SVEUČILIŠTE U SPLITU

FAKULTET ELEKTROTEHNIKE, STROJARSTVA I BRODOGRADNJE

IZVJEŠTAJ

BROJAČ AUTOMOBILA

Edi Pešut

Split, srpanj, 2020

Sadržaj

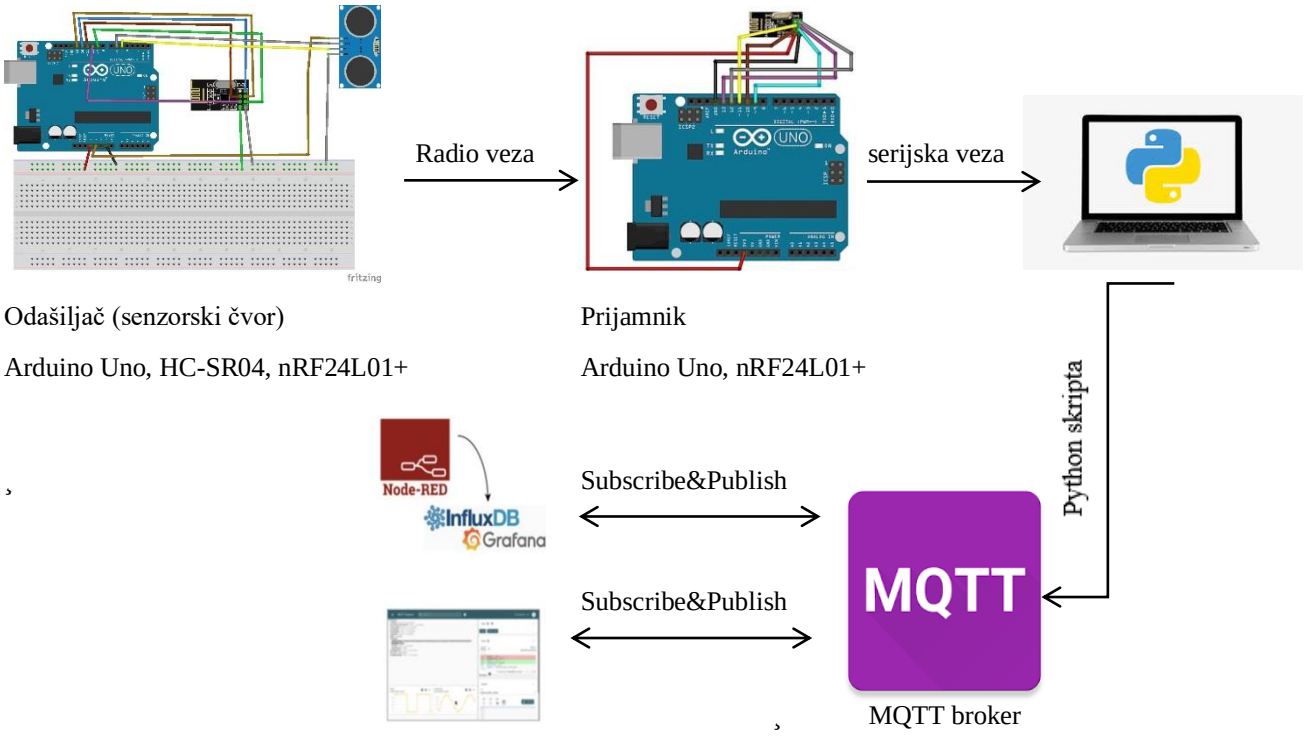
1. UVOD

Prometna mreža veoma je važna sastavnica modernog društva, a možemo je smatrati i krvotokom gospodarstva. Kako bi se osigurao što protočniji promet i bolja prometna mreža, potrebno je kvalitetno projektirati i graditi prometnice. Izgradnja novih, nadogradnja postojećih prometnica, odnosno njihovo projektiranje uvelike će ovisiti o stanju prometa na istima. Najbolji pokazatelj stanja prometa je količina prometa koja se očituje kroz brojanje prometa odnosno vozila na prometnicama. Načini brojanja prometa mogu se podijeliti na dva načina: ručno i automatsko. Ručno brojanje provodi čovjek uz pomoć raznih pomagala i uređaja dok automatsko brojanje provode uređaji i naprave. Postoji više načina automatskog brojanja, poput induktivne petlje, mikrovalnog radara, video image processor-a (VIP), magnetskih i pneumatskih uređaja te ultrazvučnog senzora.

