Merilec nadmorske višine z uporabo WEMOS D1 mini

Erik Rakušček
Seminarska naloga, Brezžična senzorska omrežja
Fakulteta za računalništvo in informatiko
Univerza v Ljubljani

May 17, 2020

1 Uvod

Internet stvari (IoT) je že vrsto let v velikem porastu. Razširjen je v veliko različnih kategorijah kot so npr. pametne hiše, avtomobilizem, zdravstvo, pametna mesta, pametne ure... Omogoča nam poenostavitev in avtomatizacijo različnih opravil. Med taka opravila sodijo tudi spremljanje okoljskih podatkov kot so npr. temperatura, zračni tlak, vlažnost in še več z uporabo različnih senzorjev.

V okviru te seminarske naloge smo se odločili ustvariti spletno aplikacijo za spremljanje spreminjanja nadmorske višine. Najbolj očiten primer uporabe spletne aplikacije je v pohodništvu. Ob odpravi na sprehod ali izlet v hribe bi uporabnik s seboj vzel našo napravo, ki bi tekom pohoda spremljala spreminjanje nadmorske višine uporabnika. Tako bi lahko uporabnik po končanem izletu lahko preveril kakšno nadmorsko višino je premagal in kdaj ter kako hitro je premagoval najstrmejše klance. Te podatke bi si lahko tudi ogledal v realnem času z uporabo naše spletne aplikacije.

Za potrebe naše spletne aplikacije smo uporabili senzorje na WEMOS D1 mini mikrokrmilniku. Nadmorsko višino lahko merimo glede na zračni tlak zato smo na omenjeni ploščici uporabili modul BMP280, ki nam omogoča merjenje tovrstnih vremenskih podatkov. Podatke o zračnem tlaku smo se odločili posredovati do spletne aplikacije z uporabo standarda MQTT. Zato

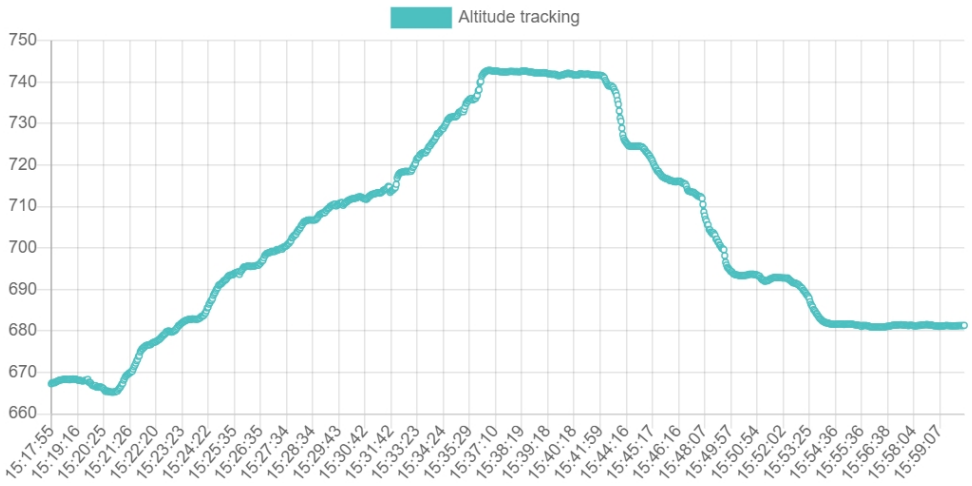

Figure 1: Spremljanje premagane nadmorske višine pri kratkem sprehodu v naravi

smo se morali s ploščico najprej povezati v omrežje. To smo storili tako, da smo na osebnem telefonu vklopili vročo točko (angl. *hotspot*) in se nanjo povezali s ploščico. Za prenos podatkov smo uporabili MQTT posrednika CloudMQTT [2]. Logiko smo na ploščici implementirali z uporabo programskega jezika C v okolju FreeRTOS [3]. Na drugi strani smo v okolju node.js ustvarili osnovno spletno aplikacijo za spremljanje podatkov poslanih s ploščice. Z njo smo se povezali na CloudMQTT in brali podatke o zračnem tlaku, nato pa jih uporabili za generiranje grafa spreminjanja nadmorske višine. Spletno aplikacijo smo tudi objavili na spletu.

