УНИВЕРЗИТЕТ У НОВОМ САДУ
ФАКУЛТЕТ ТЕХНИЧКИХ НАУКА
Трг Доситеја Обрадовића 6, 21000 Нови Сад, Република Србија
Деканат: 021 6350-413; 021 450-810; Централа: 021 485 2000
Рачуноводство: 021 458-220; Студентска служба: 021 6350-763
Телефакс: 021 458-133; e-mail: ftndean@uns.ac.rs
ИНТЕГРИСАНИ СИСТЕМ МЕНАЏМЕНТА СЕРТИФИКОВАН ОД:

PROJEKAT IZ M2M ELEKTRONSKIH SISTEMA

NAZIV PROJEKTA:

PID regulacija

TEKST ZADATKA:

Potrebno je projektovati sistem koji će omogućiti PID regulaciju motora sa nelinearnim opterećenjem putem serijske komunikacije i SMS poruka

MENTOR PROJEKTA:

Rajs dr Vladimir

PROJEKAT IZRADILI:

Vladimir Marćiš, E1 83/2021

Milomir Spajić, E1 87/2021

Aleksandar Kiš, E1 91/2021

DATUM ODBRANE PROJEKTA:

23. 04. 2021.

SADRŽAJ

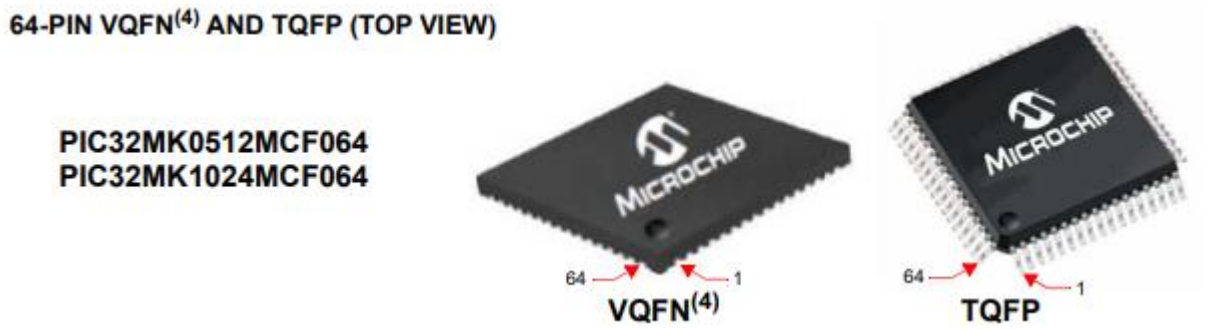
Uvod

U okviru projekta za predmet *M2M elektronski sistemi*, realizovali smo sistem koji predstavlja PID regulaciju motora na čijoj osovini se nalazi nelinearno opterećenje. Izvršili smo testiranje i uporedili signale za linearno i nelinearno opterećenje.

Sa sistemom je moguće komunicirati putem grafičkog korisničkog interfejsa. Moguće je podešavanje brzine motora, koeficijenata PID regulacije kao i uključivanje i isključivanje motora. Drugi način je slanje naredbi putem SMS poruka. Putem SMS poruka, omogućeno je uključivanje i isključivanje motora kao i postavljanje brzine.

Glavne komponente korišćene u realizaciji projekta i njihove karakteristike

Komponenta koja vrši obrađivanje informacija sa enkodera i upravlja motorom je mikrokontroler *PIC32MK1024MCF064*. Na slici 1. prikazan je ovaj mikrokontroler sa svojim priključcima.



Slika 1. Mikrokontroler PIC32MK1024MCF064

Ovaj mikrokontroler ima sve potrebne resurse za realizaciju našeg projekta. Koristili smo priključke za digitalni signal, tajmere, serijsku komunikaciju, impulsno – širinsku modulaciju.

Za napajanje se koristi LiPo baterija sa 4 ćelije i naponom od 14.8V. Ova baterija je prikazana na slici 2.

Slika 2. LiPo baterija sa 4 ćelije

Za komunikaciju SMS porukama, koristili smo Mikroelektronikinu klik pločicu *GSM2 Click*. Ova pločica prikazana je na slici 3.

Slika 3. GSM2 click

Motor koji je korišćen za potrebe testiranja je Maxon prikazan na slici 4.

Slika 4. Maxon DC motor

PID regulacija

Regulisani sistemi

Osnovni zahtevi koji svaki regulisani sistem ili sistem za regulaciju treba da ispuni su:

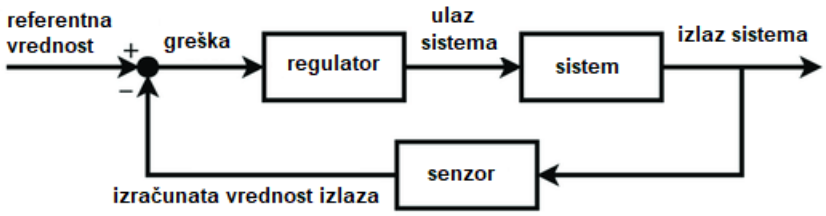
- stabilnost
- tačnost
- velika brzina odziva sistema

Najopštija podela sistema za regulaciju jeste podela u dve velike grupe:

- sistemi za regulaciju bez povratne sprege
- sistemi za regulaciju sa povratnom spregom

Sistemi bez povratne sprege svoj rad baziraju na poznavanju određenog algoritma i glavna karakteristika ovih sistema jeste da njihov rad ne zavisi od trenutne vrednosti izlaznog signala. Ovo može predstavljati problem ukoliko dođe recimo do promene uslova u kojima sistem bez povratne sprege radi.