Unutar ovog projekta realiziran je sustav za brojanje vozila koji se sastoji od dva uređaja. Jedan uređaj koristi se kao prijemnik ili bazna stanica te drugog uređaja koji služi kao senzorski čvor i odašiljač. Senzorski čvor realiziran je preko ultrazvučnog senzora pomoću kojeg mjerenjem udaljenosti očitavamo prolazak vozila. Za komunikaciju između ova dva uređaja koristimo radio modul nRF24L01, te preko MQTT brokera i Docker aplikacija, odnosno vizualizacijskog sučelja Grafana prikazujemo rezultate.

2. PREGLED SUSTAVA

Načelna ideja za realizaciju uređaja za brojanje automobila temelji se na mjerenju udaljenosti ultrazvučnim senzorom HC-SR04. Ideja je sljedeća, senzor se postavlja na visinu od 3 do 3.4 metra (prosječna visina semafora), orijentiran prema tlu te ukoliko prođe vozilo očitana udaljenost biti će manja od ~3 metra. Ukoliko je razlika između početne i očitane vrijednosti udaljenosti veća od neke predodređene vrijednosti (stavljamo 1m), preko nRF24L01 modula na prijemnik se šalje podatak o prolasku vozila. Podatak koji će se poslati biti će bool varijabla vrijednosti „true“, odnosno istinito ili logička jedinica. Prijamnik zaprima tu jedinicu te varijabli koju definiramo kao brojač povećava vrijednost za 1. Zatim se formira poruka posebnog formata iz koje se preko python skripte vrijednost brojača šalje na MQTT broker. Upotrebom Docker servisa stanje brojača sa MQTT brokera, preko Node Red, servisa se sprema u InfluxDB bazu podataka. Naposljetku stanje brojača iz InfluxDB baze podataka grafički se prikazuje u Grafani, vizualizacijskom korisničkom sučelju za InfluxDB.



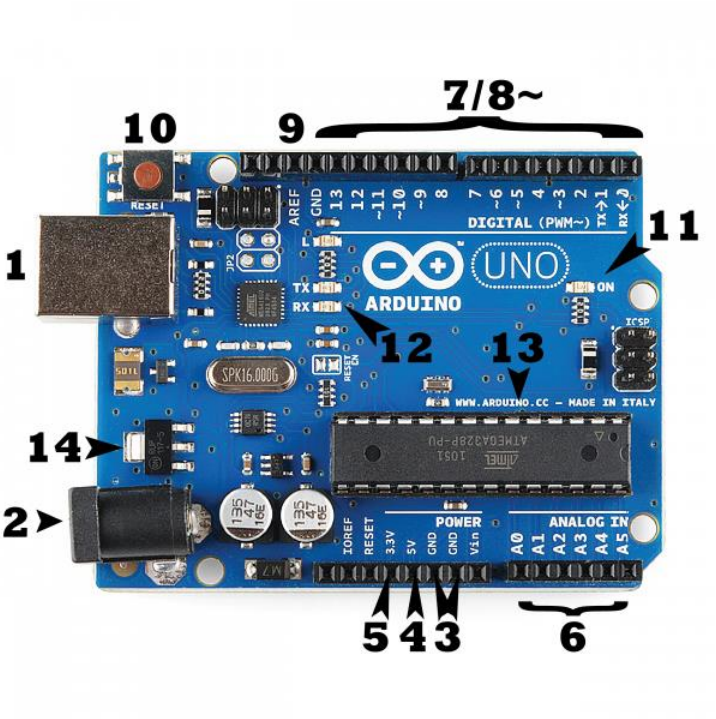
Slika 2.1. Blok dijagram sustava (arhitektura)

3. KORIŠTENA OPREMA

3.1. Arduino Uno

Arduino Uno je open-source mikrokontroler temeljen na Microchipovom ATmega328P mikroprocesoru. Sadrži 14 digitalnih i 6 analognih ulaznih/izlaznih pinova (I/O) na koje je moguće spojiti širok spektar elektroničkih komponenti i senzora. Mikrokontroler je moguće programirati pomoću Arduino IDE programa te se na računalo spaja pomoću USB B kabela. Spajanjem raznih komponenti, poput otpronika, dioda, releja, tranzistora itd. te senzora, odnosno kombinacijom elektroničkih komponenti i senzora i programiranjem mikrokontrolera moguće je kreirati razna upravljačka sučelja ili pak senzorske čvorove.