Našo aplikacijo smo preizkusili tudi na terenu in dobili zelo zanimive rezultate, ki pa so pokazali tudi nekaj hib takšne aplikacije. Ker se zračni tlak ves čas spreminja je velik izziv ustvariti do centimetra natančno meritev nadmorske višine. Do boljših rezultatov smo prišli, ko smo v izračun nadmorske višine iz zračnega tlaka dodali tudi podatke o zračnem tlaku iz bližnjih vremenskih postaj. Aplikacijo smo preizkusili tudi kot merilec relativne višin. Analizirali smo kako natančno bo senzor BMP280 izmeril višino in to primerjali z dejanskimi podatki. Kljub relativno dobrim rezultatom smo ugotovili da je spletna aplikacija bolj uporabna kot zabaven pripomoček kot pa izredno

natančen merilec.

2 Metode

V tem poglavju bomo opisali metode in pristope k implementaciji naše IoT rešitve.

2.1 Senzor zračnega tlaka

Ker je osnovna naloga našega projekta pošiljanje podatkov o zračnem tlaku do uporabnika smo morali najprej implementirati povezovanje mikrokrmilnika na splet. To smo storili tako, da smo v datoteki *sensor.c* dodali nekaj vrstic kode za avtomatsko povezovanje z WiFi-jem.

```
struct sdk_station_config config = {
    .ssid = WIFI_SSID,
    .password = WIFI_PASS,
};
sdk_wifi_station_set_auto_connect( 1 );
sdk_wifi_set_opmode( STATION_MODE );
sdk_wifi_station_set_config( &config );
sdk_wifi_station_connect();
```

Tukaj smo uporabili modul nRF24L01, ki je že vgrajen v ploščico D1 mini in lahko obratuje z zelo nizko porabo energije.

2.2 Senzor zračnega tlaka

Zračni tlak smo merili z uporabo modula BMP280. V metodi *bmp_init()* smo najprej vklopili vodilo I2C, saj je preko tega vodila senzor povezan z mikrokrmilnikom. To storimo tako da v *Makefile* dodamo knjižnico tega vodila:

```
EXTRA_COMPONENTS = /esp-open-rtos/extras/i2c
```

in ob inicializaciji modula BMP280 v datoteki *bmp.c* dodamo:

```
i2c_init(BUS_I2C, SCL, SDA, I2C_FREQ_100K);
gpio_enable(SCL, GPIO_OUTPUT)
```

Nato po zgledu primera iz FreeRTOS nastavimo BMP280. Podatke o zračnem tlaku zajemamo tako da ustvarimo zanko, kjer vsaki 2 sekundi zajamemo nove podatke in jih shranimo v globalno spremenljivko *pressure*, ki je dostopna tudi v datoteki *mqtt_client*. Meritev bi lahko izvedli tudi manj pogosto in s tem privarčevali pri energiji. Ker pa baterija ni bila problem pri našem testiranju in smo želeli ažurne podatke smo se odločili za dvo-sekundne intervale meritev.

2.3 MQTT in CloudMQTT

Ko preberemo podatke o zračnem tlaku, jih prenesemo do naše spletne aplikacije. Tukaj smo se odločili uporabiti protokol MQTT ali Message Queuing Telemetry Transport. To je protokol za pošiljanje podatkov uporabnikom brez shranjevanja le-teh. Protokol je primarno namenjen ravno storitvam v svetu IoT saj omogoča izredno varčen način pošiljanja sporočil v obliki objavi/naroči (angl. *publish/subscribe*). Pošiljatelj objavlja podatke na neki temi, uporabnik pa se na to temo naroči in tako lahko prebere nove podatke takoj ko so ti na voljo.

