Univerzitet u Beogradu
Elektrotehnički fakultet

Balansiranje loptice na šini

Projekat iz predmeta mašinska vizija i primena mikrokontrolera

Mentor:
Prof. dr Nenad Jovičić

Student:
Kristijan Mitrović, 3132/19

Beograd, Jun 2020.

Sadržaj

Glava 1

Uvod

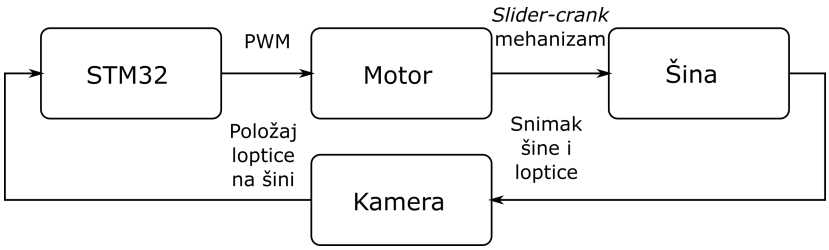
1.1 Motivacija

Projekat je izabran iz razloga što na praktičan način kombinuje mašinsku viziju i upravljanje preko mikrokontrolera. Problem je naizgled jednostavan, ali tehnički prilično zahtevan, što zbog mehaničke prirode problema, što zbog dodatne komplikacije nastale zatvaranjem povratne sprege preko kamere, čime se inherentno uvode problemi vezani za obradu slike u realnom vremenu. Jednom kada je na uniforman način postignuta obrada snimka sa kamere, jednostavan interfejs prema mikrokontroleru omogućava implementaciju *PID* regulatora koji potom preko *PWM*-a kontroliše servo motor kojim se posredno kontroliše položaj šine te i loptice na njoj.

U narednim sekcijama biće predstavljen izgled sistema i detaljnije objašnjen sam rad pojedinačnih delova.

1.2 Šema sistema

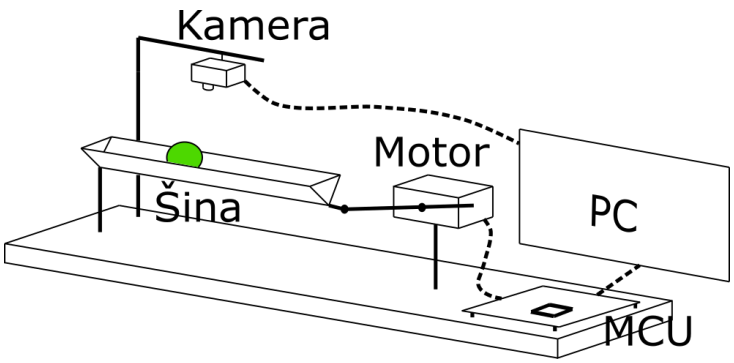
Na slici 1.1 data je blok-šema sistema.



Slika 1.1: *Blok šema sistema*

Šina je jednim krajem učvršćena za naslon oko kog može da rotira, dok je drugim krajem preko *slider-crank* mehanizma (mehanizam poput onog na točkovima lokomotive) pričvršćena za osovinu motora. Promena nagiba šine kontroliše se položajem osovine motora, što uslovljava kotrljanje loptice po njoj. Cilj je održati lopticu na sredini šine. Kako bi se dobila informacija o tome gde je loptica, iznad šine montirana je kamera koja konstantno prati kretanje loptice i računar obaveštava o njenom položaju.

Izgled ovog sistema dat je na slici 1.2.



Slika 1.2: *Izgled sistema*

Sa prethodnih slika se vidi da je sistem u zatvorenoj sprezi i da kontroler informaciju o položaju loptice, na osnovu koje potom proračunava upravljanje motorom, dobija preko kamere. Računar je samo posrednik između kamere i kontrolera kako bi se sirova informacija sa senzora (kamere) na pogodan način obradila i predala kontroleru.

Konkretno govoreći, obrada slike na računaru rađena je korišćenjem *OpenCV* biblioteke. Nakon obrade, preko *UART*-a se sa računara kontroleru šalje informacija o polozaju loptice na šini, koju kontroler koristi kako bi sračunao upravljanje primenom *PID* regulatora. To upravljanje pogodno se transformiše u *duty cycle PWM* signala kojim se kontroliše položaj osovine motora, a time i nagib šine i položaj loptice na njoj.