Slika 3.1. Arduino Uno



Slika 3.2. Arduino Uno sa označenim dijelovima

(1)USB priključak

(2)Priključak za napajanje

(3)GND, odnosno masa

(4) i (5) napajanje 5V i 3.3V

(6) Analogni pinovi (za čitanje i pisanje na ovim pinovima koriste se funkcije analogRead() i analogWrite()).

(7) Digitalni pinovi (za čitanje i pisanje na ovim pinovima koriste se funkcije digitalRead() i digitalWrite()).

(8) digitalni pinovi sa ~ predznakom. PWM pinovi (mogu se koristiti kao digitalni pinovi ili za pulsno širinsku modulaciju)

(9) AREF, analogna referenca

(10) Reset tipka

(11) LED indikator napajanja

(12) TX, RX LED indikatori (TX-transmit, RX-receive)

(13) Integrirani krug (IC)

(14) Regulator napona

3.2. Nordic nRF24L01+

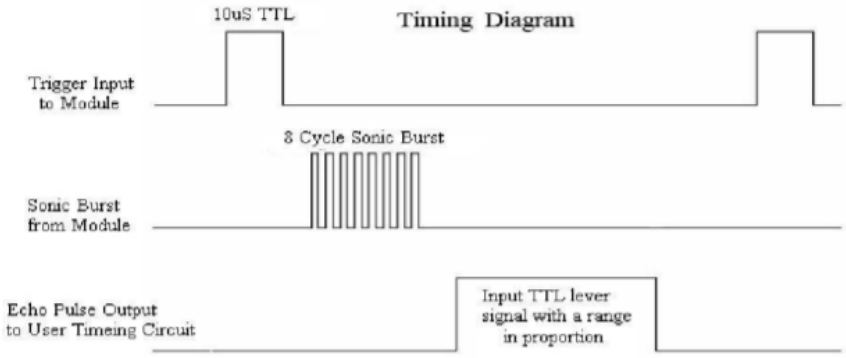
Nordic nRF24L01+ je primopredajnik koji radi u ISM radio pojasu (Industrial, scientific and medical) 2.4 – 2.5 GHz. Širina pojasa je 1 MHz što omogućava komunikaciju na 125 nepreklapajućih kanala (0-124). Ako se koristi kao prijemnik ima mogućnost primanja signala sa 6 kanala uz brzine do 2 Mbps Potrošnja ovog modula veoma je niska, reda nekoliko desetaka mili-ampera. Područje primjene je široko, od bežičnih miševa, tipkovnica, igrački, industrijskih senzora, senzorskih čvorova itd.. Napon napajanja modula je 3-3.6 V, a maksimalna izlazna snaga +20dBm.

Slika 3.3. nRF24L01+ modul sa oznakama pinova

3.3. Ultrazvučni senzor HC-SR04

Ultrazvučni (ultrasonični) senzor udaljenosti HC-SR04 pruža mogućnost beskontaktnog mjerenja udaljenosti u rasponu od 2cm do 400cm uz osjetljivost do 3mm. Senzor se sastoji od ultrazvučnog odašiljača (slično zvučniku), ultrazvučnog prijamnika (slično mikrofonu) te upravljačkog kruga. Princip rada je sljedeći, korištenjem IO triggera proizvodimo signal visoke vrijednosti trajanja najmanje 10 mikro sekundi (TTL pulse), zatim modul automatski šalje „pulse train“ odnosno niz od osam signala frekvencije 40kHz te detektira da li ima povratnog signala. Ukoliko ima povratnog signala, vrijeme trajanja IO signala visoke razine je vrijeme od slanja do povratka ultrazvučnog signala. Udaljenost će biti jednaka umnošku trajanja IO signala i brzine zvuka, podijeljen sa 2. Napon napajanja senzora je 5V DC, a radna struja 15mA. Maksimalan kut pod kojim može očitavati udaljenost je 15°. Također potrebno je koristiti kašnjenje od minimalno 60ms kako ne bi došlo do preklapanja ultrazvučnih signala.

