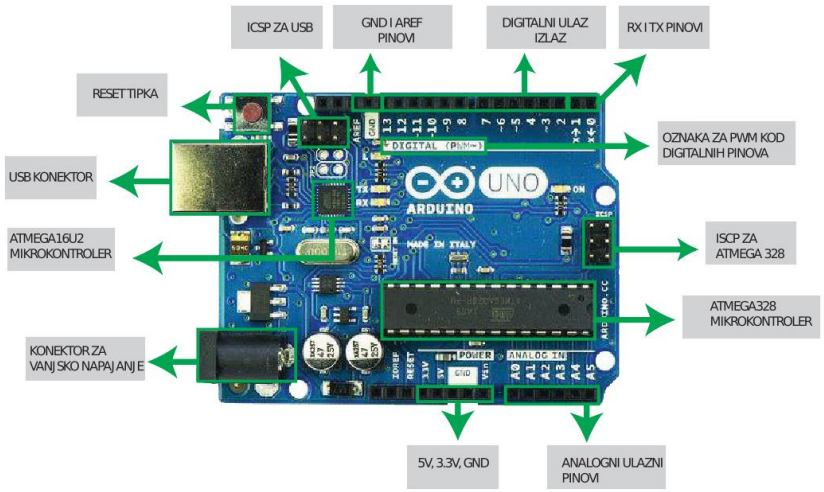
Arduino pločica sastoji se od Atmel AVR mikrokontrolera, tipično Atmega8, Atmega168, Atmega 1280, Atmega 2560, te komplementarnih komponenti koje omogućuju lakše programiranje i komunikaciju s drugim krugovima. Jedna od većih prednosti Arduino pločice je mogućnost spajanja dodatnih modula koji se nazivaju štitovi (eng. *shields*).

Mikrokontroler na Arduino pločici dolazi s unaprijed isprogramiranim bootloader-om koji omogućuje jednostavno povezivanje Arduina s računalom tj. programiranje mikrokontrolera. Sam mikrokontroler se programira pomoću RS232 serijske komunikacije, dok svaka pločica uz mikrokontroler sadrži integrirani krug za konverziju između USB i RS232 komunikacijskog protokola.

Tijekom povijesti, razvilo se mnogo Arduino modula koji se međusobno razlikuju po komponentama, primjeni i veličini. Neke od tih pločica su: Arduino UNO, Arduino Nano, Arduino Mini, Arduino Mega, itd.

2.3. Arduino UNO

Arduino UNO, najzastupljenija je Arduino pločica koja je namijenjena za početnike. Ploča je opremljena ATMEGA328P mikrokontrolerom, sadrži 14 digitalnih pinova koji se mogu koristi kao ulazni i izlazni pinovi. Osim digitalnih pinova, Arduino UNO pločice imaju 6 analognih ulaznih pinova spojenih na 10-bitni AD pretvornik kojem je zadaća da očitani analogni signal u rasponu 0 – AREF, pretvori u digitalnu vrijednost u rasponu 0-1023. Radni napon je 5V, a preporučeni ulazni napon je između 7V i 12V. Maksimalna struja za 5V pinove iznosi 40mA, dok za 3.3V pinove iznosi 50mA. Na slici 2.1 prikazana je Arduino UNO pločica s naznačenim dijelovima.



Slika 2.1 Prikaz Arduino UNO pločice s naznačenim dijelovima

Analogni pinovi na Arduino UNO ploči nalaze se u donjem desnom kutu, a označeni su s A0-A5. Pinovi su isključivo ulazni te služe za očitavanje analognih vrijednosti i napona koje se pretvaraju u 10-bitnom AD pretvorniku.

Digitalni pinovi imaju samo dvije vrijednosti: 0 i 1. U odnosu na analogne pinove, digitalni pinovi su puno brži i moguće ih je postaviti kao ulaz ili izlaz. Arduino UNO ima 14 digitalnih pinova koji su smješteni na gornjem dijelu ploče i označeni su s brojevima od 0 do 13. Definiranje digitalnih pinova poprilično je jednostavno, izvršava se pomoću naredbi `digitalRead()`, `digitalWrite()` i `pinMode()`.

Pinovi napajanja su smješteni pored analognih pinova te su označeni sa: Vin, GND, GND, 5V, 3.3V. Vin pin služi za napajanje Arduino pločice ako se koristi vanjski izvor napajanja. Raspon vrijednosti napona kod Vin pina je od 7V do 12V. GND pinovi ili uzemljenje su referentna točka. Pinovi 5V i 3.3V na svom izlazu daju stabilizirani istosmjerni napon naznačene vrijednosti koji može poslužiti za napajanje vanjskih modula.

ICSP pinovi (eng. *In Circuit Serial Programming*), omogućuju klasično programiranje mikrokontrolera pomoću SPI protokola. U slučaju ako je *bootloader*, koji se inače koristi za programiranje Arduino ploče oštećen ili nedostaje, umjesto njega može se koristiti ICPS. Klasično ICPS zaglavlje ima 6 pinova a to su: MISO, MOSI, SCK, RESET ,VCC (napajanje), GND (uzemljenje).

Kristalni oscilator je izvor radnog takta frekvencije od 16 MHz.

USB konektor (tip B) omogućuje komunikaciju između Arduino razvojne pločice i računala.

LED indikatori su na ploči označeni s RX, TX, ON i L. ON LED indikator svijetli kad je Arduino pločica pod naponom, L LED indikator je povezan s pinom 13. RX i TX su povezane na digitalne pinove 0 i 1 preko kojih se ujedno i programira mikrokontroler uz pomoć prethodno spomenutog *bootloader-a*.

RESET tipka nalazi se u gornjem lijevom kutu pločice i služi za ponovno pokretanje programskog koda u bilo kojem trenutku.

Atmega328 je 8-bitni mikrokontroler koji sadrži 32KB brze memorije u kojoj se pohranjuje programski kod, 2 kB SRAM memorije u koju se pohranjuju različite vrijednosti zapisane u varijablama koje programski kod koristi te 1KB EEPROM memorije. U tablici 2.1 prikazani su osnovni podatci o Arduino UNO pločici.

Tablica 2.1 Tehničke specifikacije Arduina

Mikrokontroler	Atmega328P
Radni napon	5V
Napon napajanja	7-12V
Digitalni ulazi/izlazi	14
PWM digitalni ulazi/izlazi	6
Analogni ulazi	6
Struja na ulazu/izlazu	40mA
Struja na 3.3V	50mA
Brza memorija	32Kb (0.5kB koristi *bootloader*)
SRAM	2kB
EEPROM	1kB
Frekvencija radnog takta	16MHz
Duljina	68.8mm
Širina	53.4mm
Masa	25g

2.4. Arduino IDE

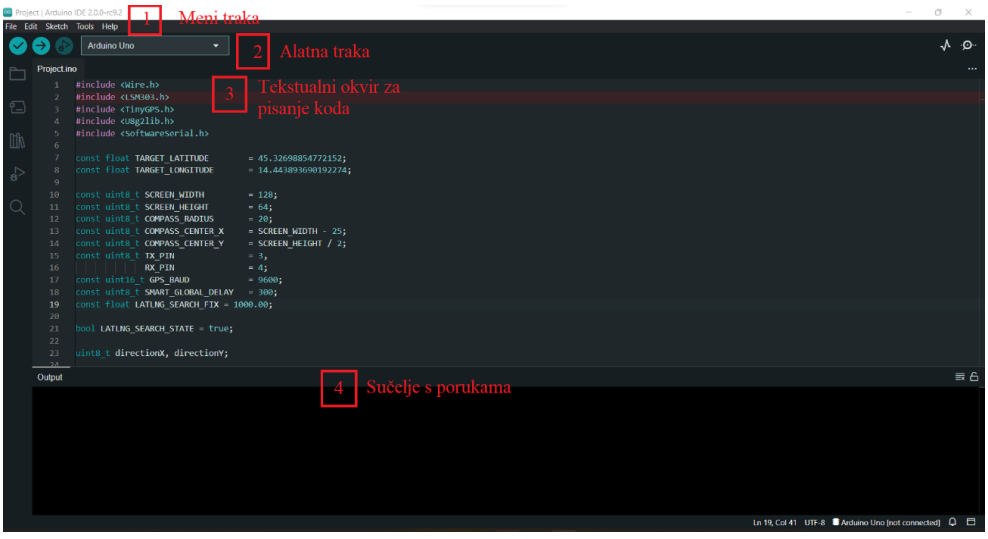
Programsko razvojno sučelje (IDE, eng. *Integrated Development Environment*) se sastoji od sučelja za pisanje kodova, serijske konzole, sekcije s porukama kompajlera i alatne trake s gumbima. Arduino razvojna pločica, kao što je već rečeno, s računalom se povezuje preko USB veze koja služi za slanje napisanog programskog koda mikrokontroleru. Da bismo to napravili, u programskom sučelju potrebno je prvo odrediti broj COM sučelja i vrstu razvojne pločice iz izbornika *Tools* nakon čega je potrebno kliknuti na *Compile*. Arduino IDE također je besplatan i dostupan svima.

Arduino IDE koristi pojednostavljeni C++. Programi koji su napisani u Arduino IDE još se nazivaju *sketches* ili skice, te datoteke imaju nastavak '.ino' . Svaki programski kod mora imati barem dvije funkcije *loop*() i *setup*(). Funkcija *setup*() se poziva jednom prilikom uključenja ili resetiranja mikrokontrolera te se uobičajeno koristi za inicijaliziraju varijable ili ulazno/izlaznih pinova. Funkcija *loop*() poziva se nakon izvršenja koda funkcije *setup*() te se iterativno izvršava sve dok je Arduino spojen na napajanje.