Implementirati smo poskušali pošiljanje preko več različnih spletnih posrednikov MQTT (angl. *online MQTT brokers*). Najprej smo poskušali pošiljanje podatkov implementirati preko *test.mosquitto.org* [6], vendar nam ni uspelo vzpostaviti varne povezave z njihovimi strežniki. Imeli smo težave s povezovanjem preko protokola TLS. Nato smo skušali vzpostaviti povezavo še preko *broker.hivemq.com* [4], vendar smo hitro ugotovili, da se preko tega posrednika ne da vzpostaviti varne povezave ampak so vsi podatki, ki jih objavimo z uporabo njihovega brezplačnega programa javno dostopni vsem.

Na koncu smo se odločili za posrednika CloudMQTT. Omogoča zelo enostavno nastavitev lastnega uporabniškega imena in gesla za pošiljanje podatkov preko njihovega MQTT posrednika. Na njihovi spletni strani lahko definiramo novo temo (angl. *topic*), na katero pošiljamo sporočila, v našem primeru so to podatki o zračnem tlaku.

Pošiljanje podatkov preko MQTT je implementirano v datoteki *mqtt.c*. Z uporabo metode *xTaskCreate()* inicializiramo periodično pošiljanje podatkov preko MQTT v dvo-sekundnih intervalih. Podatke shranjujemo v vrsto (angl. *queue*), in nato enega za drugim zapisujemo na temo *"data"* v CloudMQTT.

```
ret = mqtt_publish(&client, "/data", &message);
```

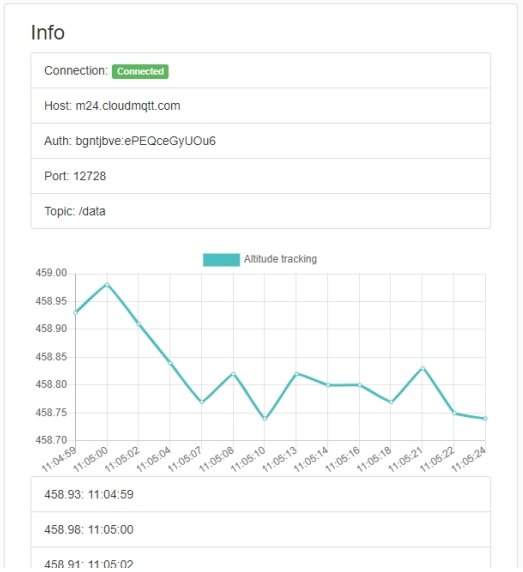

Figure 2: Spletna aplikacija

2.4 Spletna aplikacija

Za vizualizacijo podatkov smo ustvarili spletno aplikacijo v okolju *node.js* z uporabo okolja *expressjs*. Spletno aplikacijo smo tudi objavili z uporabo Heroku na naslovu *https://altitudetracker.herokuapp.com/*. Spletna aplikacija se ob odprtju naroči na našo temo pri posredniku CloudMQTT preko URL-ja:

```
mqtt://bgntjbve:ePEQceGyUOu6@m24.cloudmqtt.com:12728,

kjer je URL sestavljen kot:
[protokol]://[uporabnik]:[geslo]@[posrednik]:[vrata]
```

Za delo z MQTT v okolju *node.js* smo uporabili odprto-kodno knjižnico *mqtt*. Naročimo se na temo *"data"* z uporabo metode *client.subscribe()*, ki se samodejno izvede vsakič ko je na voljo nov podatek pri ponudniku.

Nadmorsko višino smo iz zračnega tlaka izračunali po barometrični formuli 1.

$$altitude = \frac{1 - \sqrt[5.25588]{\frac{pressure}{sea\_level\_pressure}}}{2.25577 * 10^{-}5}, \tag{1}$$

kjer je *pressure* zračni pritisk izmerjen s senzorjem BMP280 na našem mikrokrmilniku, *sea_level_pressure* pa referenčni zračni tlak na višini morske gladine.