Glava 2

Mašinska vizija

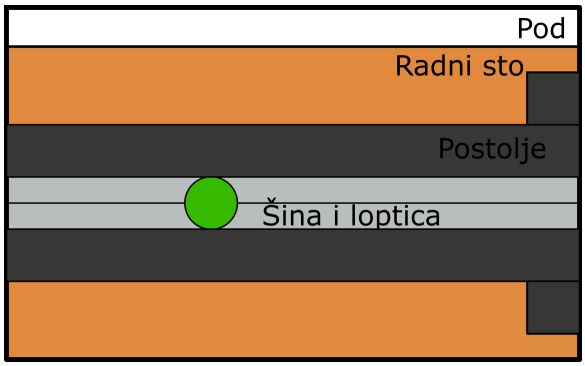
OpenCV biblioteka, u kojoj je implementirano pregršt funkcija korisnih u standardnim zadacima vezanim za mašinsku viziju, pokazala se vrlo korisnom prilikom obrade slike sa kamere. Pisanje funkcija koje se često koriste od nule često je zamorno i podložno greškama. Stoga ova biblioteka predstavlja nezamenljiv alat pri programiranju neke realne primene poput ove.

U suštini, program na računaru, pisan u *C++*-u, se sastoji od dve velike funkcije: `calibrate` i `run`, koje će biti opisane u narednim sekcijama.

2.1 Funkcija `calibrate`

Zbog fizičke mane sistema, koji je sam po sebi glomazan i težak za nošenje, nosač na koji je zakačena kamera nije fiksiran i u nekim situacijama se okreće oko svoje ose. Kako bi se ovaj problem rešio, implementirana je funkcija `calibrate` koja bi pre početka rada programa trebalo da pomogne korisniku da nosač kamere i samu kameru pravilno pozicionira.

Sve što ova funkcija radi je dohvatanje frejma sa kamere i njegovo isecanje na predefinisane dimenzije, i to tako da na slici samo ostane prostor širok i visok taman toliko koliko je dovoljno da cela šina stane u frejm, kao što je prikazano na sledećim slikama.



Slika 2.1: *Originalni frejm sa kamere*

Na slici 2.1 se uočavaju i elementi poput postolja na kom ceo sistem stoji, kao i stola ili poda i sličnih objekata koji su irelevantni za informaciju koju sa slike treba

Slika 2.2: *Isečen frejm sa kamere*

izvući, a to je položaj loptice na šini. Na slici 2.2 se vidi isečen frejm sa kamere koji je omogućio pravilno podešavanje nosača kamere tako da je sada samo vidljiva šina sa lopticom. Ovaj korak kalibracije nužan je ne samo zbog fizičkog položaja kamere, već se isecanjem frejma sa slike uklanjaju svi suvišni elementi koji, što zbog svog oblika, što zbog refleksije svetla sa njih, prave ozbilje probleme pri detekciji loptice. Na ovaj način je interferencija svedena na minimum.

2.2 Funkcija `run`

Glavna funkcija programa je funkcija `run`. Ova funkcija u beskonačnoj petlji dohvata frejm sa kamere, vrši njegovo isecanje i procesiranje tako da se dobije crno-bela slika na kojoj se potom mogu lako detektovati oblici. Bitno je reći da je iskorišćena činjenica da je šina crne boje, a loptica na njoj bele, što je omogućilo prebacivanje slike u *HSV* kolor sistem u kome je poređenjem sa pragom potom lako izdvojiti svetlu lopticu. Korišćenjem ugrađenih *OpenCV* funkcija na toj slici se detektuje kontura loptice i oko nje upisuje kružnica. Centar te kružnice, tj. njegova horizontalna koordinata, šalje se preko serijskog porta mikrokontroleru. Ova vrednost je zapravo indeks piksela na kom se nalazi centar loptice, ali je direktno srazmerna položaju loptice na šini u bilo kojim fizičkim jedinicama za rastojanje.