Biblioteke omogućuju lako korištenje velikog broja prethodno napisanih funkcija te omogućuju lakše i jednostavnije programiranje naročito prilikom rada s različitim modulima. Biblioteke se uz programski kod spremaju u *flash* memoriju mikrokontrolera, pa je potrebno voditi računa o broju deklariranih tj. korištenih biblioteka.

Na posljetku, serijska konzola služi za prikazivanje serijskih podataka koje dobivamo kroz USB priključak. Putem konzole moguće je slati podatke na Arduino upisivanjem teksta.

Na slici 2.2 prikazano je Arduino IDE sučelje.



Slika 2.2 Prikaz Arduino sučelja

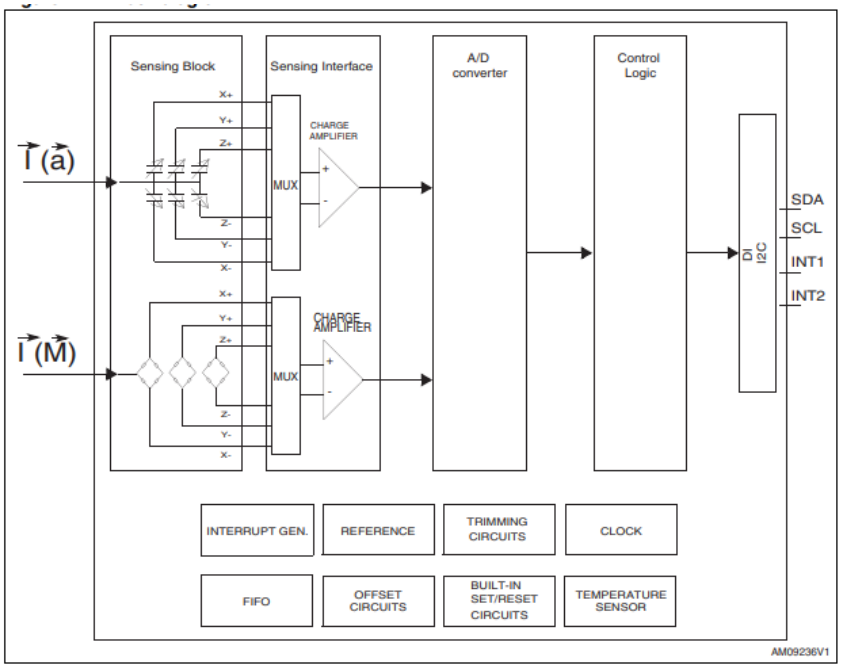
3. KOMPONENTE

U prethodnom poglavlju detaljno je opisan Arduino, kao i Arduino UNO pločica koja se koristi za izradu rada. U ovom odlomku opisan je princip rada preostalih komponenti korištenih za izradu ovog završnog rada.

3.1. Magnetometar GY-511

Glavna ideja senzora magnetometra temelji se na Lenzovom zakonu: unutar vodiča koji se nalazi u magnetskom polju inducirati će se struja. Smjer zemljinog magnetskog polja utječe na protok elektrona u senzoru. Mjerenjem ovih promjena struje, dolazimo do magnetskih tokova u sve tri prostorne ravnine, na temelju kojih možemo izračunat smjer prema sjeveru.

GY-511 modul koristi LSM303DLHC integrirani krug koji mjeri jakost magnetskog polja u sve tri osi. Komunikacijski protokol ovog modula je I2C te ima mogućnost spajanja na različite procesore uključujući Arduino pločica koristeći SCL i SDA pinove. Blokovski prikaz prethodno opisanog modula prikazan je na slici 3.1., dok je u tablici 3.1 prikazan opis pinova modula.

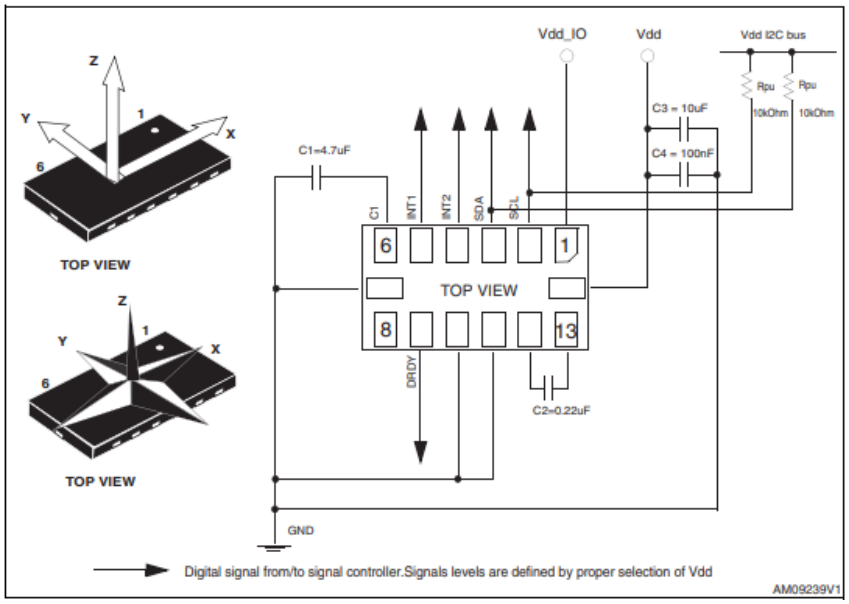


Slika 3.1 Blokovski prikaz magnetometra

Tablica 3.1 Pinovi magnetometra GY-511

Pin	Naziv	Funkcija
1	Vdd_IO	Napajanje za ulazno/izlazne pinove
2	SCL	Sučelje signala I^2C serijskog sata (SCL)
3	SDA	Sučelje signala I^2C serijskih podataka (SDA)
4	INT2	Inercijski prekid 2
5	INT1	Inercijski prekid 1
6	C1	Rezervni priključak kondenzatora (C1)
7	GND	Uzemljenje
8	Reserved	Ostavlja se nepovezan
9	DRDY	Spremni podatci
10	Reserved	Uzemljenje
11	Reserved	Uzemljenje
12	SETP	S/R priključak kondenzatora (C2)
13	SETC	S/R priključak kondenzatora (C2)
14	Vdd	Napajanje

Integrirani krug uključuje posebne senzorske elemente i IC sučelje koje može mjeriti i linearno ubrzanje i jakost magnetskog polja u sve tri osi. Senzorski elementi proizvode se pomoću specijaliziranih procesa mikrostrojne obrade, dok je sučelje integriranog kruga realizirano pomoću CMOS. LSM303DLHC ima dva signala spremnosti podataka (RDY) koji pokazuju kada je novi set mjerenja dostupan, čime olakšavaju sinkronizaciju mjerenja s mikrokontrolerom. Tipična shema spajanja LSM303DLHC prikazana je na slici 3.2.



Slika 3.2 Tipična shema spajanja magnetometra

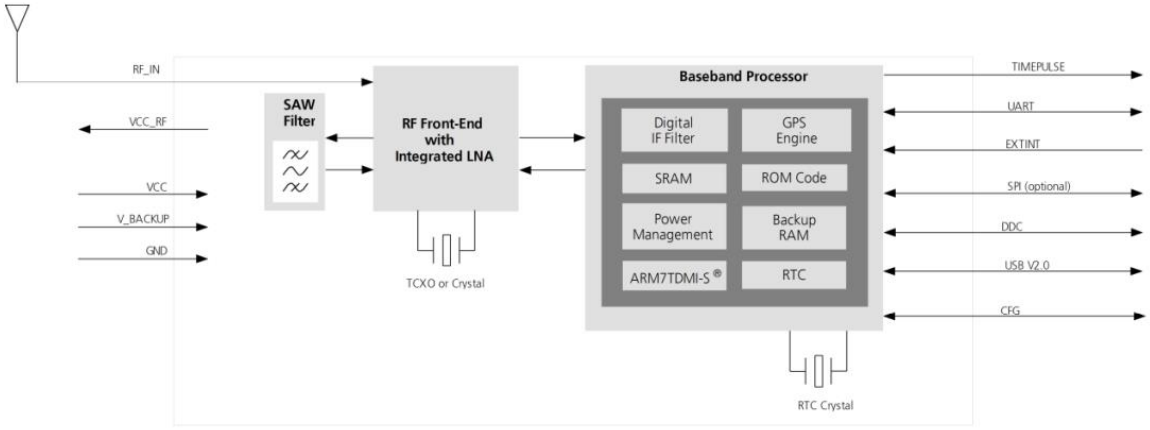
3.2. NEO-6M GPS Modul

NEO-6MV2 GPS modul je iz obitelji samostalnih GPS prijamnika koji imaju u-blox 6 integrirani krug za pozicioniranje visokih performansi. Kompaktna arhitektura, mogućnost napajanja i memorije čine NEO-6 module idealnim za uređaje napajane iz baterija s vrlo dobrim omjerom potrošnje električne energije, fizičkim dimenzijama te performansama. Na slici 3.3 prikazan je GPS modul.

Slika 3.3 Prikaz NEO-6M GPS modula

Ovaj modul jedan je od popularnih GPS modula na tržištu, gdje je uz performanse važno napomenuti i relativno nisku cijenu modula. Podatci o lokaciji koje nudi dovoljno su točni da

zadovolje većinu aplikacija, a često se koristi kao GPS prijamnik u pametnim telefonima i tabletima. Na slici 3.4 prikazana je blokovska shema NEO-6M GPS modula, dok je u tablici 3.2 navedena funkcija četiri izlazna pina.