Slika 3.4. Ultrazvučni senzor HC-SR04



Slika 3.5. Vremenski dijagram rada HC-SR04

4. SOFTVER

Za programiranje odašiljača i prijamnika odnosno Arduino Uno mikrokontrolera koristimo PlatformIO ekstenziju unutar Visual Studio Code programa. U slijedećim poglavljima biti će navedeni i pojašnjeni kodovi za odašiljač, prijamnik te python i docker skripta.

4.1. Kod za odašiljač

```
#include <Arduino.h>
#include <SPI.h>
#include <nRF24L01.h>
#include <RF24.h>
// uključivanje potrebnih biblioteka
#define CE_PIN 9
#define CSN_PIN 10 //definiranje pinova nRF24L01+ radio modula
const int trigPin = 3;
const int echoPin = 2; //definiranje pinova HC-SR04 ultrazvučnog senzora

const byte address[6] = {"1node"}; // postavljanje adrese odašiljača
RF24 radio(CE_PIN, CSN_PIN);

struct SensorData
{
    bool info; // definiranje strukture SensorData i varijable info šalj
};
SensorData dataToSend;

unsigned long currentMillis;
unsigned long prevMillis;
unsigned long txIntervalMillis = 50; //postavlja interval slanja podataka
int ldistance = 330; //početna vrijednost udaljenosti

void send();
long read();

void setup()
{
    Serial.begin(115200);

    Serial.println(F("SimpleTx Starting"));
    radio.begin();
    radio.setDataRate(RF24_2MBPS); //postavljanje brzine prijenosa radio veze
    radio.setChannel(112); //postavljanje kanala
    radio.setPALevel(RF24_PA_MAX); //postavljanje snage signala
    radio.setRetries(3, 5);
    radio.openWritingPipe(address);
```

```
    pinMode(trigPin, OUTPUT);
    pinMode(echoPin, INPUT);
}
void loop()
{
    currentMillis = millis();
    if (currentMillis - prevMillis >= txIntervalMillis)
    {
        send(); //ulazi u funkciju send svakih 50 ms
        prevMillis = millis();
    }
}
void send()
{

    long distance = read(); //iščitavamo varijablu distance iz funkcije read
    if (ldistance - distance > 100)//ukoliko je prethodna udaljenost veća od
//očitane za 1 metar
    {
        dataToSend.info = true; //podatak koji se šalje je bool vrijednosti 1
        delay(50);
        while (!(radio.write(&dataToSend, sizeof(dataToSend)))) //dok god ra-
dio.write vraća 0, odnosno while petlja 1 ponavljaj slanje
//unutar while 1 će biti ako je slanje neuspješno
            {
            Serial.println("\nTXfailed");
            }
    }
    ldistance = distance; //postavi prethodnu vrijednost na sadašnju, očitanu
    dataToSend.info = false; //postavi podatak koji se šalje u 0
}
long read()  //funkcija za očitavanje udaljenosti pomoću ultrazvučnog senzora
{
    long duration, distance;
    digitalWrite(trigPin, LOW);
    delayMicroseconds(2);
    digitalWrite(trigPin, HIGH);
    delayMicroseconds(10);
    digitalWrite(trigPin, LOW);
    duration = pulseIn(echoPin, HIGH);
    distance = (duration / 2) / 29.1;
    delay(50);
    return distance;
}
```