Pseudo *C++* kod, zajedno sa kratkim komentarima, dat je u prilogu, kako bi se lakše sagledao tok ove funkcije.

```
int run()
{
  // Instanciranje objekta kamere i serijskog porta
  VideoCapture cap;
  SerialPort* serial = new SerialPort(port_name);

  while (true)
  {
    // Dohvatanje frejma sa kamere
    cap.read(frame);
    // Isecanje frejma tako da se samo sina i loptica vide
    cropped = frame(Range(low, high), Range::all());
    // Procesiranje isecene slike poput blurovanja,
    // prebacivanja u HSV color space, erozije i dilatacije
    process(cropped, processed);

    // Nalazenje kontura na crno-beloj slici koriscenjem OpenCV fja
    findContours(processed, contours, ...);

    // Ako je kontura uspesno nadjena
    if (contours.size()>0)
    {
      // Zbog pogodnog isecanja i procesiranja slike, de facto
      // najveca kontura je loptica
      largestContour = findLargestContour(contours);
```

```
      // Nalazi se kruznica najmanjeg precnika koja obuhvata
      // konturu loptice i njen centar
      minEnclosingCircle(largestContour, center, radius);

      // ...

      // x koordinata centra se salje preko UARTa kontroleru
      sendValueOverUart(serial, center.x);
    }

    // Prikaz frejma koji je dodatno ukrasen kruznicom oko loptice
    imshow("Camera", frame);

  }
}
```

Pošto je cilj lopticu zadržati na sredini šine, treba imati na umu šta se zapravo šalje kontroleru: kako je kamera pozicionirana direktno iznad šine, to znači da, iz ugla kamere gledano, kada je loptica na krajnjem levom kraju šine, horizontalna koordinata centra ima malu vrednost (oko 50, zbog širine same loptice), a u krajnjem desnom položaju ima veliku vrednost (oko 600, jer je širina frejma 640 piksela). Dakle, lopticu je potrebno zadržati negde na 322. pikselu. Svi položaji desno od toga daće pozitivnu grešku, a svi položaji levo negativnu. Pošto je šina fizički široka oko 25 cm, greške od par piksela su zanemarljivo male.

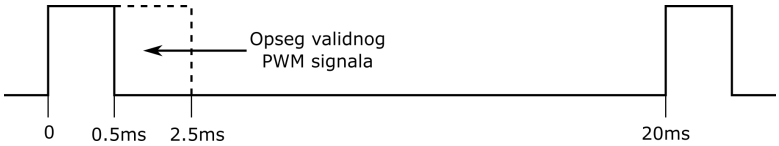
Još jedna bitna stvar koju je potrebno napomenuti je problem sinhronizacije kontrolera i računara. Naime, nemoguće je predvideti tačne trenutke u kojima će računar završiti obradu frejma sa kamere. Tačno je da frejmovi pristižu po predefinisanom *frame rate*-u, ali obrada svakog frejma zahteva neko vreme koje inherentno unosi kašnjenje. Zavisno od toga da li je loptica direktno vidljiva ili postoji neka opstrukcija (poput ruke), te se ona ne može detektovati, jedna iteracija algoritma ne traje uvek podjednako. Stoga se može uzeti da sa računara preko serijskog porta asinhrono, ali konstantno pristižu podaci. Na kontroleru je da ovaj problem reši.

Glava 3

Primena mikrokontrolera

3.1 Upravljanje motorom

Upravljanje servo motorom vrši se preko PWM-a. Naime, motor ima tri dovodne žice: $V+$, GND i PWM. Motor se napaja preko eksternog adaptera od 5 V, a PWM i GND žicama je spojen na jedan digitalni izlaz kontrolera. Motor očekuje PWM impuls učestanosti 50 Hz, čiji je *duty cycle* u opsegu $[0.025, 0.125]$ %. Tj. trajanje logičke jedinice PWM signala je u opsegu $[0.5, 2.5]$ ms, kao što je prikazano na slici 3.1.



Slika 3.1: *Opseg validnog PWM signala*

3.2 Tajmer *TIM3* i generisanje *PWM*-a

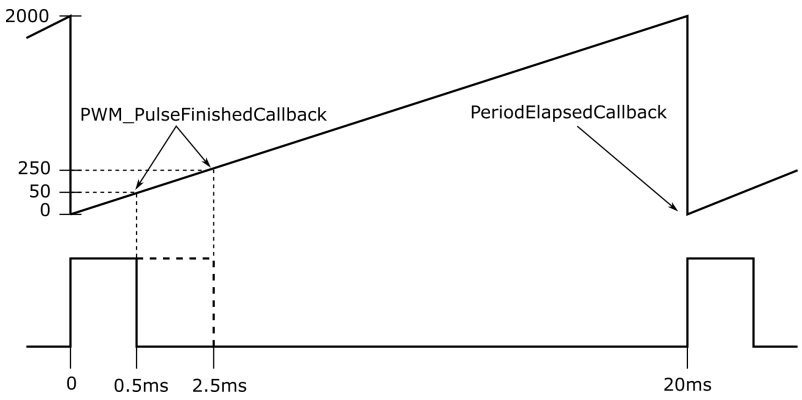
Generisanje PWM signala postignuto je korišćenjem tajmera *TIM3*. *Prescaler* i *AutoReload* registar su podešeni tako da je postignuto prirodno mapiranje digitalnih vrednosti koje tajmer broji i validnog opsega PWM signala. Naime, pošto je potrebno dobiti signal od 50 Hz, a za *TIM3* je iskorišćen interni *clock* od 48 MHz, iz