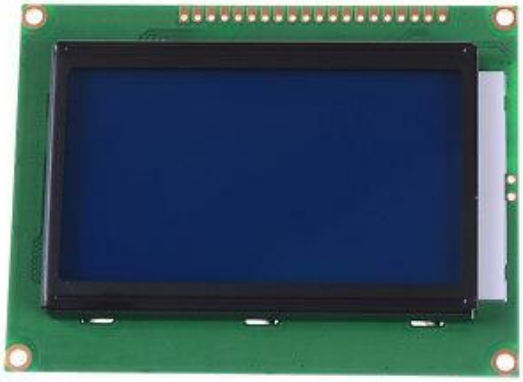
Slika 3.4 Blokovski prikaz NEO-6M GPS modula

Tablica 3.2 Tablica pinova GPS NEO-6M modula

Naziv pina	Opis
VCC	Pin napajanja
RX	UART prijemni pin
TX	UART prijenosni pin
GND	Uzemljenje

3.3. LCD modul ST7920

ST7920 LCD modul može prikazati znakove abecede, brojeve, kineske fontove te samo-definirane znakove. S modulom se može komunicirati na tri načina: preko 8- ili 4-bitne paralelne sabirnice ili serijski. Osim LCD zaslona, modul sadržava integrirani krug u kojem se nalazi RAM zaslon, ROM podržanih karaktera, te cjelokupna upravljačka i komunikacijska elektronika. U integriranom znakovnom ROM-u definirano je 8192 kineskih karaktera dimenzija 16x16 piksela, te 126 alfanumeričkih karaktera s dimenzijama 16x8 piksela. Također za grafički prikaz podržava područje grafičkog prikaza od 64x256 točaka（GDRAM）i ICON RAM od 240 točaka. Na slici 3.5 prikazan je LCD ST7920 modul.



Slika 3.5 Prikaz LCD modula ST7920

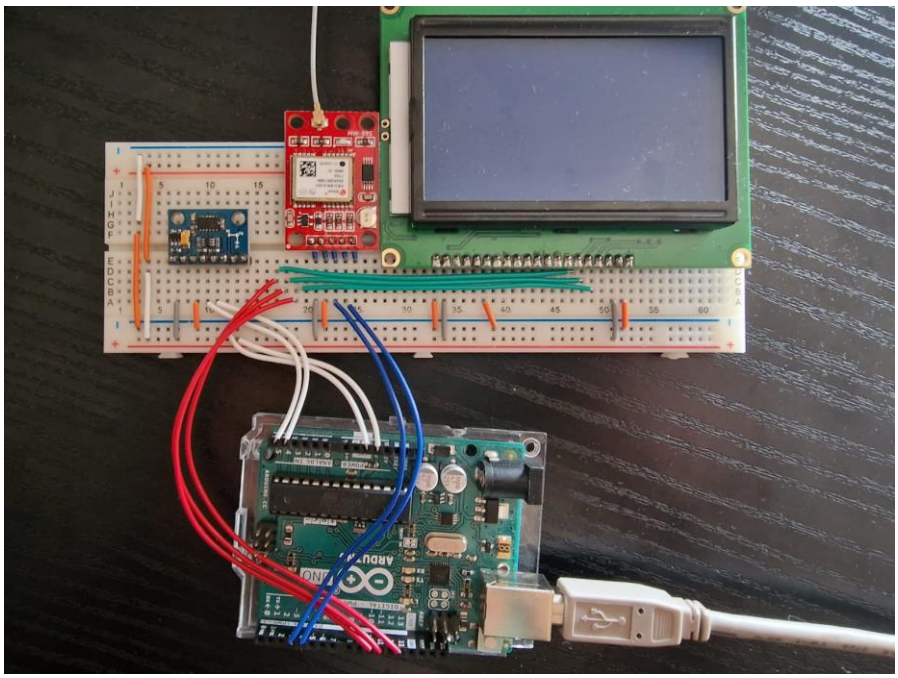
U tablici 3.3. upisani su pinovi korištenog LCD modula, njihova funkcija i Arduino pinovi na koje su spojeni u ovom završnom radu.

Tablica 3.3 Tablica pinova LCD panela ST7920

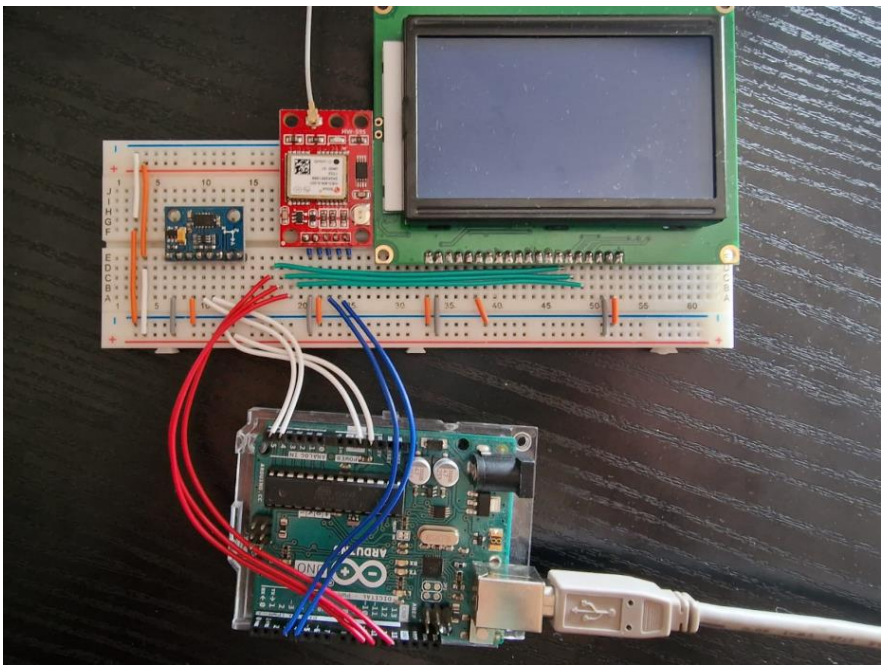
LCD pin	Oznaka	Funkcija	Arduino pin
1	GND	Uzemljenje	Uzemljenje
2	VCC	Napajanje	
3	V0	Kontrast	
4	RS (izbor čipa)	PSB = 1 0 – na paralelnoj sabirnici se nalaze podatci za prikaz 1 - na paralelnoj sabirnici se nalaze instrukcije PSB = 0 Serijski ulaz	D8
5	RW(podatci, MOSI)	Čitanje/pisanje	D7
6	E (CLK)	PSB = 1 Pin za omogućavanje rada	D5

		PSB = 0 Radni takt za serijsku komunikaciju	
7	DB0	Paralelni podatci	
8	DB1	Paralelni podatci	
9	DB2	Paralelni podatci	
10	DB3	Paralelni podatci	
11	DB4	Paralelni podatci	
12	DB5	Paralelni podatci	
13	DB6	Paralelni podatci	
14	DB7	Paralelni podatci	
15	PSB	0 – serijska komunikacija 1 – paralelna komunikacija	uzemljenje
16	NC		
17	RST	Resetiranje	
18	Vout		
19	BLA	Napajanje pozadinskog osvjetljenja	3.3V
20	BLK	GND	GND

Prethodno opisane komponente s Arduino razvojnom pločicom spojene su prema shemi prikazanoj na slici 3.6, dok je konačni izgled sklopa prikazan na slici 3.7.



Slika 3.6 Shema završnog spoja



Slika 3.7 Konačni spoj na ispitnoj pločici

4. IZRADA KODA

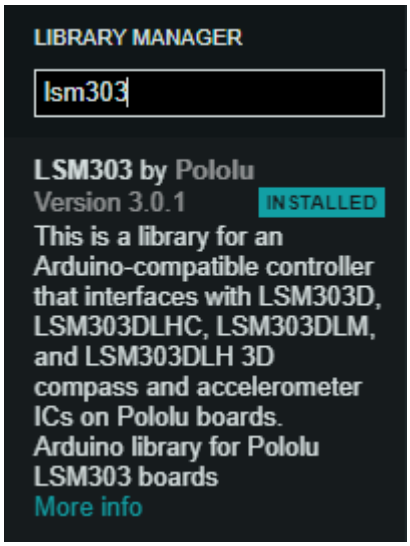
Izrada programskog koda započela je s pisanjem osnovnog Arduino programa koji se sastoji od jedne *setup* i *loop* funkcije.

```
void setup()
{
        Serial.begin(9600);
}


void loop()
{
        Serial.println("Hello World!");
}
```

Naredba Serial.begin(9600) uspostavlja serijsku komunikaciju s Arduino razvojnom pločicom brzinom 9600 bitova po sekundi.

4.1. Magnetometar

Prvi korak u izradi završnog rada je omogućavanje rada magnetometra GY-511 za što je korištena biblioteka LSM303 prikazana na slici 4.1.



Slika 4.1 Prikaz LSM303 biblioteke

U nastavku je dan kod za testiranje funkcionalnosti magnetometra:

```
#include <Wire.h>
#include <LSM303.h>

LSM303 compass;


void setup()
{
	Serial.begin(9600);

	Wire.begin();
	compass.init();
	compass.enableDefault();
}


void loop()
{
	compass.read();

	Serial.println("Hello World!");
}
```

Na početku programa, potrebno je uključiti pripadne biblioteke. Biblioteka *Wire.h* omogućuje komunikaciju s I2C protokolom. Prije *setup* funkcije deklarirana je globalna varijabla *compass* pomoću konstruktora iz knjižnice LSM303.h.