Najprej smo *sea_level_pressure* določili kar konstantno glede na standardni referenčni zračni tlak - 101325 Pa. Kmalu pa smo ugotovili, da smo s tem dobili zelo slab približek dejanski nadmorski višini. Spletna aplikacija nam je namreč kazala nadmorsko višino z napako okoli 30m. Zato smo se odločili izboljšati natančnost računanja nadmorske višine z uporabo podatkov iz najbližje vremenske postaje. Podatke smo pridobili z uporabo brezplačnega OpenWeatherMap API [5]. Vrednost *sea_level_pressure* smo nastavili na referenčno vrednost, ki nam jo vrne prej omenjeni API.

3 Problemi in kako smo jih reševali

Med delom na seminarski nalogi smo se srečali z vrsto problemov zato bomo opisali le največje izmed njih.

3.1 MQTT posredniki

Zelo veliko težav smo imeli z vzpostavitvijo povezave s katerimkoli brezplačnim MQTT posrednikom. Povezavo smo poizkušali vzpostaviti najprej s *test.mosquitto.org* in *broker.hivemq.com* vendar smo po več neuspelih poskusih ugotovili, da se nam ne bo uspelo povezati do teh posrednikov. Izkaže se da je rokovanje z uporabo certifikatov prezahtevno za procesiranje preko MQTT protokola, saj se nam je "task" za MQTT na mikrokrmilniku prenehal odzivati preden bi mu uspelo vzpostaviti povezavo. Na noben način nam ni uspelo usposobiti rokovanja s certifikati, zato smo se odločili za avtentikacijo z uporabniškim imenom in geslom. Med reševanjem teh problemov smo tudi odkrili novega MQTT posrednika *m24.cloudmqtt.com* z bolj dodelanimi in uporabniku prijaznimi navodili kako vzpostaviti posredovanje sporočil preko MQTT.

3.2 Nihanje zračnega tlaka

Ob merjenju nadmorske višine smo opazili, da se zračni tlak zelo pogosto spreminja. Zaradi tega smo dobili zelo čudne meritve. Čeprav smo imeli

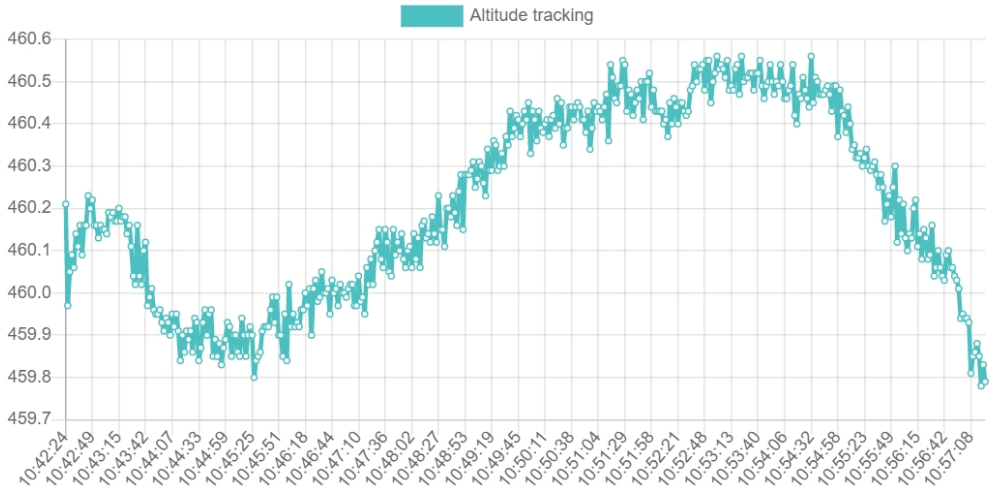

Figure 3: Primer meritev z napako $\pm 0.5m$ ko mikrokrmilnik miruje

mikrokrmilnik ves čas položenega ob sebi na isti nadmorski višini pa so meritve zračnega pritiska nihale in smo posledično v zelo kratkem času naračunali zelo različno nadmorsko višino. Napaka je bila velikosti $\pm 3m$. To napako smo zmanjšali na $\pm 1.5m$ z uvedbo podatkov iz vremenskih postaj opisanih v prejšnjem poglavju.