4.2. Kod za prijamnik

```
#include <Arduino.h>
#include <SPI.h>
#include <nRF24L01.h>
#include <RF24.h>
//inicijaliziranje potrebnih biblioteka
#define CE_PIN 9
#define CSN_PIN 10
//definiranje pinova za nRF24L01+ radio modul
int brojac = 0; // definiranje i inicijaliziranje brojača
const byte adress[][6] = {"1node", "2node", "3node", "4node", "5node", "6node"
}; // definiranje adresa odašiljača, u ovom projektu koristimo samo jednu

RF24 radio(CE_PIN, CSN_PIN);

struct SensorData
{
  bool info; //definiramo bool varijablu info unutar structure SensorData
};

SensorData dataReceived;

bool newData = false;

void getData();
void showData();

void setup()
{

  Serial.begin(115200);

  Serial.println(F("SimpleRx Starting"));
  radio.begin();
  radio.setDataRate(RF24_2MBPS);
  radio.setChannel(112);
  radio.setPALevel(RF24_PA_MAX);
  radio.openReadingPipe(1, address[0]);
  radio.openReadingPipe(2, address[1]);
  radio.openReadingPipe(3, address[2]);
  radio.openReadingPipe(4, address[3]);
  radio.openReadingPipe(5, address[4]);
  radio.openReadingPipe(6, address[5]);
  radio.startListening();
} //kod za radio modul
```

```
void loop()
{
  getData();
  showData();
}

void getData()
{
  uint8_t pipeNum;
  if (radio.available(&pipeNum))
  {
    radio.read(&dataReceived, sizeof(dataReceived));
    newData = true;
  }
} //u ovoj funkciji dohvaćamo podatke sa radio modula, te varijablu newData
//postavljamo u 1 ukoliko je zaprimljen podatak sa radio komunikacijom
void showData()
{
  if (newData == true) //ako smo zaprimili novi podatak
  {
    brojac++; //povećavamo stanje brojača
    Serial.print("B: "); //naredne tri linije formiraju poseban format poruke
    Serial.print(brojac);//iz kojeg onda preko python skripte šaljemo
    Serial.print("!");//vrijednost brojača na mqtt broker

    newData = false; //vraćamo newData u logičku 0
  }
}
```

4.3. Python i Docker skripte

Python je programski jezik visokog nivo te opće namjene, podržava imperativni, objektno orijentirani i funkcionalni način programiranja. U ovom projektu Python skripta korištena je za prijenos podataka sa prijamnika na MQTT broker.

```
import time
import serial
import paho.mqtt.client as mqtt

BROKER = "mqtt.eclipse.org"           //odabiremo broker
CLIENTID = "MQTTExample"              //odabiremo ID klijenta
TOPIC_ONE = "CarCounter/sensor1"      //odabiremo temu gdje spremamo podatke
COMPORT = "COM11"  #ovisno o port una koji spajamo prijamnik
QOS = 1      //Quality of service


flag_connected = 0


def on_connect(client, userdata, flags, rc):
    logging.debug("Connected result code "+str(rc))
    client.loop_stop()


def on_disconnect(client, userdata, rc):
    logging.debug("DisConnected result code "+str(rc))
    client.loop_stop()


def on_publish(client, userdata, result):  #funkcija za povratnu informaciju
    print("data published \n")
    pass


mqttc = mqtt.Client(CLIENTID)
mqttc.on_connect = on_connect
mqttc.on_disconnect = on_disconnect
mqttc.on_publish = on_publish
mqttc.connect(BROKER)
mqttc.loop_start()

ser = serial.Serial(COMPORT, 115200, timeout=0.1) //timeout određuje brzinu
while True:                                       //slanja podataka
    message = ser.readline()
    print(message)
    if b'B:' in message:       //ovdje očitavamo posebno formatiranu poruku
```

```
        string, brojac = message.split(b' ')
        brojac, rest = brojac.split(b'!')
        print(brojac.decode('utf-8'))
        mqttc.publish(TOPIC_ONE, payload=brojac.decode(
            'utf-8'), qos=QOS, retain=False)
    time.sleep(0.01)
mqttc.disconnect()
time.sleep(0.01)
# python serial_read_mqtt.py  - upisati u terminal za pokretanje Python
skripte
```