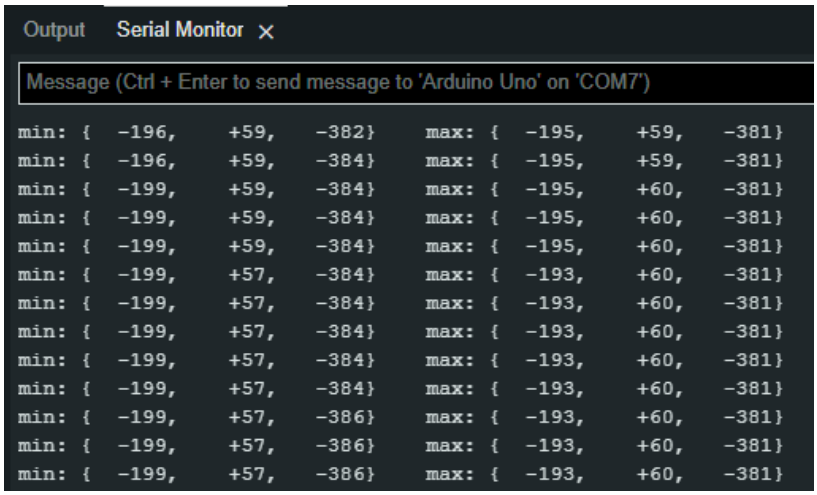
4.1.1. Kalibracija magnetometra

Sljedeći korak u izradi kompasa je kalibracija magnetometra. Kod za kalibraciju magnetometra dolazi uz dokumentaciju uređaja:

```
#include <Wire.h>
#include <LSM303.h>

LSM303 compass;
LSM303::vector<int16_t> running_min = {32767, 32767, 32767}, running_max = {-32768, -32768, -32768};

char report[80];


void setup()
{
	Serial.begin(9600);
	Wire.begin();
	compass.init();
```

```
        compass.enableDefault();
}


void loop()
{
        compass.read();

        running_min.x = min(running_min.x, compass.m.x);
        running_min.y = min(running_min.y, compass.m.y);
        running_min.z = min(running_min.z, compass.m.z);

        running_max.x = max(running_max.x, compass.m.x);
        running_max.y = max(running_max.y, compass.m.y);
        running_max.z = max(running_max.z, compass.m.z);

        snprintf(report, sizeof(report), "min: {%+6d, %+6d, %+6d}    max: {%+6d, %+6d, %+6d}",
                running_min.x, running_min.y, running_min.z,
                running_max.x, running_max.y, running_max.z);
        Serial.println(report);

        delay(100);
}
```

Pokretanjem koda za kalibraciju kompasa u serijskoj konzoli se ispisuju željene vrijednosti, kao što je to prikazano na slici 4.2.



Slika 4.2 Prikaz očitanih podataka kalibracije

Prilikom kalibracije magnetometar je potrebno zakretati u svim smjerovima dok vrijednosti koje ispisuje ne postanu stabilne. Proces traje 3 do 5 minuta. Dobivene informacije o kalibraciji potrebno je iskoristiti u početnom kodu na sljedeći način:

```
#include <Wire.h>
#include <LSM303.h>

LSM303 compass;

void setup()
{
	Serial.begin(9600);

	Wire.begin();
	compass.init();
	compass.enableDefault();

	compass.m_min = (LSM303::vector<int16_t>){-527,  -548,  -394};
	compass.m_max = (LSM303::vector<int16_t>){+225,  +81,  -63};
}


void loop()
{
	compass.read();

	Serial.println("Hello World!");
}
```

4.1.2. Orijentacija magnetometra

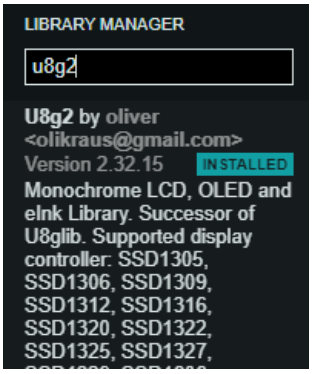
Nakon izvršene kalibracije, iz magnetometra možemo iščitati informaciju koja je potrebna u ovom završnom radu. Orijentacija, odnosno kut magnetometra prema sjeveru dobivamo pozivom funkcije *heading*. Pritom treba voditi računa i o magnetskoj deklinaciji, koja za Rijeku iznosi 4.2.

```
#include <Wire.h>
#include <LSM303.h>

const float MAGNETIC_DECLINATION    = 4.2;

LSM303 compass;

void setup()
{
	Serial.begin(9600);

	Wire.begin();
	compass.init();
	compass.enableDefault();

	compass.m_min = (LSM303::vector<int16_t>){-527,  -548,  -394};
	compass.m_max = (LSM303::vector<int16_t>){+225,  +81,  -63};
}


void loop()
{
	compass.read();

	uint16_t heading = compass.heading() + MAGNETIC_DECLINATION;
```

```
        Serial.println("Hello World!");
}
```

4.2. Ispis LCD zaslona

Sljedeći korak u izradi rada je spajanje LCD zaslona te iscrtavanje kompasa. Za početak trebamo odabrati biblioteku U8G2 prikazanu na slici 4.3 s postupkom identičnim onim za magnetometar.



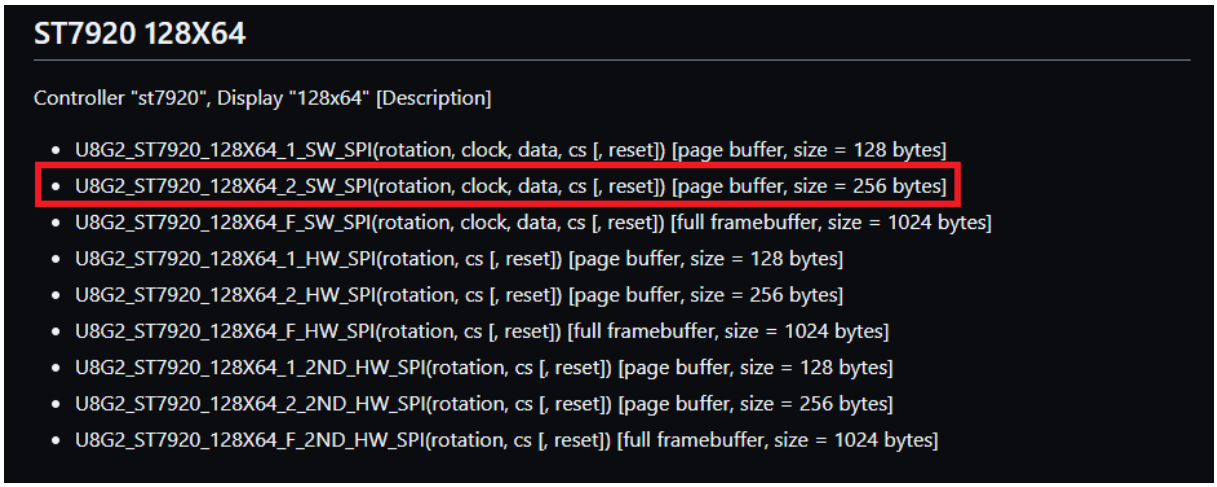
Slika 4.3 Prikaz U8G2 biblioteke

Zatim su dodane sljedeće linije koda:

```
#include <Wire.h>
#include <LSM303.h>
#include <U8g2lib.h>

const float MAGNETIC_DECLINATION    = 4.2;

LSM303 compass;
U8G2_ST7920_128X64_2_SW_SPI u8g2(U8G2_R0, 13, 11, 10);


void setup()
{
        Serial.begin(9600);

        u8g2.begin();
        u8g2.setFont(u8g2_font_6x10_tf);

        Wire.begin();
        compass.init();
        compass.enableDefault();

        compass.m_min = (LSM303::vector<int16_t>){-527,   -548,   -394};
        compass.m_max = (LSM303::vector<int16_t>){+225,    +81,    -63};
}


void loop()
{
        compass.read();

        uint16_t heading = compass.heading() + MAGNETIC_DECLINATION;

        u8g2.firstPage();
```

```
        do
        {
                u8g2.setCursor(1, 8);
                u8g2.print(heading);
        } while ( u8g2.nextPage() );
}
```

Kao i u prethodnom slučaju, prije *setup()* funkcije uključili smo korištenu knjižnicu i deklarirali varijablu *U8G2* korištenjem odgovarajućeg konstruktora. Knjižnica ima nekoliko konstruktora, te je potrebno ispravno odabrati ovisno o korištenom LCD zaslonu, načinu komunikacije te količini RAM memorije. U dokumentaciji knjižice slika 4.4 pronađeni je odgovarajući konstruktor. Odabrani konstruktor ima četiri parametra: rotaciju, pin za radni takt, podatkovni pin i liniju odabira integriranog kruga.



Slika 4.4 Prikaz odabranog konstruktora koji odgovara odabranom LCD zaslonu

U *setup()* funkciji LCD zaslon je inicijaliziran te je odabrani font ispisa, dok je u *loop()* funkciju dodani kod koji na LCD zaslonu ispisuje podatak dobiven iz magnetometra. Dobiveni ispis je prikazan na slici 4.5.

Slika 4.5 Prikaz ispisanih podataka na LCD zaslonu

4.2.1. ISCRTAVANJE KRUGA NA LCD ZASLONU

Sljedeći korak je iscrtavanje kompasa na LCD zaslonu, što podrazumijeva iscrtavanje kruga s podjelom po stupnjevima i pravilno označenim stranama svijeta pomoću prethodno dobivene informacije iz magnetometra.

Da bismo olakšali čitljivost programskog koda, podijeljen je u funkcije.

```
 void show_compass_heading(uint16_t heading)
{
        u8g2.setCursor(56, 9);
        u8g2.print(heading);

        u8g2.drawUTF8(73, 9, "°"); // X, Y, simbol stupnja
}


void draw_compass(uint16_t heading)
{
        compass_circle();
        compass_notches();
        compass_direction_letters();
        compass_arrow();

}

void U8G2_draw()
{
        uint16_t heading = compass.heading() + MAGNETIC_DECLINATION;

        show_compass_heading(heading);
        draw_compass(heading);
}
```

U *loop()* funkciju se poziva funkcija *U8G2_draw()* koja iščitava vrijednost magnetometra te poziva dvije funkcije. Prva funkcija *draw_compass* ispisuje dobivenu orijentaciju, dok funkcija *draw_compass* poziva četiri funkcije koje iscrtavaju dijelove kompasa.

