SVEUČILIŠTE U ZAGREBU

FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

DIPLOMSKI RAD br. 2035

Simulacija upravljanja i ostvarivanje prikaza kinematičkog sustava

Stjepan Čajić

Zagreb, prosinac 2009.

SADRŽAJ

1. Diplomski zadatak

Proučiti kinematičke strukture te načine povezivanja i modeliranja navedenih struktura. Proučiti načine fizičke realizacije kinematičkih struktura i upravljanja takvim strukturama. Razraditi fizičku realizaciju kinematičke strukture. Načiniti model koji odgovara fizički realiziranoj strukturi i omogućava simulaciju tog modela. Ostvariti prikaz simulacije virtualnog modela i usporediti ga sa izvođenjem pokreta stvarnog modela. Analizirati razlike uspoređenih modela. Na nizu primjera prezentirati ostvarene rezultate. Izraditi odgovarajući programski proizvod. Koristiti programski jezik C++ i grafički standard OpenGL. Rezultate prezentirati putem Interneta. Radu priložiti algoritme, izvorne kodove i rezultate uz potrebna objašnjenja i dokumentaciju. Citirati korištenu literaturu i navesti dobivenu pomoć.

2. Uvod

U današnjem svijetu vlada ekspanzija video uradaka kao što su filmovi, televizijske serije, emisije, vijesti i reklame. To se, između ostalog, odražava i na rastući trend tehničke kvalitete tih materijala.

Zato se prilikom izrade većine video snimaka, koriste tehnička pomagala kako bi se postigli oku što ugodniji i pravilniji kadrovi, a jedno od takvih pomagala je i dizalica za video kameru.

Najčešća uporaba tog pomagala je podizanje kamere i odmicanje od objekta snimanja, što je i čest način završavanja filmova.

Prilikom podizanja kamere, mijenja se njen položaj u prostoru, pa ju je često potrebno stalno okretati prema objektu snimanja. U ovom radu opisati ćemo izvedbu elektroničkog uređaja koji automatizira takvo okretanje kamere potrebno da bi se objekt stalno očuvao u centru kadra.

3. Kinematičke strukture dizalice za kameru

Dizalica za kameru sastoji se od 3 segmenata povezanih sa 2 zgloba.

3.1. Prvi segment – stalak

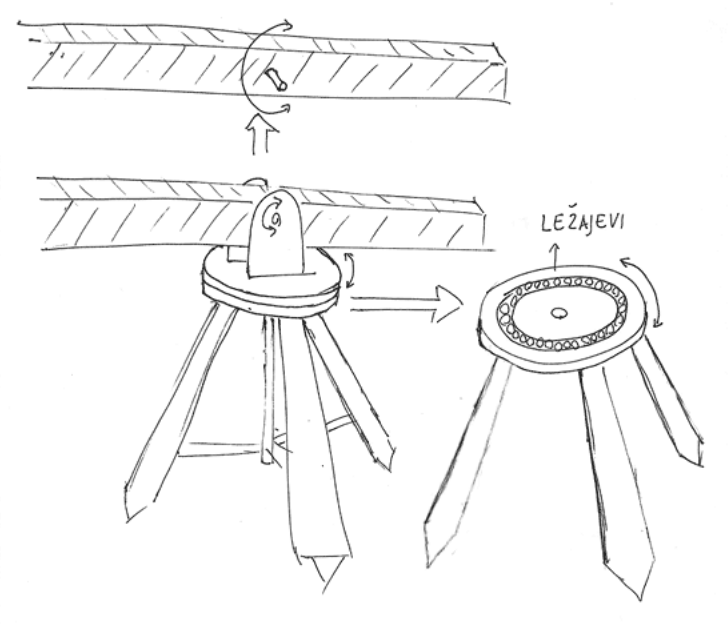
Slika 3.1.1. Stalak dizalice za kameru.

Stalak je osnova koja nosi cijelu dizalicu. Sastoji se od tri noge na vrhu kojih je zglob koji ga spaja s „rukom“ krana (Slika 3.1.1.) Baš kao što je računalnoj grafici pri trodimenzionalnim prikazima objekata uobičajeno koristiti samo 3 točke za prikaz ravnine ili poligona (jer već četvrta točka ne mora nužno ležati na istoj ravnini), tako se i na stalku koriste samo tri noge. Kada bi koristili više od tri noge, na neravnim površinama jedna bi se noga mogla naći u zraku, što bi moglo dovesti do klimanja i nestabilnosti cijele konstrukcije.

U stvarnim, dinamičnim uvjetima snimanja gdje često imamo ograničenu količinu vremena, poželjno je omogućiti relativno brzo premještanje dizalice sa jednog mjesta na drugo.

To je razlog zašto se na dnu svake noge nalaze kotači koji omogućavaju brzo premještanje dizalice.

3.2. Prvi zglob



Slika 3.2.1. Zglob s 2 stupnja slobode.