Ker še vedno nismo bili zadovoljni z rezultati, smo uporabili še "Moving average" filter in povprečili zadnji 5 prebranih vrednosti nadmorske višine. Tako smo izničili velik del nestabilnosti meritev in napako zmanjšali na $\pm 0.5m$. Tako smo sicer zakasnili meritve in žrtvovali odzivnost aplikacije za bolj natančne podatke.

Med testiranjem naše aplikacije smo ugotovili še eno pomanjkljivost podatkov iz vremenskih postaj. Ker nam uporabljeni API v brezplačni verziji vrača zračni tlak le v kPa, pride do pojava velikega padanja in naraščanja meritev nadmorske višine. To se zgodi, ko se referenčni zračni tlak na morski gladini giba okoli določene vrednosti npr. na intervalu od 1020.95 do 1021.05. OpenWeatherMap API nam v tem primeru zaporedoma vrača vrednosti 1020 in 1021, ki pa zelo vplivajo na izračun nadmorske višine. Žal nam ni uspelo najti bolj natančnega brezplačnega ponudnika vremenskih podatkov.

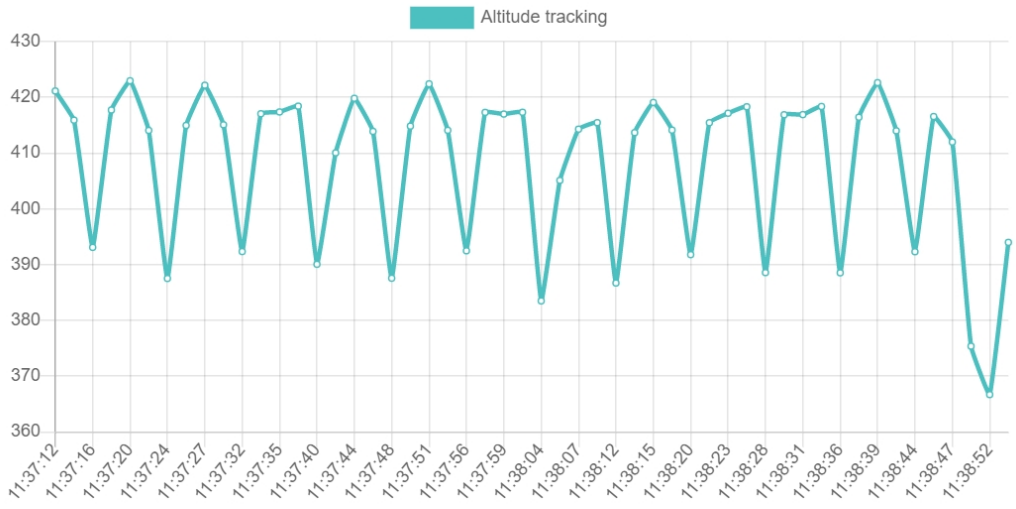

Figure 4: Zračni tlak *sea_level_pressure* se spreminja med 1021 kPa in 1020 kPa

3.3 Razvojno okolje

Poleg zgoraj navedenih problemov smo imeli tudi vrsto problemov z okoljem, ki pa smo jih sproti reševali. Nismo namreč imeli veliko izkušenj s pisanjem in razhroščevanjem v programskem jeziku C.