Iznad prethodno deklarirane konstante *MAGNETIC_DECLINATION* potrebno je definirati tri nove konstante koje opisuju visinu i širinu LCD zaslona, te konstantu koja opisuje radijus kompasa.

```
const uint8_t SCREEN_WIDTH        = 128;
const uint8_t SCREEN_HEIGHT       = 64;
const uint8_t COMPASS_RADIUS      = 20;
const float MAGNETIC_DECLINATION   = 4.2;
```

Zatim je u funkciju *compass_circle()* dodana naredba za ispis kruga na LCD zaslonu. Korištena funkcija prima četiri argumenta: x i y koordinatu ishodišta, radijus i stil linije. Ispis je prikazan na slici 4.6.

```
const uint8_t COMPASS_RADIUS      = 20;
const uint8_t COMPASS_CENTER_X    = SCREEN_WIDTH / 2;
const uint8_t COMPASS_CENTER_Y    = SCREEN_HEIGHT / 2;
const float MAGNETIC_DECLINATION   = 4.2;


void compass_circle()
{
        u8g2.drawCircle(SCREEN_WIDTH / 2, SCREEN_HEIGHT / 2, COMPASS_RADIUS,
U8G2_DRAW_ALL);
}
```

Centar kompasa je određen tako što se središte kružnice postavilo na polovicu visine, odnosno širine LCD zaslona.

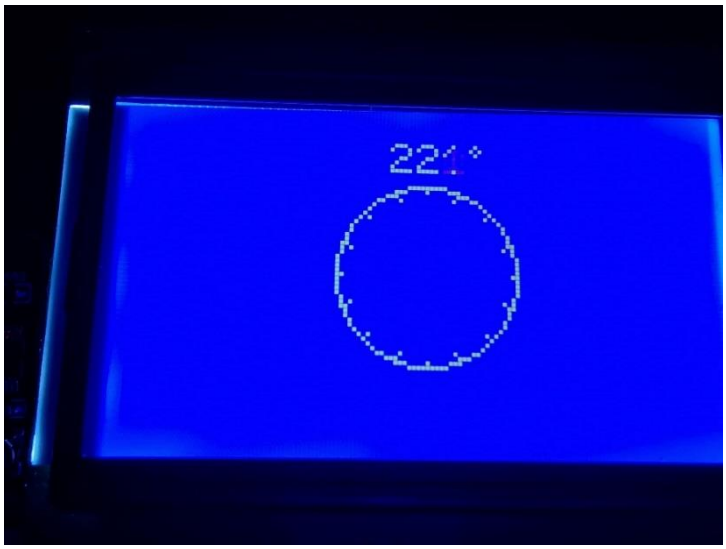
Slika 4.6 Prikaz iscrtanog kruga na LCD zaslonu

```
const uint8_t COMPASS_RADIUS       = 20;
const uint8_t COMPASS_CENTER_X     = SCREEN_WIDTH / 2;
const uint8_t COMPASS_CENTER_Y     = SCREEN_HEIGHT / 2;
const float MAGNETIC_DECLINATION   = 4.2;
```

U funkciji *compass_notches()* realizirano je iscrtavanje podjele kompasa.

```
void compass_notches()
{
        int8_t dxo, dyo, dxi, dyi;

        for (float i = 0; i < 360; i = i + 22.5)
        {
                dxo = COMPASS_RADIUS * cos(i * PI / 180);
                dyo = COMPASS_RADIUS * sin(i * PI / 180);
                dxi = dxo * 0.95;
                dyi = dyo * 0.95;
                u8g2.drawLine(dxi + COMPASS_CENTER_X, dyi + COMPASS_CENTER_Y, dxo +
COMPASS_CENTER_X, dyo + COMPASS_CENTER_Y);
        }
}
```

Unutar funkcije prvo su deklarirane četiri pomoćne varijable: dx0, dy0, dxi i dyi. Varijable dx0 i dy0 označavaju početak linije, a varijable dxi i dyi kraj linije. Iterator *for* petlje ima raspon od 0 do 360 te inkrementiramo varijablu s 22.5 jer želimo prikazati 16 crtica (360/22,5=16). Unutar petlje uz pomoć trigonometrije dobiven je početak crtica na krunici, dok se njihovi krajevi definiraju množenjem istih sa faktorom 0.95. U zadnjoj liniji koda unutar *for* petlje iscrtana je podjela s naredbom *drawLine* koja prima četiri argumenta: x i y koordinate početka i kraja crtice. Na slici 4.7 prikazan je ispis na LCD ekranu.



Slika 4.7 Prikaz iscrtanog kruga podjeljen po jednakim dijelovima na LCD zaslonu

4.2.2. Ispis strana svijeta

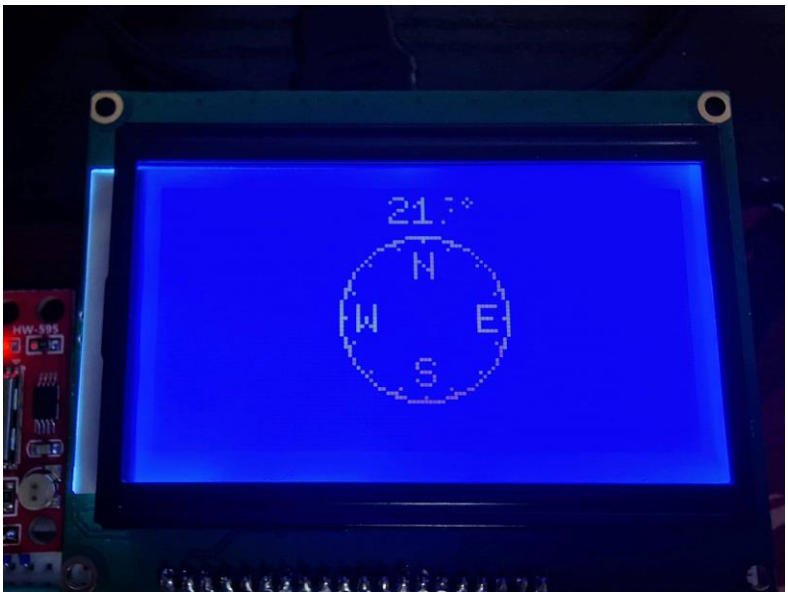
Za ispisivanje strana svijeta na obodu kružnice, prvo je definirana funkcija *compass_letter* koja prima tri argumenta: x i y koordinate te karakter kojeg je potrebno ispisati.

```
void compass_letter(uint8_t x, uint8_t y, char dir[10])
{
        u8g2.setCursor(x, y);
        u8g2.print(dir);
}
```

Zatim u funkciji *compass_direction_letters* pozivamo prethodnu funkciju četiri puta za četiri strane svijeta.

```
void compass_direction_letters()
{
        compass_letter((COMPASS_CENTER_X - 2),(COMPASS_CENTER_Y - 9), "N");
        compass_letter((COMPASS_CENTER_X - 2),(COMPASS_CENTER_Y + 17), "S");
        compass_letter((COMPASS_CENTER_X + 13),(COMPASS_CENTER_Y + 4), "E");
        compass_letter((COMPASS_CENTER_X - 16),(COMPASS_CENTER_Y + 4), "W");
}
```

Ispis na ekranu prikazan je na slici 4.8.



Slika 4.8 Prikaz kruga s naznačenim stranama svijeta

4.2.3. Iscrtavanje strelice

Na vrhu programa deklarirane su dvije nove globalne varijable:

```
uint8_t directionX, directionY;
```

Za početak, uz pomoć trigonometrije potrebno je izračunati X i Y koordinatu točke na kružnici koja pokazuje prema sjeveru (u odnosu na ishodište) pomoću podataka iz magnetometra. To je napravljeno unutar *draw_compass* funkcije, prije samog poziva naredbe *compass_arrow* funkcije koja je zadužena za iscrtavanje same strelice.

```
void draw_compass(uint16_t heading)
{
        compass_circle();
        compass_notches();
        compass_direction_letters();

        directionX = (0.7 * COMPASS_RADIUS * cos((heading - 90) * PI / 180)) + COMPASS_CENTER_X;
        directionY = (0.7 * COMPASS_RADIUS * sin((heading - 90) * PI / 180)) + COMPASS_CENTER_Y;

        compass_arrow(directionX, directionY, COMPASS_CENTER_X, COMPASS_CENTER_Y, 2, 2);
}
```

Poteškoća koja se pojavila prilikom crtanja strelice bila je mali broj piksela koja je riješena množenjem s 0.7 čime se dobije najbolji prikaz strelice.

Unutar funkcije *comapss_arrow* deklarirane su četiri varijable: X1, X2, Y1 i Y2, koje predstavljaju točke pravca. Kako bi se dobio kut igle definirane su dodatne varijable : dx, dy, x2o, y2o, x3, y3, x4, y4. Sljedeći korak je računanje duljine igle i spremanje u varijablu distance. Nakon toga definiran je izraz za pomoćnu udaljenost, te će se koristiti kako bi se nacrtala glava igle (oblik trokuta).