Python skripta

```
version: '2'
services:
  influxdb:
    image: influxdb:latest
    ports:
      - '8086:8086'
    volumes:
      - influxdb-storage:/var/lib/influxdb
    environment:
      - INFLUXDB_DB=db0
      - INFLUXDB_ADMIN_USER=${INFLUXDB_USERNAME}
      - INFLUXDB_ADMIN_PASSWORD=${INFLUXDB_PASSWORD}
  chronograf:
    image: chronograf:latest
    ports:
      - '127.0.0.1:8888:8888'
    volumes:
      - chronograf-storage:/var/lib/chronograf
    depends_on:
      - influxdb
    environment:
      - INFLUXDB_URL=http://influxdb:8086
      - INFLUXDB_USERNAME=${INFLUXDB_USERNAME}
      - INFLUXDB_PASSWORD=${INFLUXDB_PASSWORD}
  grafana:
    image: grafana/grafana:latest
    ports:
      - '3000:3000'
    volumes:
      - grafana-storage:/var/lib/grafana
      - ./grafana-provisioning/:/etc/grafana/provisioning
    depends_on:
      - influxdb
    environment:
      - GF_SECURITY_ADMIN_USER=${GRAFANA_USERNAME}
      - GF_SECURITY_ADMIN_PASSWORD=${GRAFANA_PASSWORD}
  node-red:
```

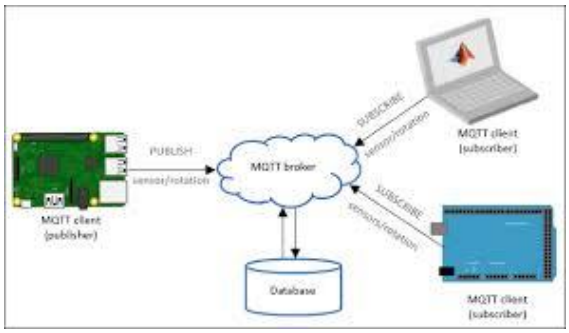
```
    image: nodered/node-red:latest
    environment:
      - TZ=Europe/Amsterdam
    ports:
      - "1880:1880"
    volumes:
      - node-red-data
volumes:
  influxdb-storage:
  chronograf-storage:
  grafana-storage:
  node-red-data:
    #Za kreaciju Docker containera u terminal upisati docker-compose -
f docker-compose.yml -p Projekt up
```

Skripta za kreiranje Docker containera

5. KORIŠTENI SERVISI

5.1. MQTT

MQTT je M2M (machine-to-machine)/IOT (Internet of Things) komunikacijski protokol. Dizajniran je prema principu „publish/subscribe“ na način da se minimalno opterećuje mreža i umanje zahtjevi resursa uređaja. Veoma je koristan za uspostavljanje komunikacije sa udaljenim lokacijama i uređajima na mjestima gdje je jakost mreže ograničena. U ovom projektu korišten je za prikupljanje podataka sa prijamnika te njihovo prosljeđivanje u InfluxDB bazu podataka.

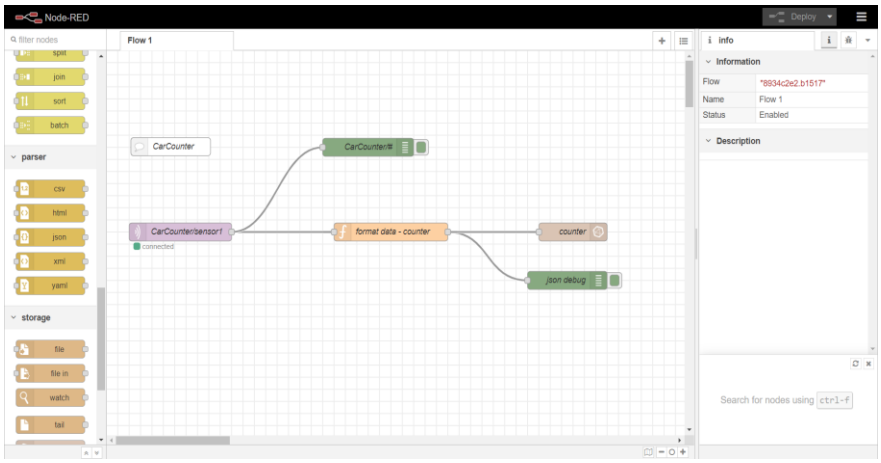


Slika 5.1. Primjer spajanja na MQTT broker

5.2. Docker aplikacija i servisi

Unutar Docker aplikacije, odnosno Docker kontejnera „pakiramo“ cjelokupan kod, biblioteke i komponente o kojima aplikacija ovisi. Docker je alat dizajniran kako bi korisniku olakšao stvaranje, implementaciju i pokretanje aplikacija pomoću kontejnera.