$$\frac{CLK\_FREQ}{DESIRED\_FREQ} = PSC \times AUTOREL = \frac{48\,M}{50} = 960000$$

se vidi da je moguće formirati vrednost *AutoReload* registra od 2000, čime je *Prescaler* fiksiran na 480. Ovo znači da će *clock* koji se dovodi do tajmera biti skaliran tako da tajmeru treba 2000 klokova da bi prošlo 20 ms, tj. tajmer treba da odbroji do 2000. Zatim je *CH1* tajmera *TIM3* iskorišćen da se na njemu generiše PWM signal kao sa slike 3.1. *Duty cycle* željenog signala podešava se upisom u *CCR1* registar tajmera *TIM3*, a *HAL* biblioteka to dodatno olakšava zajedno sa *CubeMX* okruženjem. U *CCR1* registar može se upisati vrednost iz opsega $[50, 250]$, što je direktno i prirodno preslikavanje na opseg $[0.5, 2.5]$ ms. PWM s levog kraja opsega

osovinu motora postavlja u "nulti" položaj, a *PWM* sa drugog kraja u položaj koji je 180° spram nultog.

Nakon odbrojavanja do vrednosti upisane u *AutoReload* registar, tajmer generiše `PeriodElapsed` prekid, a nakon odbrojavanja do vrednosti upisane u *CCR1* registar generiše `PWM_PulseFinished` prekid. Ove prekide moguće je povezati sa odgovarajućim prekidnim rutinama:
`HAL_TIM_PeriodElapsedCallback` i `HAL_TIM_PWM_PulseFinishedCallback`,
kao što je prikazano na slici 3.2.



Slika 3.2: *Opseg validnog PWM signala*

Ove rutine su namerno ostavljene nedefinisane unutar *HAL* biblioteke, kako bi korisnik mogao da ih redefiniše u svom kodu. Ovde konkretno, ove rutine su iskorišćene kako bi se vrednost digitalnog pina *PA5* postavila na visok ili nizak nivo. Premda je kanal *CH1* tajmera *TIM3* direktno vezan na pin *PA6*, i sa njega se može "skinuti" potreban signal, ovo je urađeno zbog toga što je na pin *PA5* vezana ugrađena LE dioda na razvojnoj ploči. Kako je *PWM* signal na 50 Hz, promenom *duty cycle*-a postiže se efekat prigušenja ili pojačanja svetla sa diode, što je vizuelno zgodna stvar, tako da je ipak *PWM* žica sa motora fizički vezana na *PA5* pin.

3.3 Tajmer *TIM6* i problem sinhronizacije

Kao što je prethodno objašnjeno, usled nejednakog vremena potrebnog za obradu pojedinačnih frejmova sa kamere, ne može se uspostaviti sinhrono slanje podataka o centru loptice sa računara na kontroler. Stoga se smatra da podaci konstantno pristižu na kontroler. Ovo je ispoštovano tako što je `main` funkcija kontrolera organizovana na sledeći način:

```
int main()
{
  // Inicijalizacija periferija i klokova
  // ...
  // Deklaracija nekih promenljivih poput PID konstanti
  // ...
  while (1)
  {
    // Citanje podatka iznova i iznova
    // i belezenje u globalni bafer val
    receiveValue(&val);
```

```
    if (FLAG_10ms)
    {
      // PID regulacija
    }
  }
}
```

U beskonačnoj petlji se iznova i iznova beleži podatak pristigao sa računara korišćenjem `receiveValue` funkcije, koja je, u suštini, *wrapper* oko `HAL_UART_Receive` funkcije, i koja u globalnu promenljivu `val` beleži novopristiglu vrednost.