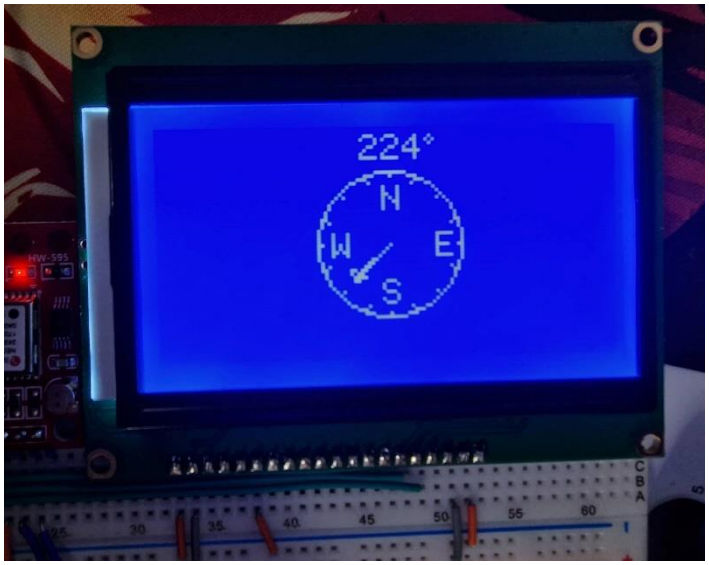
```
float distance;

int8_t dx, dy, x2o, y2o, x3, y3, x4, y4, coefficientDirection;

distance = sqrt(pow((x1 - x2), 2) + pow((y1 - y2), 2));

dx = x2 + (x1 - x2) * arrowLength / distance;
dy = y2 + (y1 - y2) * arrowLength / distance;

coefficientDirection = arrowWidth / arrowLength;
```

Nakon određivanja smjera strelice, potrebno je dovršiti donji dio strelice za što ćemo koristiti:

```
x2o = x2 - dx;
y2o = dy - y2;

x3 = y2o * coefficientDirection + dx;
y3 = x2o * coefficientDirection + dy;

x4 = dx - y2o * coefficientDirection;
y4 = dy - x2o * coefficientDirection;
```

Na posljetku, pomoću biblioteke *u8g2* izvršava se funkcija *drawLine()* te se strelica iscrtava na panelu. Na slici 4.9 prikazan je ispis kompasa.

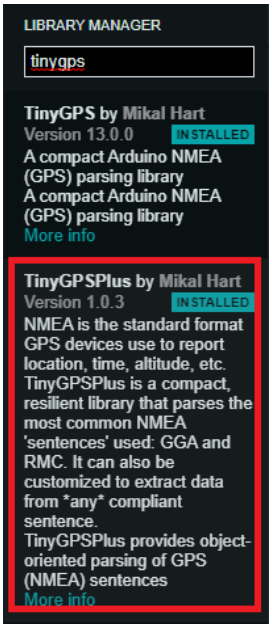
```
u8g2.drawLine(x1, y1, x2, y2);
u8g2.drawLine(x1, y1, dx, dy);
u8g2.drawLine(x3, y3, x4, y4);
u8g2.drawLine(x3, y3, x2, y2);
u8g2.drawLine(x2, y2, x4, y4);
```



Slika 4.9 Prikaz iscrtanog kompasa

4.3. GPS modul

Zadnji dio ovog programa je iščitavanje podataka iz GPS modula. Korištena je biblioteka *TinyGPS++* koja je prikazana na slici 4.10.



Slika 4.10 Prikaz odabrane TinyGPS++ biblioteke

Kao i u slučaju za prethodne module, na početku programa uključit ćemo potrebne biblioteke. Uz *TinyGPS++.h* potrebno je uključiti i *SoftwareSerial.h* knjižicu koja omogućuje kreiranje serijskog sučelja na bilo koja dva Arduino pina.

```
#include <Wire.h>
#include <LSM303.h>
#include <U8g2lib.h>
#include <TinyGPS++.h>
#include <SoftwareSerial.h>
...
```

Također, deklarirane su tri konstante koje definiraju komunikaciju između Arduino razvojne pločice i GPS modula: RX i TX pinove, te brzinu serijskog sučelja. Brzina serijskog sučelja definirana je u dokumentaciji GPS modula.

```
const uint8_t COMPASS_CENTER_X      = SCREEN_WIDTH / 2;
const uint8_t COMPASS_CENTER_Y      = SCREEN_HEIGHT / 2;
const float MAGNETIC_DECLINATION    = 4.2;
const uint8_t TX_PIN                = 3,
              RX_PIN                = 4;
const uint16_t GPS_BAUD             = 9600;
...
```

Zatim su deklarirane dvije globalne varijable. Varijable gps s konstruktorom iz *TinyGPS++* biblioteke i *gpsSerial* s konstruktorom iz *SoftwareSerial* biblioteke koja prima dva argumenta: RX i TX pinove serijskog sučelja.

```
LSM303 compass;
U8G2_ST7920_128X64_2_SW_SPI u8g2(U8G2_R0, 13, 11, 10);
TinyGPSPlus gps;
```

```
SoftwareSerial gpsSerial(TX_PIN, RX_PIN);
```

Unutar *setup()* funkcije potrebno je otvoriti serijsku komunikaciju s brzinom *GPS_BAUD*.

```
void setup()
{
	Serial.begin(9600);
	gpsSerial.begin(GPS_BAUD);

	u8g2.begin();
	u8g2.setFont(u8g2_font_6x10_tf);
...
```

Kada Arduino očita podatke s GPS modula potrebno je iščitati iste. Iščitavanje podataka vrši se putem serijske komunikacije. Na serijsku komunikaciju podaci dolaze brže nego što se promijene, mogu se prikazati na LCD zaslonu, stoga moramo dodati odgodu u samoj *loop* funkciji. Dodaje se funkcija iznad *loop* funkcije:

```
static void smart_delay(unsigned long ms)
{
	unsigned long start = millis();

	do
	{
		while (gpsSerial.available())
		{
			gps.encode(gpsSerial.read());
		}
	} while (millis() - start < ms);
}


void loop()
...
```

Funkcija prima varijablu koja ima ulogu odgode. U varijablu *start* se spremna vrijednost vraća u funkciju *millis()*, čija je uloga vraćanje broja milisekundi proteklih od uključivanja mikrokontrolera. Unutar funkcije kreiramo *do while* petlju pomoću koje iščitavamo podatke sa GPS-a na serijski međuspremnik dok na *serial buffer* za ono vrijeme u kojem se šalje kroz argument funkcije. Potrebno je unutar *do while* petlje kreirati unutarnju *while* petlju pomoću koje se iščitavaju podatci sa serijske komunikacije. Potrebno je kontinuirano iščitavati podatke dokle god na *serial bufferu* postoje dostupni podatci jer u protivnom nestaje prikaz na LCD zaslonu.

U sljedećem koraku definiramo globalnu varijablu *SMART_GLOBAL_DELAY*.

```
const uint8_t TX_PIN		= 3,
	RX_PIN		= 4;
const uint16_t GPS_BAUD		= 9600;
const uint8_t SMART_GLOBAL_DELAY	= 300;
...
```

Prethodno definiranu funkciju pozivamo u *loop* funkciji:

```
void loop()
{
        compass.read();

        smart_delay(SMART_GLOBAL_DELAY);

        uint16_t heading = compass.heading() + MAGNETIC_DECLINATION;

        u8g2.firstPage();
        do
        {
                U8G2_draw();
        } while ( u8g2.nextPage() );

        delay(SMART_GLOBAL_DELAY);
}
```

4.3.1. Orijentacija kompasa

Svrha završnog rada je pokazivanje smjera prema određenoj lokaciji.

```
#include <U8g2lib.h>

const float TARGET_LATITUDE          = 45.32698854772152;
const float TARGET_LONGITUDE         = 14.443893690192274;
const uint8_t SCREEN_WIDTH           = 128;
...
```

Sljedeći korak je izračun udaljenosti prilikom čega je potrebno očitati podatke s GPS modula:

```
gps.location.lat()
gps.location.lng()
```

Za izračun udaljenosti se koristi funkcija *distanceBetween* iz biblioteke *TinyGPS++* koja prima četiri argumenta: geografske koordinate trenutne lokacije i željenog odredišta.

```
uint16_t heading = compass.heading() + MAGNETIC_DECLINATION;

      unsigned long distance = (unsigned long)TinyGPSPlus::distanceBetween(
            gps.location.lat(),
            gps.location.lng(),
            TARGET_LATITUDE,
            TARGET_LONGITUDE
      );
...
```

Jedna od problematičnijih stvari kod GPS modula je što neke od željenih funkcionalnosti zahtijevaju kretanje određenom brzinom, što znači da će biti nedostupne u stacionarnom stanju.

Da bi odredili smjer prema krajnjim koordinatama, korištena je *courseTo* funkcija. Radimo ga na isti način kao i udaljenost između nas, samo sto ovaj put koristimo *courseTo* funkciju.

U funkciji to_target pozvana je *courseTo* funkcija s 4 argumenta: geografskim koordinatama odredišta, te onih dobivenih iz GPS modula. Pozivom ove funkcije omogućuje se rad funkcijama unutar *TinyGPSPlus* klase koje ovise o ovoj funkciji.

```
double course_to_target = TinyGPSPlus::courseTo(
                gps.location.lat(),
                gps.location.lng(),
                TARGET_LATITUDE,
                TARGET_LONGITUDE
        );
```

Nakon poziva *courseTo* funkcije, moguće je koristiti funkciju *course.deg()* koja vraća traženi kut, no ova funkcionalnost je dostupna samo prilikom gibanja GPS modula pa je prvo potrebno izvršiti provjeru je li informacija dostupna na sljedeći način. Ukoliko nije, varijabla *course_degrees* poprima vrijednost 0.

```
double course_degrees = gps.course.isValid() ? gps.course.deg() : 0;
```

4.4. Završni detalji

Kako bi ovaj završni rad bio u potpunosti dovršen, potrebno je još nekoliko detalja. Sustav treba raditi u dva načina rada: ako su koordinate cilja jednake nuli, sustav radi kao klasični kompas te ako imamo određeni cilj, sustav treba prikazivati smjer i udaljenost na LCD zaslonu.