Prvi od Docker servisa koji koristimo je Node RED. Node RED je flow-based razvojni alat za vizualno programiranje sa jednostavnim korisničkim sučeljem.



Slika 5.2. Programiranje unutar Node RED servisa

Za ovaj projekt unutar Node RED servisa šaljemo podatke sa MQTT brokera u InfluxDB bazu podataka. InfluxDB je „time series“ baza podataka. Preko Grafane, vizualizacijskog korisničkog sučelja, podatke iz InfluxDB baze podataka se prikaže grafički. U sljedećem poglavlju prikazani su rezultati testiranja, te neki problemi na koje smo naišli.

6. REZULTATI

Prije samog prikaza rezultata testiranja spomenuti ćemo problem na koji smo naišli prilikom realizacije sustava. Problem se javljao prilikom slanja podataka radio modulom sa odašiljača prema prijamniku. Naime, događalo se to da komunikacija ne bi uspjela otprilike svaki drugi put. Ovaj problem riješen je na način da stavljamo while petlju koja osigurava da će se poruka slati sve dok god je komunikacija neuspješna, odnosno sve dok slanje ne postane uspješno. Ovdje je bilo zanimljivo pratiti broj pokušaja slanja podataka, odnosno broj neuspjelih slanja. Do tog broja došli smo pomoću brojača (varijabla counter), a kod koji smo koristili nalazi se na slici 6.1.. Counter2 varijabla služi nam za praćenje ukupnog broja promjena, odnosno broji koliko puta je objekt prošao ispred senzora. Testiranje je provedeno na način da su odašiljač i prijamnik bili udaljeni 5 metara bez prepreka između, za broj promjena 100. Rezultati testiranja prikazani su u tablici 6.1., a cijelu tablicu moguće je pronaći na sljedećem linku: https://github.com/epesut/izvjestaji_WiSe_2019_20/tree/master/Projekt

```
if (ldistance - distance > 100)
{
    dataToSend.info = true;
    delay(50);
    while (!(radio.write(&dataToSend, sizeof(dataToSend))))
    {
        counter++;
        Serial.println("TXfailed");
    }
    counter2++;
    Serial.print("\n Broj pokušaja slanja :");
    Serial.print(counter);
    Serial.print("\n");
    counter = 0;
    Serial.print("\n ukupan broj promjena :");
    Serial.print(counter2);
    Serial.print("\n");
}
ldistance = distance;
dataToSend.info = false;
```

Slika 6.1. Kod za brojanje neuspjelih pokušaja slanja podataka

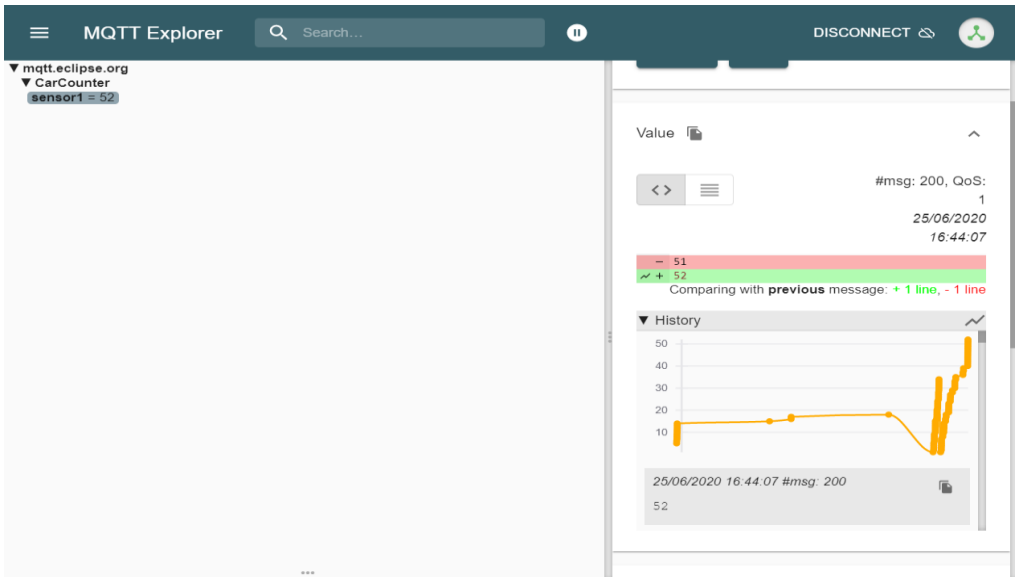
Broj promjena	Ukupno pokušaja	Prosječni broj pokušaja slanja po jednoj promjeni
100	159	1.59