Sistemi sa povratnom spregom svoj rad zasnivaju na korišćenju povratne sprege. Zadatak povratne sprege jeste da vraća na ulaz sistema izmerene izlazne veličine i na taj način se na ulazu sistema formira greška jer se izmerena izlazna veličina na ulazu u sistem upoređuje sa željenom vrednošću izlaznog signala. Blok dijagram sistema sa povratnom spregom je prikazan na sledećoj slici:



Slika 5. Blok dijagram sistema sa povratnom spregom

Sa blok dijagrama se može videti da se izlazni signal preko određenog senzora vraća na ulaz čitavog regulisanog sistema. Na osnovu nekog referentnog odnosno željenog signala i signala vraćenog sa izlaza sistema se formira signal greške i taj signal greške je zapravo ulazni signal u

određeni regulator. Regulator na osnovu tog signala greške generiše određene kontrolne signale za neki sistem.

Kod PID regulacije (eng. proportional integral derivative), dejstvo regulatora ima tri zavisnosti:

- linearno zavisi od izračunate greške (P - proportional)
- zavisi od integrala te izračunate greške (I - integral)
- zavisi od prvog izoda izračunate greške po vremenu (D - derivative)

Da bi se što bolje objasnilo dejstvo PID regulacije, u nastavku će biti objašnjeno dejstvo svakog dela regulacije posebno.

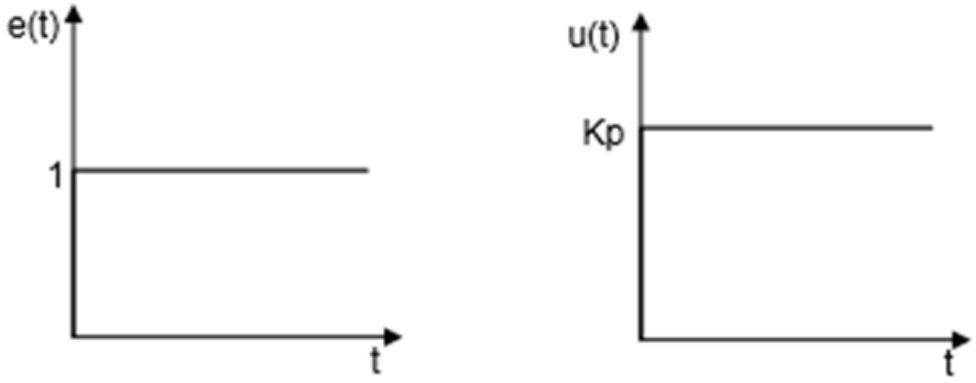
Proporcijalni (P) regulator

Proporcijalni regulator proporcionalno odnosno linearno povezuje grešku regulacije sa upravljačkom promenljivom.

$u(t) = Kp * e(t)$

u(t) - upravljačka promenljiva

e(t) - greška regulacije



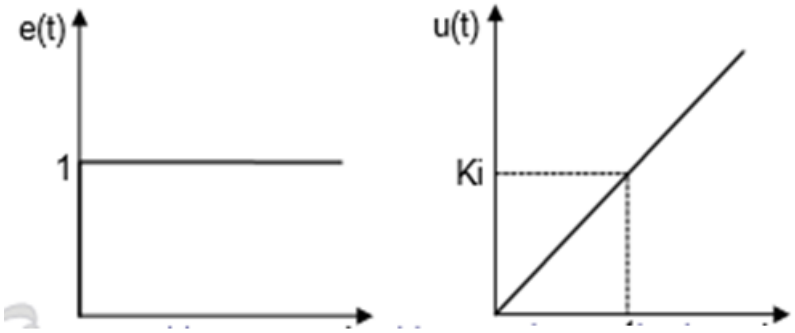
Slika 6. Izgled greške regulacije i upravljačke promenljive kod P regulatora

Integralni (I) regulator

Integralni regulator linearno povezuje brzinu promene upravljačke promenljive sa greškom regulacije.

$$\frac{du(t)}{dt} = Ki * e(t)$$

$$u(t) = Ki * \int_0^t e(t)dt$$



Slika 7. Izgled greške regulacije i upravljačke promenljive kod I regulatora

Diferencijalni (D) regulator

Diferencijalni regulator linearno povezuje brzinu promene greške regulacije sa upravljačkom promenljivom.

$$u(t) = Kd * \frac{de(t)}{dt}$$

PID regulator

Kombinovanjem sva tri prethodno navedena regulatora i dobija se idealni PID regulator.

$$u(t) = Kp * e(t) + Ki * \int_0^t e(t)dt + Kd\frac{de(t)}{dt}$$

Gorenavedena jednačina opisuje ukupno dejstvo jednog PID regulatora. Međutim, jednačina u ovom obliku se ne može upotrebiti prilikom programiranja kontrolera. Odnosno potrebno je nekako transformisati ovu jednačinu u oblik koji mikrokontroler može da "razume". Primenom Ojlerovih formula i diskretizacijom dobijamo sledeću jednačinu:

$$u(kT) = u(kT - T) + Kp[e(kT) - e(kT - T)] + Ki * e(kT) + Kd[e(kT) - 2e(kT - T) + e(kT - 2T)]$$

$u(kT)$ - trenutna upravljačka promenljiva

$u(kT - T)$ - upravljačka promenljiva dobijena u prethodnoj iteraciji

$e(kT)$ - trenutna greška regulacije

$e(kT - T]$ - greška regulacije dobijena u prethodnoj iteraciji

$e(kT - 2T)$ - greška regulacije dobijena pre dve iteracije

Na osnovu ovako dobijene formule se može zaključiti da će se upravljačka promenljiva kod PID regulacije računati ne samo na osnovu trenutne greške regulacije već i na osnovu prethodnih grešaka regulacije.

U nastavku će biti navedeni najbitniji delovi kod vezani za realizaciju PID regulatora.