Prvo je definirana varijabla *bool* tipa koju ćemo nazvat *IS_TARGET_ZERO* koja će u početku biti definirana s *false*, no promjenom koordinata na nule, postati će *true*.

```
uint8_t directionX, directionY;

bool IS_TARGET_ZERO             = false;
...
```

Zbog rasporeda elemenata na LCD zaslonu, varijable *COMPASS_CENTER_X* i *COMPASS_CENTER_X* pozicioniramo ispred varijabli *directionX* i *direction_Y*. Pomoću ove promjene svi elementi iscrtani na LCD zaslonu nalazit će se na desnoj strani.

```
const uint8_t COMPASS_RADIUS        = 20;
const uint8_t COMPASS_CENTER_X      = SCREEN_WIDTH / 2;
const uint8_t COMPASS_CENTER_Y      = SCREEN_HEIGHT / 2;
const float MAGNETIC_DECLINATION    = 4.2;
const uint8_t TX_PIN                = 3,
              RX_PIN                = 4;
const uint16_t GPS_BAUD             = 9600;
const uint8_t SMART_GLOBAL_DELAY    = 300;
```

```
uint8_t COMPASS_CENTER_X             = SCREEN_WIDTH - 25;
uint8_t COMPASS_CENTER_Y             = SCREEN_HEIGHT / 2;
uint8_t directionX, directionY;
...
```

Unutar *setup* funkcije kreirana je *if* petlja. U slučaju da su koordinate cilja jednake nuli tada *IS_TARGET_ZERO* mijenja vrijednost u *true* te se kompas centrira na sredinu ekrana i ima funkciju rada kao klasični kompas.

```
void setup()
{
      Serial.begin(9600);
      gpsSerial.begin(GPS_BAUD);

      u8g2.begin();
      u8g2.setFont(u8g2_font_6x10_tf);

      Wire.begin();
      compass.init();
      compass.enableDefault();

      compass.m_min = (LSM303::vector<int16_t>){-527,   -548,   -394};
      compass.m_max = (LSM303::vector<int16_t>){+225,    +81,    -63};

      if (TARGET_LATITUDE == 0.00 && TARGET_LONGITUDE == 0.00)
      {
            IS_TARGET_ZERO = true;

            COMPASS_CENTER_X = SCREEN_WIDTH / 2;
            COMPASS_CENTER_Y = SCREEN_HEIGHT / 2;
      }
}
...
```

U sljedećem koraku u *U8G2_draw* funkciju prosljeđuju se varijable *distance* i *course_degrees* kako bi se proslijedile drugim funkcijama:

```
do
{
      U8G2_draw(distance, course_degrees);
} while ( u8g2.nextPage() );
...
```

Unutar te funkcije *U8G2_draw*, funkcijama *show_compass_heading* i *draw_compass* prosljeđuju se te varijable *distance* i *course*, te će se nove funkcije pozvati ukoliko su definirane koordinate odredišta.

```
void U8G2_draw(unsigned long d, double cd)
{
  uint16_t heading = compass.heading() + MAGNETIC_DECLINATION;

  show_compass_heading(heading, cd);
  draw_compass(heading, cd);

  if (!IS_TARGET_ZERO)
  {
```

```
    show_lat_lng();
    show_distance_from_target(d);
  }
}
...
```

Zatim definiramo dvije nove globalne varijable:

```
bool LATLNG_SEARCHING_STATE          = true;
bool IS_TARGET_ZERO                  = false;
bool IS_TARGET_INVALID               = true;
```

U slučaju da je LAT veći od 90 odnosno manji od -90, LNG veći od 180 odnosno manji od -180 na LCD panelu će se ispisati WRNGTRGT, tj. krivi cilj.

```
void show_distance_from_target(unsigned long distance)
{
  u8g2.setCursor(1, 63);
  u8g2.print(F("Distance: "));

  if (!IS_TARGET_INVALID)
  {
            if (LATLNG_SEARCHING_STATE)
            {
                  u8g2.setCursor(58, 63);
                  u8g2.print(F("CALCULATING"));
            }

            else
            {
                  u8g2.setCursor(57, 63);
                  u8g2.print(distance);

                  u8g2.setCursor(100, 63);
                  u8g2.print(F("m"));
            }
      }

      else
      {
            Serial.println(F("Wrong target coordinates. Make sure ( -90 > LAT
< 90 ) and ( -180 > LNG < 180 )!!!"));
            u8g2.setCursor(57, 63);
            u8g2.print(F("WRGTRGT"));
      }
}
```

U slučaju da GPS nije dobio potreban signal za prikaz trenutne lokacije i udaljenosti od odredišta tada će *LATLNG_SEARCHING_STATE* imati vrijednost *true*, te će se na LCD panelu ispisati: *LAT: SEARCHING* i *LNG: SEARCHING*. Ukoliko *LATLNG_SEARCHING_STATE* ima vrijednost *false*, na LCD panelu će se ispisivati trenutna lokacija:

```
void show_lat_lng()
{
	u8g2.setCursor(1, 31);
	u8g2.print(F("LAT:"));
	u8g2.setCursor(26, 31);

	if (!gps.location.isValid())
	{
		u8g2.print(F("SEARCHING"));
	}

	else
	{
		LATLNG_SEARCHING_STATE = false;

		u8g2.print(gps.location.lat(), 5);
	}

	u8g2.setCursor(1, 41);
	u8g2.print(F("LNG:"));
	u8g2.setCursor(26, 41);

	if (!gps.location.isValid())
	{
		u8g2.print(F("SEARCHING"));
	}

	else
	{
		LATLNG_SEARCHING_STATE = false;

		u8g2.print(gps.location.lng(), 5);
	}
}
```

5. ZAKLJUČAK

Ubrzanim razvojem tehnologije, pojavljuje se kako želja tako i potreba digitalizacije uređaja koje su se kroz povijest koristile. Kompas je jedna od njih. Arduino razvojne pločice jedan su od najzastupljenijih alata zbog jednostavnosti učenja te dostupnosti i obuhvatnosti brojne dokumentacije koja omogućuje relativno jednostavnu realizaciju ideja. Uz prihvatljivu cijenu, programiranje rada mikrokontrolera se vrši u programskom sučelju Arduino IDE koji je besplatan. Sam rad je osmišljen tako da ima dva načina rada. Prvi način rada se koristi u slučaju da je korisniku potrebna orijentacija u prostoru, te je zadaća uređaja da radi kao klasični kompas. Drugi način rada je taj da korisnik unese geografske koordinate svoga cilja, a zadatak uređaja je da pokaže smjer prema izračuna udaljenost od cilja. Sama prednost ovog uređaja je lako izmjenjivanje načina rada te jednostavnost korištenja.

6. LITERATURA

[1] Link s interneta, https://www.arduino.cc/, pristupljeno 16.8.2022.

[2] Link s interneta, https://en.wikipedia.org/wiki/Arduino, pristupljeno 16.8.2022.

[3] Link s interneta, https://www.arduino.cc/en/software, pristupljeno 17.8.2022.

[4] Link s interneta, https://blog.robertelder.org/gy-511-lsm303dlhc/ , pristupljeno 19.8.2022.

[5] Link s interneta, https://iqbalasyadad.medium.com/lets-work-with-lsm303dlhc-gy-511-1e3615130346 , pristupljeno 20.8.2022.

[6] Link s interneta, http://www.datasheet.es/PDF/866235/NEO-6M-pdf.html , pristupljeno 20.8.2022.

[7] Link s interneta, https://www.crystalfontz.com/controllers/Sitronix/ST7920/, 21.8.2022.

[8] Link s interneta, https://e-radionica.com/en/graphical-lcd-display-128x64-blue.html, pristupljeno 23.8.2022.

[9] Link s interneta, https://en.wikipedia.org/wiki/Arduino, pristupljeno 23.8.2022.

[10] Link s interneta, https://stonekick.com/blog/using-basic-trigonometry-to-measure-distance.html, pristupljeno 26.8.2022.