4 Testiranje

Spletno aplikacijo smo testirali na tri različne načine. Testirali smo uporabnost aplikacije v daljšem obdobju npr. kot pripomoček ki ga vzamemo s seboj na izlet oziroma sprehod za namen spremljanja premagovanja nadmorske višine. Nato smo testirali tudi vpliv vremena na meritve. Nazadnje pa smo testirali še uporabnost aplikacije in mikrokrmilnika kot merilne naprave.

4.1 Analiza delovanja na dolgi rok

Uporabnost spletne aplikacije smo najprej preizkusili tako, da smo merili spremembo nadmorske višine ob vožnji z avtomobilom. Mikrokrmilnik smo

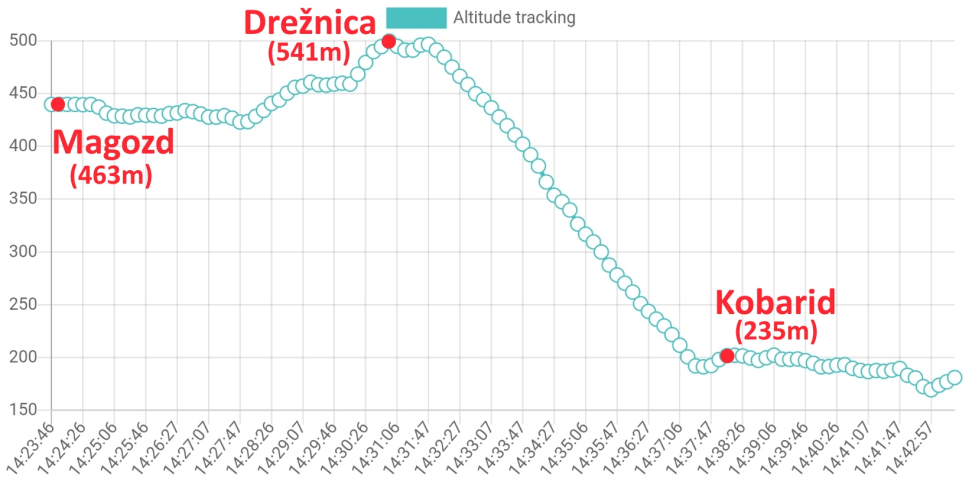

Figure 5: Premagana nadmorska višina in uradni podatki o nadmorski višini

priključili na prenosno baterijo in ga vzeli s seboj na kratko vožnjo z avtomobilom. Spreminjanje nadmorske višine med vožnjo je razvidno na sliki 5. Meritve smo primerjali z uradno določenimi nadmorskimi višinami krajev skozi katere smo se vozili in ugotovili, da smo se dobro približali dejanski nadmorski višini. Upoštevati moramo namreč, da se naselja skozi katera smo se vozili nahajajo na zelo neenakomerno dvignjeni površini in lahko pride do odstopanja tudi zaradi drugega mesta na katerem so bili izmerjeni uradni podatki o nadmorski višini.

4.2 Vpliv vremena

Testiranje smo večinoma opravljali ob sončnem vremenu in to vedno v podobnih pogojih. Glede na to, da naš izračun nadmorske višine delno temelji tudi na podatkih iz vremenskih postaj, nismo pričakovali bistvene razlike ob prihodu slabega oziroma deževnega vremena. Z uporabo podatkov iz najbližje vremenske postaje smo se res izognili večjim nihanjem meritev pod vplivom vremena.