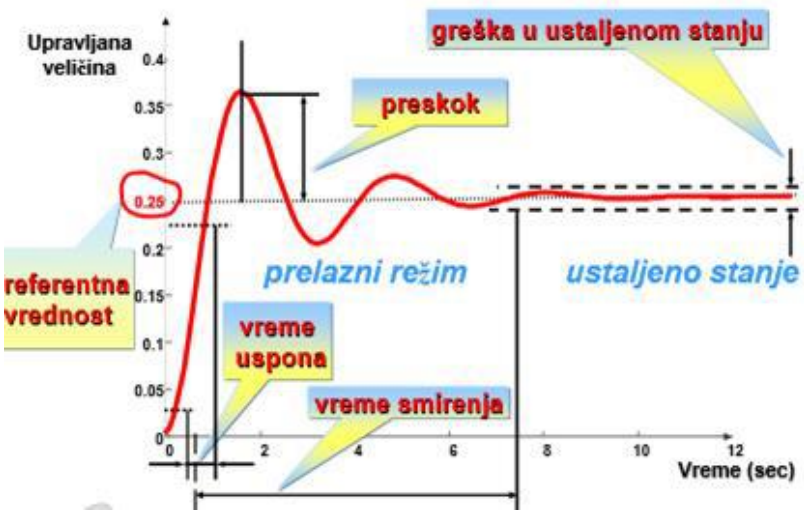
```
uint16_t pid_control(int error, int error_accumulator, int error_derivative){

      static float PID = 0.0;

      PID += (Kp * error + Ki * error_accumulator + Kd * error_derivative) *
6000/500;

      if(PID > PID_MAX)PID = PID_MAX;

      if(PID < PID_MIN)PID = PID_MIN;

      return (uint16_t)PID;

   }
```

Dakle, ova funkcija služi za izračunavanje upravljačke promenljive PID regulatora. Na osnovu koda se može zaključiti da se trenutna upravljačka promenljiva sabira sa prethodnom baš kao što je to opisano u formuli. Takođe, zadati su i određene granice tj. maksimalne i minimalne vrednosti koje upravljačka promenljiva može imati. Deo koda koji služi za izračunavanje trenutne greške regulacije je naveden u nastavku:

```
velocity = (uint16_t)((500.0/7140.0)*(float)encoder_get_velocity());

error_previous = error_velocity;

error_velocity = pid_error(desired_velocity,velocity)

error_accumulator += error_velocity;

error_derivative = error_previous - error_velocity;

pid_output = pid_control(error_velocity, error_accumulator,
error_derivative);

if(state)start_motor(pid_output,direction);
```

Najpre se dobija informacija od enkodera o trenutnoj brzini motora. Na osnovu određene zadate vrednosti za brzinu motora i trenutne brzine motora se izračunava trenutna greška regulacije. Trenutna greška regulacije se smešta u dodatnu promenljivu u kojoj se čuva ta vrednost do sledeće iteracije jer je za izračunavanje upravljačke promeljive bitna vrednost greške regulacije koja je dobijena i u prethodnim iteracijama.

Kako bi se što je moguće bolje objasnio uticaj parametara regulacije odnosno Kp, Ki i Kd parametara, na sledećoj slici će biti prikazan signal sa naznačenim parametrima.



Slika 8. Signal sa svim naznačenim delovima

Dakle, parametri koji su od značaja kod određivanja vrednosti Kp, Ki i Kd jesu vreme uspona i smirenja, preskok i greška u ustaljenom stanju. Idealno bi bilo kada bi svi parametri bili jednaki nuli ali pošto to u realnosti nije moguće, onda se teži ka tome da navedeni parametri

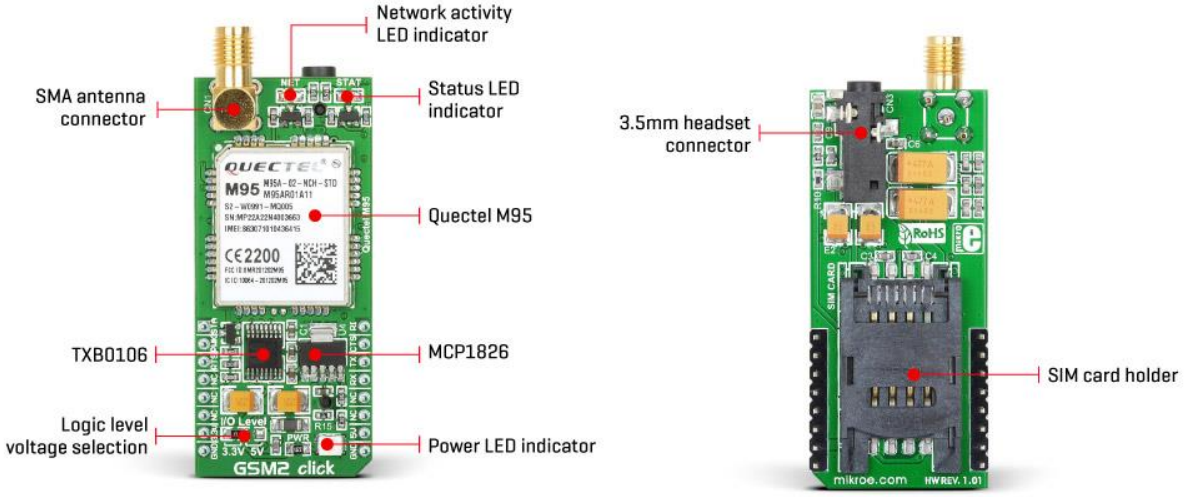
budu što je moguče manji. Uticaj Kp, Ki i Kd na navedene parametre će biti naveden u tabeli ispod.