„Ruka“ je na stalak pričvršćena preko osovine koja prolazi kroz nju i naslanja se na dva željezna nosača. Taj spoj omogućuje rotaciju ruke u smjeru gore-dolje. Rotacija u smjeru gore-dolje ograničena je doticajem prednjeg ili stražnjeg kraka ruke sa podlogom. Budući da je njena dužina modularna tako su i ograničenja promjenljiva. U ovom dokumentu promatrati ćemo ruku dužine 6 metara, tako da su dozvoljeni kutovi rotacije u intervalu <-15°,60°>, gdje pozitivni kut označava podignut prednji krak ruke.

Ta dva nosača pričvršćena su na dvije okrugle ploče između kojih se nalaze ležajevi. Te dvije ploče omogućuju drugi stupanj slobode rotacije: „lijevo-desno“ kao što prikazuje Slika 3.2.1.

Dakle prvi zglob omogućuje rotaciju ruke u dva stupnja slobode: „gore-dolje“ i „lijevo-desno“.

3.3. Drugi segment – Ruka dizalice.

Slika 3.3.1. Ruka dizalice.

Ruka dizalice, kao što smo već ranije spomenuli, pričrvšćena je za stalak preko zgloba koji ima 2 stupnja slobode. Dužina ruke može se mijenjati dodavanjem produžetaka, ali, kao što smo već napomenuli, u ovom dokumentu ćemo razmatrati duljinu ruke od 6 metara. Na njenom prednjem kraku (dužem) nalazi se robotska glava koja elektromotorima okreće kameru. Robotska glava i kamera imaju svoju masu, pa je potrebno i na suprotnom (kraćem) kraku ruke imati ravnotežnu masu koja bi dovela strukturu u ravnotežni položaj. Za tu svrhu koristimo utege koje stavljamo na kraj stražnjeg kraka ruke (kao što radi analogna vaga).

Budući da se svaki kilogram ruke, glave, kamere, i utega prenosi na stalak, bitno je osigurati što manju ukupnu masu tih dijelova. Iz toga razloga, ruka je napravljena od aluminija, koji je u odnosu na ostale raspoložive metale dosta lagan.

Iz istog razloga, bitno je postići što duži stražnji krak ruke, jer time postižemo duži krak poluge, i samim time može se koristiti manje utega. Međutim, moramo paziti da stražnji krak ruke ne bude predugačak, jer njegova dužina ograničava moguću rotaciju prednjeg dijela ruke prema gore, i samim time se smanjuje moguća visina kamere. Okvirno se težina ovakve konstrukcije kreće oko 200 kg.

3.4. Drugi zglob – robotska glava

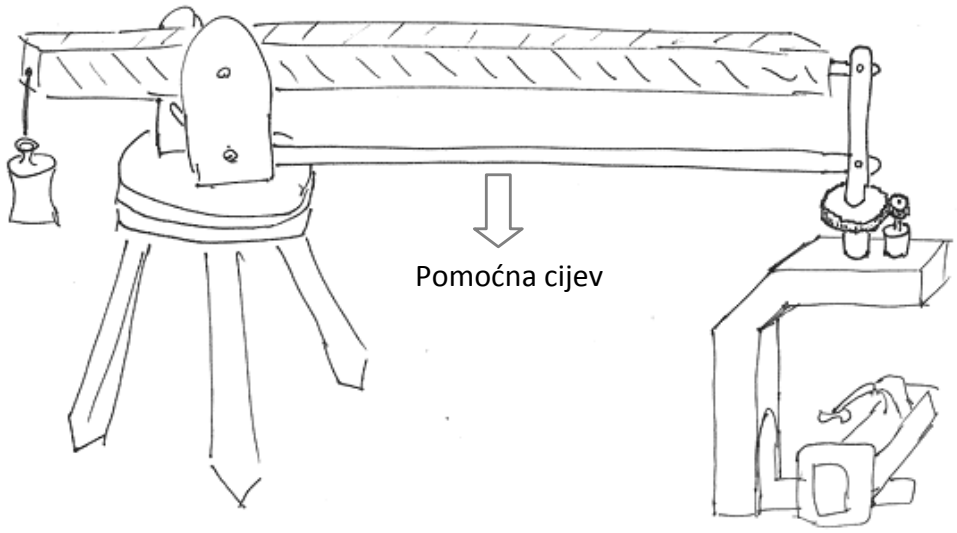
Slika 3.4.1. Robotska glava pričvršćena na vrh ruke dizalice.

Svrha robotske glave za kameru, je rotacija kuta snimanja kamere u dva stupnja slobode: „gore-dolje“, i „lijevo-desno“. Samu glavu mogli bi razdvojiti na 2 segmenta i dva zgloba jednog stupnja slobode. Njen prvi segment i prvi zglob, zaduženi za rotaciju „lijevo-desno“, spojeni su s rukom preko metalne osovine na koju je pričvršćen veliki zupčanik. Elektromotor je prislonjen uz taj veliki zupčanik i on je zadužen za rotaciju glave „lijevo-desno“.