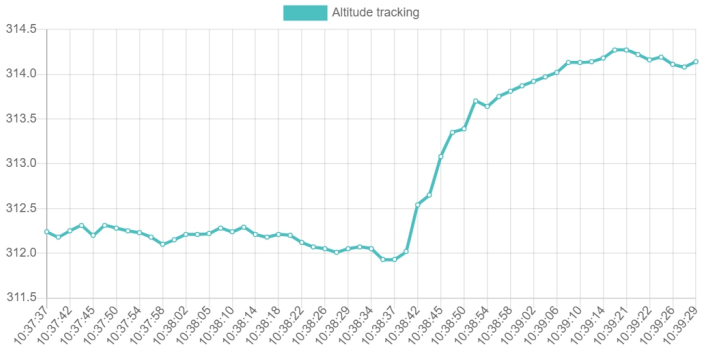

Figure 6: Senzor dvignimo za 2 metra

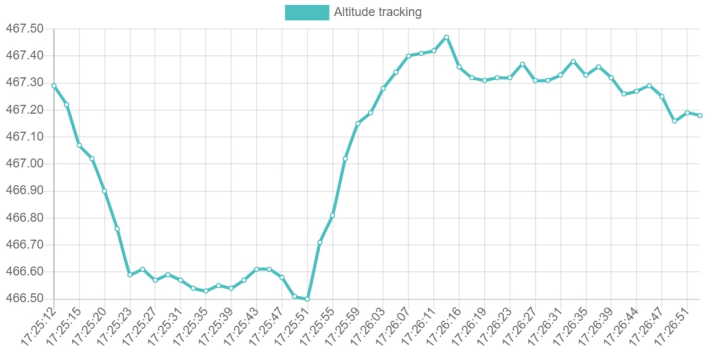

Figure 7: Senzor dvignimo za 1 meter

4.3 Merjenje relativne višine

Zanimala nas je tudi odzivnost in s tem uporabnost aplikacije za merjenje relativnih višin. Odločili smo se izmerjeno višino primerjati z višino izmerjeno z metrom. Senzor zračnega tlaka smo postavili na tla in počakali da se nadmorska višina stabilizira. Nato smo z metrom izmerili 2 metra višine in senzor dvignili na to višino. Rezultati so prikazani na sliki 6. Opazimo lahko, da traja nekaj časa preden se podatki stabilizirajo zaradi "moving average" filtra. Iz slike 6 je razvidno, da smo senzor dvignili iz okoli 312 m nadmorske višine na okoli 314 m nadmorske višine. Napaka meritev je bila približno 20 cm. Tudi ob dvigu senzorja za 1 meter smo dobili napako okoli 20 cm.

5 Zaključek

V okviru seminarske naloge smo izdelali spletno aplikacijo za sledenje spreminjanja nadmorske višine. Na mikrokrmilik WEMOS D1 mini smo naložili program, ki preko protokola MQTT v aplikacijo pošilja podatke o zračnem tlaku. Analizirali smo meritve in jih primerjali z uradnimi podatki o nadmorski višini in ugotovili, da smo se dobro približali le-tem. V nadaljnjem delu bi aplikaciji lahko dodali tudi sledenje geolokacije in tako še izboljšali natančnost meritev. Prav tako bi barometrično formulo lahko še izboljšali in dodali še podatke o temperaturi. Med delom na seminarski nalogi smo se veliko naučili o delu z mikrokrmilniki in o uporabi energetsko varčnega protokola MQTT. Poleg tega smo tudi ugotovili, da je pri izračunu nadmorske višine na podlagi zračnega tlaka potrebno upoštevati še veliko drugih vremenskih dejavnikov. Celoten projekt in programska koda je dostopen v Github repozitoriju [1].

References

[1] Altitude Tracker - Erik Rakušček. Dosegljivo: https://github.com/erikrakuscek/Altitude-Tracker. [Dostopano: 16. 5. 2020].

[2] CloudMQTT. Dosegljivo: https://www.cloudmqtt.com/. [Dostopano: 16. 5. 2020].

[3] esp-open-rtos. Dosegljivo: https://github.com/SuperHouse/esp-open-rtos. [Dostopano: 16. 5. 2020].

[4] HIVEMQ. Dosegljivo: https://www.hivemq.com/. [Dostopano: 16. 5. 2020].

[5] OpenWeatherMap. Dosegljivo: https://openweathermap.org/. [Dostopano: 16. 5. 2020].

[6] R. Light. Mosquitto: server and client implementation of the mqtt protocol. *Journal of Open Source Software*, 2(13):265, 2017.