	Vreme uspona	Preskok	Vreme smirenja	Greška u ustaljenom stanju
Kp	smanjuje	povećava	mala promena	smanjuje
Ki	smanjuje	povećava	povećava	eliminiše
Kd	mala promena	smanjuje	smanjuje	mala promena

Na osnovu tabele koja je gore napisana, može se zaključiti da niti jedan od ova tri parametra nema samo pozitivno dejstvo na svaki bitan parameter nekog signala. Međutim, ukoliko se izabere dobra kombinacija ova tri parametra mogu se dobiti jako dobri, gotovo idealni rezultati odnosno može se dobiti signal koji će imati jako dobre parametre.

GSM Komunikacija

GSM odnosno Globalni sistem za međunarodne komunikacije je najpopularniji standard mobilne telefonije koji služi za prenos glasa i SMS poruka na velike udaljenosti. Funkcioniše po principu baznih stanica koji primaju odgovarajuće signale tj. poruke na odgovarajućoj frekvenciji sa mobilnih telefona odnosno GSM modema, zatim manupuliše sa tim porukama i šalje ih baznu stanicu u okolini koje se nalazi GSM modem koji treba da primi poruku. Bazna stanica koja je primila odgovarajuće signale manipuliše sa njima I prosleđuje ih do modema primaoca. Glavna prednosti zbog kojih je upotrebljen GSM modem u ovoj aplikaciji je velika razdaljina na kojoj se može vršiti komunikacija, takođe i jednostavnost implementacije.



Slika 9. GSM 2 Click pločica

Na slici je prikazana *GSM 2 Click* pločica firme Mikroelektronika koja je iskorišćena u ovom projektu za ostvarenje GSM komunikacije sa mobilnim telefonom korisnika. *GSM 2 Click* pločica podržava *quad-band GSM/GPRS*, što je omogućava da bude korišćena širom sveta. Podržava širok spektar komunikacionih protokola i opcija za povezivanje, dok se njome upravlja preko jednostavnim AT komandnin interfejsom preko UART magistrale, tako da ova pločica omogućava širok spektar primena u M2M aplikacijama.

Na slici se vidi *quad-band GSM/GPRS* modem firme *Quectel M95* koji je ključna komponenta ove pločice. Ovaj modem pokriva frekvencije od 850/900 MHz sa sa snagom odašiljanja 2W, i frekvencije od 1800/1900 MHz sa snagom odašiljana do 1W. Takođe na slici vidimo da pločica na sebi ima *SMA antenna* connector na koji se povezuje antena kojom se signal prima i odašilja, više indikacionih dioda koje nam ukazuju na stanje napajanja, povezanost na mrežu, SIm card holder u koji se postavlja kartica odgovarajućeg mobilnog operatera, takođe i

3.5mm headset connector za povezivanje slušalica kojim možemo preslušati audio signal koji GSM modem zaprimi. Modem M95 mora da radi na stabilnom napajanju od 4V.

Implementacije GSM 2 Click

U okviru realizovane aplikacije GSM modem je iskorišćen za primanje komandi pomoću kojih se može pokretati i zaustavljati motor komandama *"Pokreni motor."* odnosno *"Zaustavi motor."* takođe omogućeno je i postavljanje brzine motora komandom *"Brzina"*.

```
uint8_t msg_motorOn[14] = "Pokreni motor.";
uint8_t msg_motorOff[15] = "Zaustavi motor.";
uint8_t msg_velocity[7] = "Brzina ";
```

Da bi se GSM 2 Click pločica pokrenula potrebno je poslati određeni naponski signal na RSK pin kojim se modem postavlja u radon stanje.

```
void gsmModemInit() {
   LATFbits.LATF0 = 0;
   CORETIMER_DelayMs(100);
   LATFbits.LATF0 = 1;
   CORETIMER_DelayMs(15000);
   LATFbits.LATF0 = 0;
}
```

Prikazana je jednostavna funkcija kojom pokrećemo GSM modem. RSK pin modema je povezan na F0 pin kontrolera poreko koga se šalje naponski signal u odgovarajućem intervalu vremena kojim signaliziramo modemu da se postavi u radno stanje za primanje i slanje odgovarajućih signala.

Funkcija *deleteMsg()* nam služi za brisanje poruka iz memorije, pošto mi M95 modem ograničenog kapaciteta memorije ovo je veoma bitna funkcija koja nam omogućava da slanje naših komandi ne bude ograničeno na memoriju modema. Celokupan sistem funkcioniše tako što stigne SMS poruka sa komandom, mikrokontroler je iščita i nakon toga se pomoću ove funkcije odmah briše i oslobađa memoriju za naredne komande. Za brisanje poruka koristi AT komanda CMGD kojom se briše sve *read* i *unread* poruke.

```
void deleteMsg() {
   // DELETE MESSAGES FROM MEMORY //
   UART2_Write("AT+CMGD=0,4\r", 13);
   flag = 0;
   while(! UART2_ReceiverIsReady());
   UART2_Read(rxbuffer, 6);
   for(int j = 0; j<6; j++) {
      if(rxbuffer[j] != at_ok[j])
           flag = 1;
   }
}
```

Ispod možemo videti i ključnu funkciju readMsg() u kojoj se iščitavaju pristigle poruke odnosno komande i na osnovu njih izvršavaju radnje koje je korisnik prethodno zahtevao SMS porukom. Na slici ispod se može videti početak funkcije, pošto je funkcija kompleksna u daljem tekstu biće objašnjena po delovima. Ključni deo prvog dela koji će biti objašnjen je AT komanda kojom se iščitavaju poruke sa UART2 port-a. AT komanda koja se koristi za primanje poruka je CMGR, koja kada primi vrednost jedan(AT+CMGR=1) čita prvu poruku koja mu pristigne.

```
int readMsg() {
   UART2_Write("AT+CMGR=1\r", 11);
   while(! UART2_ReceiverIsReady());
   UART2_Read(rxbuffer, 23);
```