7. PRILOG

7.1. Popis slika

7.2. Popis tablica

7.3. Programski kod

```
#include <Wire.h>
#include <LSM303.h>
#include <TinyGPS++.h>
#include <U8g2lib.h>
#include <SoftwareSerial.h>

const float TARGET_LATITUDE         = 45.32698854772152;
const float TARGET_LONGITUDE        = 14.443893690192274;
// const float TARGET_LATITUDE         = 0;
// const float TARGET_LONGITUDE        = 0;

const uint8_t SCREEN_WIDTH          = 128;
const uint8_t SCREEN_HEIGHT         = 64;
const uint8_t COMPASS_RADIUS        = 20;
const uint8_t TX_PIN                = 3,
              RX_PIN                = 4;
const uint16_t GPS_BAUD             = 9600;
const uint8_t SMART_GLOBAL_DELAY    = 300;
const float LATLNG_SEARCH_FIX       = 1000.00;
const float MAGNETIC_DECLINATION    = 4.2; // Rijeka magnetic declination

uint8_t COMPASS_CENTER_X            = SCREEN_WIDTH - 25;
uint8_t COMPASS_CENTER_Y            = SCREEN_HEIGHT / 2;

bool LATLNG_SEARCHING_STATE         = true;
bool IS_TARGET_ZERO                 = false;
bool IS_TARGET_INVALID              = true;

uint8_t directionX, directionY;

float latitude, longitude;

LSM303 compass;
U8G2_ST7920_128X64_2_SW_SPI u8g2(U8G2_R0, 13, 11, 10);
TinyGPSPlus gps;
SoftwareSerial gpsSerial(TX_PIN, RX_PIN);


void setup(void)
{
  u8g2.begin();
  u8g2.setFont(u8g2_font_6x10_tf);

  Serial.begin(9600);
  gpsSerial.begin(GPS_BAUD);

  Wire.begin();
  compass.init();
  compass.enableDefault();

  Serial.println(F("Setup complete..."));

  compass.m_min = (LSM303::vector<int16_t>){-527,   -548,   -394}; // Replace
with MIN values from calibration
  compass.m_max = (LSM303::vector<int16_t>){+225,    +81,    -63}; // Replace
with MAX values from calibration

  if (TARGET_LATITUDE == 0.00 && TARGET_LONGITUDE == 0.00)
  {
    IS_TARGET_ZERO = true;
```

```
    COMPASS_CENTER_X = SCREEN_WIDTH / 2;
    COMPASS_CENTER_Y = SCREEN_HEIGHT / 2;
  }

  if ((TARGET_LATITUDE < 90 && TARGET_LATITUDE > -90) && (TARGET_LONGITUDE <
180 && TARGET_LONGITUDE > -180))
  {
    IS_TARGET_INVALID = false;
  }
}


void show_compass_heading(uint16_t heading, int cd)
{
  if (!IS_TARGET_ZERO)
  {
    u8g2.setCursor(1, 8);
    u8g2.print(F("CRSDGRS: "));
  }

  u8g2.setCursor(56, 9);

  IS_TARGET_ZERO ? u8g2.print(heading) : u8g2.print(cd);

  u8g2.drawUTF8(73, 9, "°");
}


void compass_circle()
{
  if (IS_TARGET_ZERO)
  {
    u8g2.drawCircle(SCREEN_WIDTH / 2, SCREEN_HEIGHT / 2, COMPASS_RADIUS,
U8G2_DRAW_ALL);
  }

  else
  {
    u8g2.drawCircle(SCREEN_WIDTH - 25, SCREEN_HEIGHT / 2, COMPASS_RADIUS,
U8G2_DRAW_ALL);
  }
}


void compass_notches()
{
  int8_t dxo, dyo, dxi, dyi;

  for (float i = 0; i < 360; i = i + 22.5)
  {
    dxo = COMPASS_RADIUS * cos(i * PI / 180);
    dyo = COMPASS_RADIUS * sin(i * PI / 180);
    dxi = dxo * 0.95;
    dyi = dyo * 0.95;
    u8g2.drawLine(dxi + COMPASS_CENTER_X, dyi + COMPASS_CENTER_Y, dxo +
COMPASS_CENTER_X, dyo + COMPASS_CENTER_Y);
  }
}


void compass_letter(uint8_t x, uint8_t y, char dir[10])
{
```

```
  u8g2.setCursor(x, y);
  u8g2.print(dir);
}


void compass_direction_letters(double cd)
{
  if (!IS_TARGET_ZERO)
  {
    auto cardinal_course_degrees = TinyGPSPlus::cardinal(gps.course.deg());

    if (strlen(cardinal_course_degrees) == 1)
    {
      compass_letter((COMPASS_CENTER_X - 2),(COMPASS_CENTER_Y - 9),
cardinal_course_degrees);
    }

    if (strlen(cardinal_course_degrees) == 2)
    {
      compass_letter((COMPASS_CENTER_X - 5),(COMPASS_CENTER_Y - 9),
cardinal_course_degrees);
    }

    if (strlen(cardinal_course_degrees) == 3)
    {
      compass_letter((COMPASS_CENTER_X - 8),(COMPASS_CENTER_Y - 9),
cardinal_course_degrees);
    }
  }

  else
  {
    compass_letter((COMPASS_CENTER_X - 2),(COMPASS_CENTER_Y - 9), "N");
    compass_letter((COMPASS_CENTER_X - 2),(COMPASS_CENTER_Y + 17), "S");
    compass_letter((COMPASS_CENTER_X + 13),(COMPASS_CENTER_Y + 4), "E");
    compass_letter((COMPASS_CENTER_X - 16),(COMPASS_CENTER_Y + 4), "W");
  }
}


void compass_arrow(int8_t x2, int8_t y2, int8_t x1, int8_t y1, int8_t
arrowLength, int8_t arrowWidth)
{
  float distance;

  int8_t dx, dy, x2o, y2o, x3, y3, x4, y4, coefficientDirection;

  distance = sqrt(pow((x1 - x2), 2) + pow((y1 - y2), 2));

  dx = x2 + (x1 - x2) * arrowLength / distance;
  dy = y2 + (y1 - y2) * arrowLength / distance;

  coefficientDirection = arrowWidth / arrowLength;

  x2o = x2 - dx;
  y2o = dy - y2;

  x3 = y2o * coefficientDirection + dx;
  y3 = x2o * coefficientDirection + dy;

  x4 = dx - y2o * coefficientDirection;
  y4 = dy - x2o * coefficientDirection;
```

```
  u8g2.drawLine(x1, y1, x2, y2);
  u8g2.drawLine(x1, y1, dx, dy);
  u8g2.drawLine(x3, y3, x4, y4);
  u8g2.drawLine(x3, y3, x2, y2);
  u8g2.drawLine(x2, y2, x4, y4);
}


void draw_compass(uint16_t heading, double cd)
{
  compass_circle();
  compass_notches();
  compass_direction_letters(cd);

  if (IS_TARGET_ZERO)
  {
    directionX = (0.7 * COMPASS_RADIUS * cos((heading - 90) * PI / 180)) +
COMPASS_CENTER_X;
    directionY = (0.7 * COMPASS_RADIUS * sin((heading - 90) * PI / 180)) +
COMPASS_CENTER_Y;
  }

  else
  {
    directionX = (0.7 * COMPASS_RADIUS * cos(((cd + MAGNETIC_DECLINATION) -
90) * PI / 180)) + COMPASS_CENTER_X;
    directionY = (0.7 * COMPASS_RADIUS * sin(((cd + MAGNETIC_DECLINATION) -
90) * PI / 180)) + COMPASS_CENTER_Y;
  }

  compass_arrow(directionX, directionY, COMPASS_CENTER_X, COMPASS_CENTER_Y,
2, 2);
}


void show_lat_lng()
{
  u8g2.setCursor(1, 31);
  u8g2.print(F("LAT:"));
  u8g2.setCursor(26, 31);

  if (!gps.location.isValid())
  {
    u8g2.print(F("SEARCHING"));
  }

  else
  {
    LATLNG_SEARCHING_STATE = false;

    u8g2.print(gps.location.lat(), 5);
  }

  u8g2.setCursor(1, 41);
  u8g2.print(F("LNG:"));
  u8g2.setCursor(26, 41);

  if (!gps.location.isValid())
  {
    u8g2.print(F("SEARCHING"));
  }

  else
```

```
  {
    LATLNG_SEARCHING_STATE = false;

    u8g2.print(gps.location.lng(), 5);
  }
}


void show_distance_from_target(unsigned long distance)
{
  u8g2.setCursor(1, 63);
  u8g2.print(F("Distance: "));

  if (!IS_TARGET_INVALID)
  {
    if (LATLNG_SEARCHING_STATE)
    {
      u8g2.setCursor(58, 63);
      u8g2.print(F("CALCULATING"));
    }

    else
    {
      u8g2.setCursor(57, 63);
      u8g2.print(distance);

      u8g2.setCursor(100, 63);
      u8g2.print(F("m"));
    }
  }

  else
  {
    Serial.println(F("Wrong target coordinates. Make sure ( -90 > LAT < 90 )
and ( -180 > LNG < 180 )!!!"));
    u8g2.setCursor(57, 63);
    u8g2.print(F("WRGTRGT"));
  }
}


void U8G2_draw(unsigned long d, double cd)
{
  uint16_t heading = compass.heading() + MAGNETIC_DECLINATION;

  show_compass_heading(heading, cd);
  draw_compass(heading, cd);

  if (!IS_TARGET_ZERO)
  {
    show_lat_lng();
    show_distance_from_target(d);
  }
}


static void smart_delay(unsigned long ms)
{
  unsigned long start = millis();

  do
  {
    while (gpsSerial.available())
```

```
    {
      gps.encode(gpsSerial.read());
    }
  } while (millis() - start < ms);
}


void loop(void)
{
  compass.read();

  if (!IS_TARGET_ZERO)
  {
    smart_delay(SMART_GLOBAL_DELAY);
  }

  unsigned long distance = (unsigned long)TinyGPSPlus::distanceBetween(
    gps.location.lat(),
    gps.location.lng(),
    TARGET_LATITUDE,
    TARGET_LONGITUDE
  );

  double course_to_target = TinyGPSPlus::courseTo(
    gps.location.lat(),
    gps.location.lng(),
    TARGET_LATITUDE,
    TARGET_LONGITUDE
  );

  double course_degrees = gps.course.isValid() ? gps.course.deg() : 0;

  u8g2.firstPage();
  do
  {
    U8G2_draw(distance, course_degrees);
  } while ( u8g2.nextPage() );
}
```

8. SAŽETAK I KLJUČNE RIJEČI

Kroz ovaj završni rad realizirana je ideja izrade električnog kompasa s GPS modulom na Arduino platformi. Opisane su komponente korištene u izradi završnog rada, njegove karakteristike te njihova uloga u cijelom sklopu. Nadalje, detaljno je objašnjen postupak iscrtavanja oblika i slova na LCD panelu, te kako su ukomponirani magnetometar i GPS modul u jednu cjelinu. Navedeni su problemi koji su se pojavili prilikom izrade rada i kako su otklonjeni.

Ključne riječi: Arduino UNO, Arduino IDE, magnetometar, GPS modul, LCD zaslon

9. SUMMARY AND KEY WORDS

Through this bachelors thesis, the electronic compass with a GPS module based on the Arduino platform has been realised. All used components have been described, their characteristics, and their overall role in the complete circuit. Furthermore, a detailed overview of the project has been provided explaning how the shapes and letters were shown on a LCD screen, and how the magnetometer and the GPS module were brought together into one whole single working circuit. All the problems that occurred during the circuit implementation were described along with the implemented solutions.

Key words: Arduino UNO, Arduino IDE, magnetometer, GPS module, LCD display