Kako bi upravljanje motorom bilo sinhrono, to je tajmer *TIM6* iskorišćen i podešen da na svakih 10 ms generiše prekid, na koji se postavlja fleg `FLAG_10ms`. Ovaj fleg se hvata u `main`-u i kreće se sa proračunavanjem upravljanja koje se zadaje motoru. Tako da, suštinski, može se smatrati da se novi podatak dobija na svakih 10 ms, jer se vrednost `val` bafera koristi tek kada se uhvati fleg.

3.4 PID regulacija

PID kontroler implementiran je tako da, pošto se uhvati fleg za istek 10 ms, od vrednosti upisane u `val`, što je vrednost koja je pristigla sa računara, se oduzima 322, što je vrednost koja predstavlja centralni piksel šine, i na taj način formira signal greške. Ovaj signal se na tipičan način koristi u sračunavanju pojedinačnih P, I i D dejstava, uz dodatno ograničenje integralnog dejstva kako bi se sprečilo navijanje integratora.

Zbog mehaničke konstrukcije sistema i *slider-crank* mehanizma, šina je postavljena horizontalno za vrednost od oko 170 upisanu u $CCR1$ registar. Dakle, nije 150, kako bi se očekivalo s obzirom na moguć opseg validnih vrednosti, već 170. Pošto je signal greške koncipiran tako da može biti i negativan i pozitivan, odlučeno je da se upravljanje zadrži u nekoj okolini broja 170, tj. u okolini horizontalnog položaja šine. Ovo je urađeno kako bi se sistem, u suštini, linearizovao za mali ugao nagiba šine, a pride i da ne pravi problem kameri, već da zadrži šinu unutar frejma. Zbog svega ovoga, izlaz *PID*-a se dodaje na 170 i taj broj upisuje u *CCR1* registar tajmera *TIM3*, čime se efektivno postiže kretanje šine u maloj okolini horizontalnog položaja. Sledećim kodom ilustrovana je *PID* regulacija:

```
if (FLAG_10ms){
  // Sracuna se signal greske
  dist_error = val - 322;
  // delta_error, potrebno za diferencijalno dejstvo
  derror = dist_error - dist_prev_error;

  // Sracunaju se pojedinacna dejstva
  PID_p = Kp * dist_error;
  PID_d = Kd * (derror/dt);
  PID_i = PID_i + dist_error*dt;
  // Zastita integratora od navijanja odsecanjem
  if (PID_i > maxStep)
    PID_i = maxStep;
  else if (PID_i < -maxStep)
    PID_i = -maxStep;
```

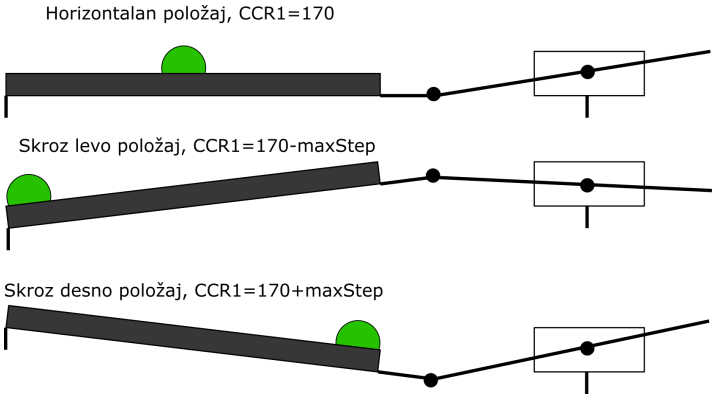
```
  // Kombinovani izlaz PID-a
  PID = PID_p + Ki * PID_i + PID_d;

  // Izlaz PID-a se direktno prevodi u upravljanje,
  // koje je u [170 - maxStep, 170 + maxStep] opsegu
  step = PID;
  if (step > maxStep)
    step = maxStep;
  if (step < -maxStep)
    step = -maxStep;

  // Motor se pomera za step vrednost oko horizontalnog polozaja
  pulse = 170 + step;
  setPulse(&htim3, pulse);

  // Priprema za novu iteraciju
  dist_prev_error = dist_error;
  FLAG_10ms = 0;
}
```

Na slici 3.3 prikazan je efekat koji ovakav dizajn upravljanja ima.



Slika 3.3: *Mogući položaji šine*

Dakle, osovina motora se kreće po relativno malom opsegu uglova, sasvim dovoljnom da lopticu dovede do krajeva šine.