U nastavku prikazan je nastavak funkcije *readMsg()* gde se primljena poruka smešta u *rxbuffer* i sa njom se dalje manipuliše. U okviru poruke koja stiže na UART2 port, nalaze se broj telefona sa kog je poslata poruku, datum i vreme slanja poruke. Prvih 12 cifara koje se smeste u *rxbuffer* su broj telefona sa kog je poslata poruka, i on premešta u promenljivu *num[]* da bi se sa njim moglo dalje manipulisati.

```
for(int j = 0; j<23; j++) {
   if(rxbuffer[j] != read_msg_hdr[j])
   flag = 1;
}
UART2_Read(rxbuffer, 13);
if(rxbuffer[13] == '"') {
   for(int j = 0; j < 12; j++)
      num[j] = rxbuffer[j];
} else {
   for(int j = 0; j < 13; j++)
      num[j] = rxbuffer[j];
}
UART2_Read(rxbuffer, 31); //IGNORE ", DATE AND TIME, AND \r\n
```

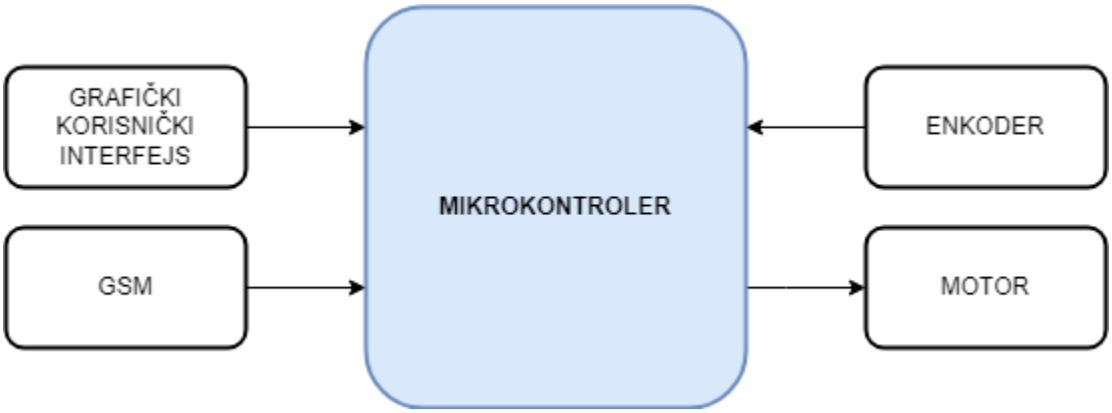
Ispod je prikazan deo koda funkcije msgRead() kojim se poredi pristigla poruka i očekivana komanda. Ukoliko je ispunjen neki od dole navedenih *if* uslova, povratna vrednost funkcije će biti odgovarajuća komanda. Ukoliko je pristigla naredba za brzinu motora, tada će povratna vrijednost funkcije biti željena brzina motora.

```
if(memcmp(text_msg, msg_motorOn, sizeof(msg_motorOn)) == 0)
   return GSM_MOTOR_ON;
if(memcmp(text_msg, msg_motorOff, sizeof(msg_motorOff)) == 0)
   return GSM_MOTOR_OFF;
if(memcmp(text_msg, msg_velocity, sizeof(msg_velocity)) == 0){
   if(text_msg[10] == '.') {
      velocity_tmp[2] = text_msg[7];
      velocity_tmp[1] = text_msg[8];
      velocity_tmp[0] = text_msg[9];
      gsmTmp = velocity_tmp[0] - 48;
      gsmTmp += (velocity_tmp[1] - 48) * 10;
      gsmTmp += (velocity_tmp[2] - 48) * 100;
      return gsmTmp;
```

```
    }
    if(text_msg[9] == '.') {
      velocity_tmp[1] = text_msg[7];
      velocity_tmp[0] = text_msg[8];
      gsmTmp = velocity_tmp[0] - 48;
      gsmTmp += (velocity_tmp[1] - 48) * 10;
      return gsmTmp;
    }
    if(text_msg[8] == '.') {
      velocity_tmp[0] = text_msg[7];
      gsmTmp = velocity_tmp[0] - 48;
      return gsmTmp;
    }
}
return 0;
}
```

Blok šema sistema

Na slici 10. je prikazana blok šema sistema.



Slika 10. Blok šema sistema

Sistem se sastoji od mikrokontrolera koji je centralni dio i upravlja ostatkom sistema, enkodera, motora, grafičkog korisničkog interfejsa koji je instaliran na računaru i povezan na sistem putem serijske komunikacije kao i GSM modema.