Tablica 6.1.

Nakon testiranja radio komunikacije uslijedilo je testiranje samog sustava. Sustav je testiran na način da je senzor bio postavljen na visinu od otprilike 3 metra, usmjeren prema tlu, te je ispod njega provlačena kartonska maketa. Rezultate testiranja prikazali smo u Grafani, gdje smo iz Influx baze podataka kreirali dva grafa. Na grafu imena „Brojač automobila“, na x-osi prikazuje se broj vozila koja su prošla a na y-osi vrijeme. Na grafu „Brojač automobila“ protok prometa ili prometna „gužva“ biti će ekvivalentna nagibu grafa, veći nagib veći protok automobila u nekom intervalu vremena. Kako bismo dobili intuitivniji prikaz koristimo graf „Pokazatelj gužve“ na kojem će se protok prometa očitavati na način da što je manji razmak između šiljaka veća je gužva odnosno protok prometa.

Slika 6.2. Grafički prikaz u Grafani

Podatak o broju vozila također možemo pratiti i preko MQTT Explorer aplikacije u kojoj to izgleda kao što je prikazano na slici 6.3..



Slika 6.3. MQTT Explorer aplikacija

7. ZAKLJUČAK

Nakon provedenih testiranja dolazimo do zaključka da bi ovakav sustav bio pogodan za okruženja (prometnice) gdje se vozila kreću manjim brzinama, jer primjerice za autoceste gdje je prosječna brzina 130 km/h, automobilu dužine 3 metra potrebno je 83 milisekunde za prolazak ispod senzora. To je veoma blizu delayu od 60 milisekundi koji je potreban za ispravan rad ultrazvučnog senzora. Također dolazimo do zaključka da radio komunikacija preko nRF24L01 modula možda nije najbolji izbor, razlog je poveći broja neuspjelih pokušaja koji također unose određeno kašnjenje u izvođenju programa. Jedan od nedostataka ovakve komunikacije je i smanjen domet. Kako bi se taj problem uklonio bilo bi potrebno postavljati „data forward“ uređaje, što bi naposljetku utjecalo na ekonomsku isplativost sustava. Osim navedenih nedostataka, ovaj sustav ima i niz prednosti poput : niske cijene, jednostavnosti, dobre točnosti za manje brzine, mogućnost montaže na već postojeću infrastrukturu itd.

Jedna zanimljivija primjena i proširenje ovog sustava bila bi implementacija na parkirališta gdje bi praćenjem broja automobila koji su ušli te automobila koji su izišli, mogli očitati broj slobodnih mjesta na parkingu te zatim preko MQTT brokera i Docker servisa taj podatak i grafički prikazati. Osim grafičkog prikaza moguće bi bilo te podatke slati u InfluxDB bazu podataka te onda njih dohvaćati unutar mobilne aplikacije.

LITERATURA

https://en.wikipedia.org/wiki/Arduino_Uno

https://learn.sparkfun.com/tutorials/what-is-an-arduino/whats-on-the-board

https://lastminuteengineers.com/nrf24l01-arduino-wireless-communication/

http://wiki.sunfounder.cc/index.php?title=NRF24L01_Test_with_Arduino

https://cdn.sparkfun.com/datasheets/Sensors/Proximity/HCSR04.pdf

http://mqtt.org/

https://github.com/toperkov/WiSe-2019-20/blob/master/instructions/lab-8.md