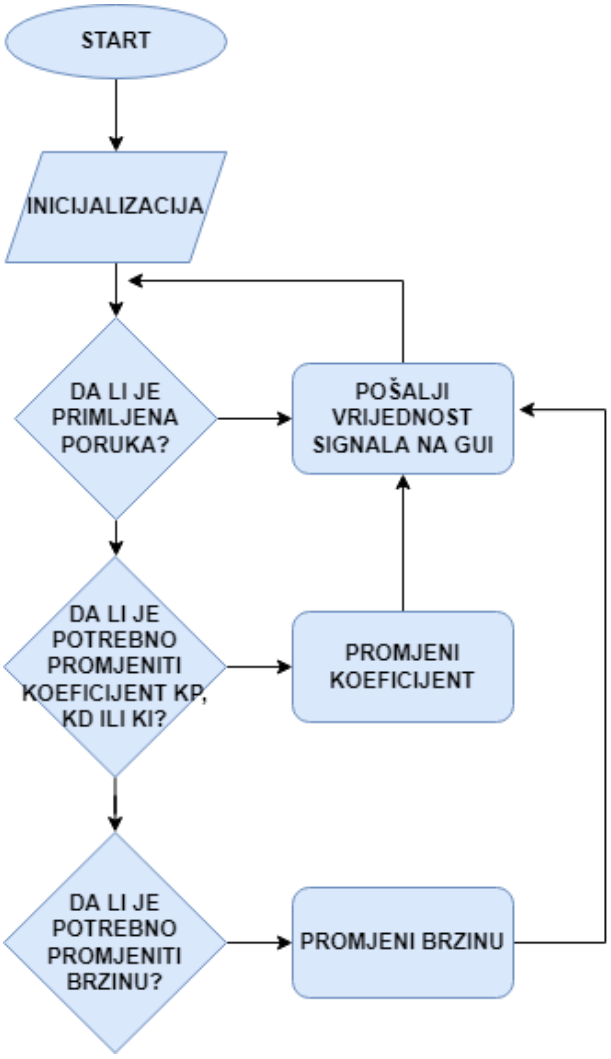
Mikrokontroler dobija informacije sa enkodera o poziciji motora, na osnovu toga vrši obradu odnosno generiše upravljački signal koji pokreće motor.

Sa grafičkog korisničkog interfejsa moguće je podešavati brzinu motora kao i parametre odnosno koeficijente PID signala.

Putem GSM modema, moguće je vršiti pokretanje i zaustavljanje motora SMS porukom. Takođe, postoji opcija za postavljanje brzine motora.

Algoritam rada sistema

Na slici 11. je prikazan dijagram toka za algoritam rada sistema.



Slika 11. Algoritam rada sistema

Na početku izvršavanja koda, potrebno je izvršiti inicijalizaciju vrijednosti koeficijenata PID regulacije. Takođe, potrebno je izvršiti inicijalizaciju GSM modema kao i svih neophodnih periferija za realizaciju projekta (pinovi mikrokontrolera, drajver za impulsno – širinsku modulaciju, drajveri za serijsku komunikaciju).

Nakon inicijalizacije, čeka se na prijem poruke na jedan od dva kanala za serijsku komunikaciju. Ukoliko je primljena poruka, provjerava se da li je potrebno mijenjati koeficijent za PID ili brzinu motora i u skladu sa tim se i izvršava operacija. Takođe, tokom svakog perioda izvršavanja koda, vrijednost signala se šalje na grafički korisnički interfejs putem serijske komunikacije.

Najvažnije funkcije implementirane u kodu

Glavni dio koda koji se izvršava u beskonačnoj petlji, prikazan je u sledećem dijelu teksta:

```
if(UART2_ReceiverIsReady()) {
   tmp = UART2_ReadByte();
   str_tmp[i] = tmp;
   i++;
   flag = 1;
   if(i == 17 && flag == 1) {
      for(int j = 0; j<17; j++)
           if(str_tmp[j] != msg_rec[j]) {
                flag = 0;
           }
      if(flag == 1) {
           ctrl = readMsg();
      }
      flag = 0;
      i = 0;
      deleteMsg();
   }
}
```

Na samom početku beskonačne petlje nalazi se dio koda koji je prikazan. Tu se vrši provjera da li je pristigla poruka sa GSM modema. Izvršava se provjera i da li je na GSM modem stigla neka SMS poruka, ukoliko jeste poziva se funkcija za čitanje te poruke. Na osnovu povratne vrijednosti ove funkcije, vrši se provjera šta tačno primljena poruka predstavlja. Nakon toga, poziva se funkcija za brisanje svih poruka iz memorije.

```
if(UART3_ReceiverIsReady() || ctrl != 0) {
   if(ctrl == -1) {
      if(last_velocity == GSM_MOTOR_ON)
           desired_velocity = 55;
      else
           desired_velocity = last_velocity;
      state = true;
      if(desired_velocity>500) desired_velocity=500;
      direction = tmp16 >> 15;
   }
   if(ctrl == GSM_MOTOR_OFF) {
      cmd8 = CMD_STOP_MOTOR;
      desired_velocity = 0;
   }
   if(ctrl == 0)
      cmd8 = UART3_ReadByte();
   if(ctrl > 0) {
      desired_velocity = ctrl;
      state = true;
      if(desired_velocity>500) desired_velocity=500;
      direction = tmp16 >> 15;
      last_velocity = desired_velocity;
   }
   ctrl = 0;
   switch (cmd8) {
      case CMD_SET_KP:
           UART3_Read(buffer,2);
           tmp16 = str_to_int16(buffer);
           Kp = tmp16/1000.0;
           break;
```

```
case CMD_SET_KI:
     UART3_Read(buffer,2);
     tmp16 = str_to_int16(buffer);
     Ki = tmp16/10000.0;
     break;
case CMD_SET_KD:
     UART3_Read(buffer,2);
     tmp16 = str_to_int16(buffer);
     Kd = tmp16/1000.0;
     break;
case CMD_START_MOTOR_PID:
     state = true;
     UART3_Read(buffer,2);
     tmp16 = str_to_int16(buffer);
     desired_velocity = tmp16 & 0x01FF;
     if(desired_velocity>500) desired_velocity=500;
     direction = tmp16 >> 15;
     last_velocity = desired_velocity;
     break;
case CMD_STOP_MOTOR:
     state = false;
     stop_motor();
     break;
case CMD_RESET:
     state = false;
     desired_velocity = 0;
     error_velocity = 0;
     error_accumulator = 0;
     error_derivative = 0;
     Kp = 0;
```

```
            Ki = 0;
            Kd = 0;
            stop_motor();
            break;
    }
    cmd8 = 0;
}
```

U prvom dijelu koda provjerava se da li je poruka primljena od GSM modema ili putem grafičkog korisničkog interfejsa i na taj način se dodjeljuje vrijednost promjenjivoj cmd8 ili pozivaju određene funkcije. Nakon toga postoji switch-case naredba koja izvršava operaciju odnosno poziva određenu funkciju.

```
if(update_count>=10){
    velocity=(uint16_t)((500.0/7140.0)*(float)encoder_get_velocity());
    error_previous = error_velocity;
    error_velocity = pid_error(desired_velocity,velocity);
    error_accumulator += error_velocity;
    error_derivative = error_previous - error_velocity;
    pid_output = pid_control(error_velocity, error_accumulator,
    error_derivative);
    if(state)start_motor(pid_output,direction);
    sprintf(str,"G %u %u %d %u\n", desired_velocity, velocity,
    error_velocity, pid_output);
    UART3_Write(str,strlen(str));
    update_count=0;
}
```

Na kraju koda, vrši se proračun greške za PID regulaciju i na taj način se dobija upravljački signal. Nakon toga se vrši slanje parametara putem serijske komunikacije kako bi bilo moguće u grafičkom korisničkom interfejsu iscrtati signal.

Osim glavnog dijela koda, potrebno je navesti i par najvažnijih funkcija za implementaciju algoritma. Neke od najvažnijih funkcija su navedene u nastavku.

```
void start_motor(uint16_t speed, MOT_DIR direction) {
   MOTOR_INH = 1;
   set_speed(speed);
   set_direction(direction);
}
```

Ova funkcija pokreće motor odnosno postavlja brzinu i smjer obrtanja motora.

```
void set_speed(uint16_t speed) {
   uint16_t duty;
   if(MOTOR_DIRECTION)
      speed=6000-speed;
   duty = speed;// * MAX_MOTOR_SPEED / 100;
   MCPWM_ChannelPrimaryDutySet(1,duty);
}
```

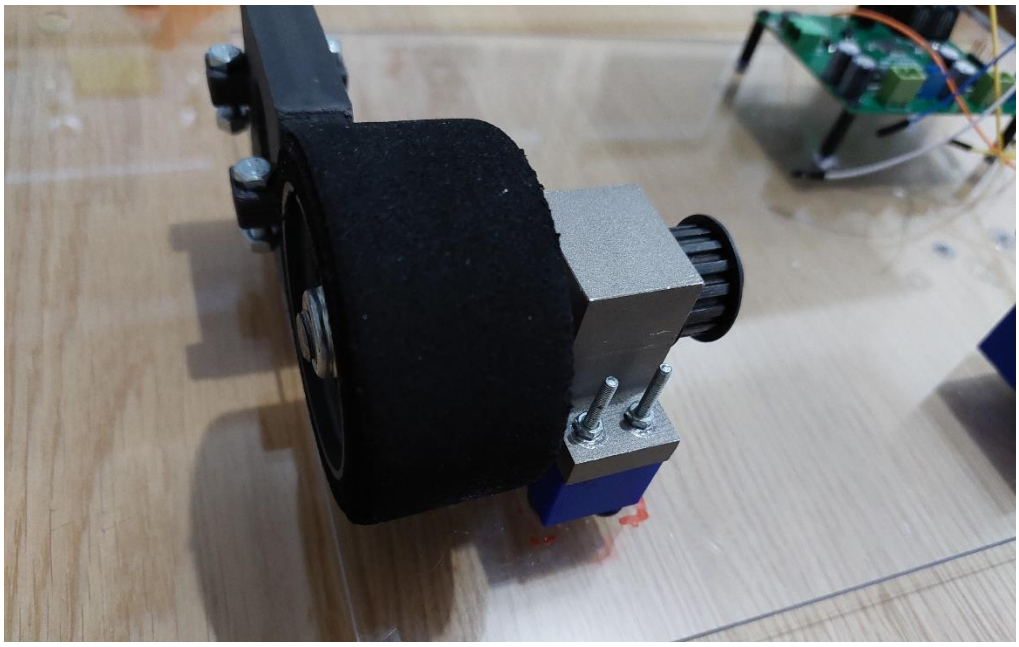
Ova funkcija postavlja brzinu motora tako što zapravo postavlja faktor ispune na određenu vrijednost. Funkcija set_direction samo postavlja jedan od dva moguća smjera obrtanja motora.

```
uint16_t pid_control(int error, int error_accumulator, int
error_derivative) {
   static float PID = 0.0;
   PID+=(Kp*error+Ki*error_accumulator+Kd*error_derivative)*6000/
500;
   if(PID > PID_MAX)PID = PID_MAX;
   if(PID < PID_MIN)PID = PID_MIN;
   return (uint16_t)PID;
}
```

Funkcija pid_control se koristi za izračunavanje signala na osnovu prosleđenih parametara i koeficijenata Kp, Ki i Kd.

Rezultati testiranja

Na slikama 12. i 13. prikazana su opterećenja koja su korišćena za testiranja. Na slici 12. prikazano je linearno opterećenje a na slici 13. nelinearno opterećenje.



Slika 13. Linearno opterećenje

Slika 12. Nelinearno opterećenje

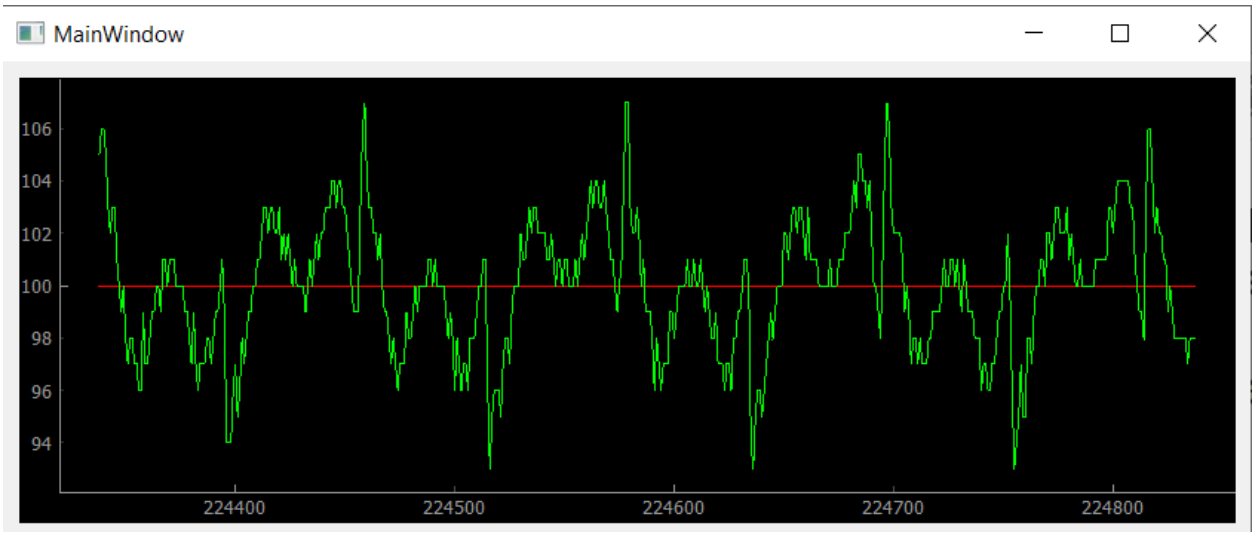
Najbolji rezultati odnosno izgled signala, dobijeni su za sledeće vrijednosti koeficijenata:

Kp = 328;

Kd = 80;

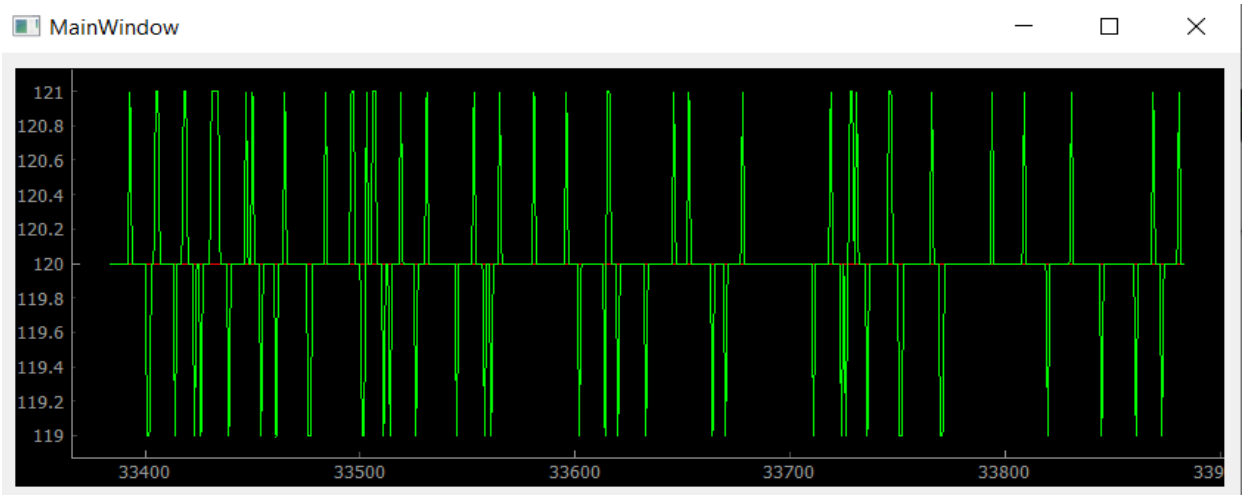
Ki = 50.

Izgled signala sa ovim koeficijentima prikazan je na slici 14. Kao što se može vidjeti na slici, signal u ustaljenom stanju odstupa za oko 6%.



Slika 14. Signal sa nelinearnim opterećenjem

Na slici 15. prikazan je izgled signala sa istim koeficijentima, ali ovoga puta za linearno opterećenje. Kao što se može vidjeti, sada je odstupanje signala u ustaljenom stanju oko 1%.



Slika 15. Signal sa linearnim opterećenjem

Zaključak

Prilikom izrade projekta urađeno je sve što je bilo propisano projektnim zadatkom. Omogućeno je upravljanje motorom pomoću PID regulacije. Parametre je moguće podešavati putem grafičkog korisničkog interfejsa. Uspješno je implementiran i GSM modem, tako da se brzina motora, kao i pokretanje i zaustavljanje može vršiti putem SMS poruke.

Na kraju je detaljno testirano sve što je bilo zadato projektnim zadatkom. Izvšeno je testiranje linearnog i nelinearnog opterećenja motora. Na kraju su rezultati upoređeni što je i prikazano u projektnoj dokumentaciji.

Zaključak je da je projekat uspješno realizovan i da je implementirano i testirano sve što je bilo potrebno.

Literatura

[1] https://www.ni.com/en-rs/innovations/white-papers/06/pid-theory-explained.html, januar 2022.

[2] https://download.mikroe.com/documents/datasheets/m95-hardware-design-v1.3.pdf, januar 2022.

[3] https://www.microchip.com/en-us/product/PIC32MK1024MCF064, januar 2022.

[4] https://www.sparkfun.com/datasheets/Cellular%20Modules/AT_Commands_Reference_Guide_r0.pdf, januar 2022.

[5] https://theautomization.com/pid-control-basics-in-detail-part-2, januar 2022.