Slika 3.4.2. Robotska glava za kameru izbliza.

Rotacija kamere „gore-dolje“ ostvarena je na sličan način. Drugi segment glave kamere na koji se neposredno pričvršćuje kamera povezan je također preko metalne osovine, zupčanika i elektromotora s prvim segmentom glave, i tako zajedno tvore jedan zglob koji omogućuje rotaciju kamere u dva stupnja slobode.

3.5. Pomoćni segment (pomoćna cijev)



Slika 3.5.1. Pomoćna cijev.

Svrha pomoćne cijevi je konstantno održavati glavu okomitom na zemlju. Ima više razloga zašto je to bitno postići. Prvi vidimo već iz pogleda na konstrukciju spoja glave i ruke. Nepomična osovina koja spaja glavu i ruku spojena je s glavom preko ležaja (da bi se glava mogla okretati lijevo desno). Kada glava ne bi bila uvijek okomita na zemlju, i nakriviljavala se podizanjem ruke, tada bi zbog sile teže dolazilo do neravnomjernog opterećenja na ležaju.

Drugi razlog je taj da bi u slučaju nakrivljenosti, moglo doći do toga, da se motor u jednu stranu mora „penjati“ po zupčaniku, a u drugu stranu „silaziti“ sa zupčanika, što bi iziskivalo motore veće snage.

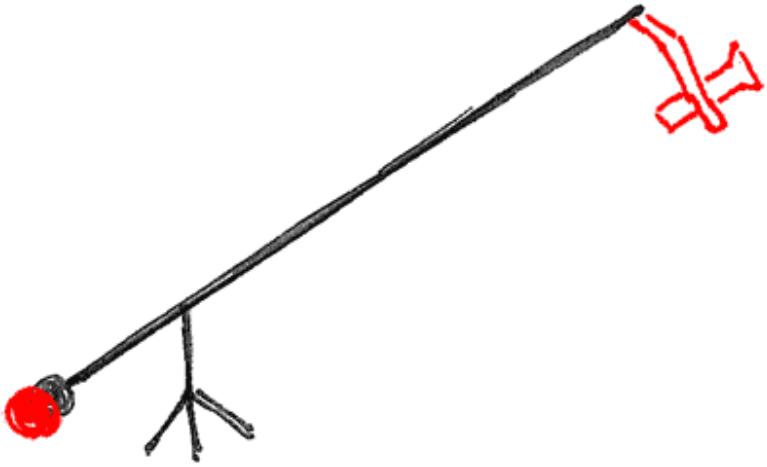
Čak i kada bi koristili vrlo snažne ležajeve i motore, ostao bi još jedan razlog zašto je bitna okomitost glave. Takva struktura bi uzrokovala neravnotežu ruke dizalice, na slijedeći način:

Slika 3.5.2. Dizalica bez glave.

Slika 3.5.2. - Prvo zamislimo dizalicu bez glave. Na nju postavimo točno onoliko utega koliko je potrebno da bi ruka došla u ravnotežu (zato što je duži krak ruke teži od kraćeg).

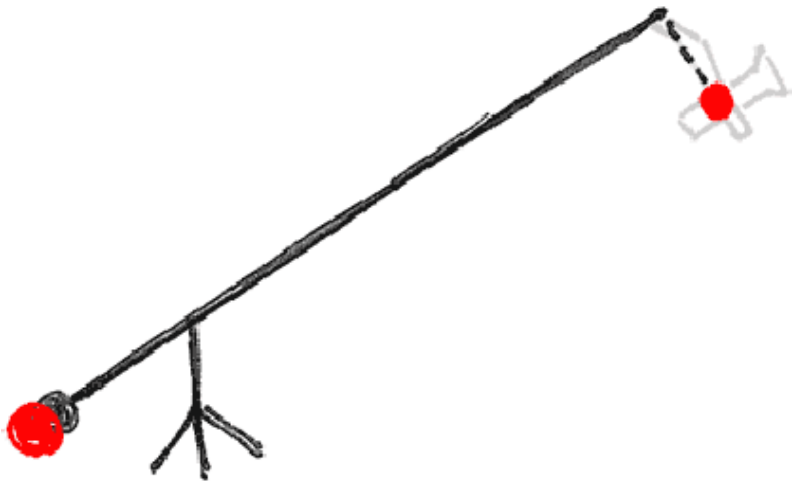
Slika 3.5.3. Dizalica sa glavom i utezima.

Slika 3.5.3. Na taj uravnoteženi model dodamo glavu i kameru. Na drugu stranu biti će potrebno dodati dodatne utege. Svaki uteg koji smo dodali služi isključivo uravnoteženju glave i kamere. Zato su obojani drugom bojom na slici.



Slika 3.5.4. Ruka dizalice sa glavom i utezima zarotirana prema gore.

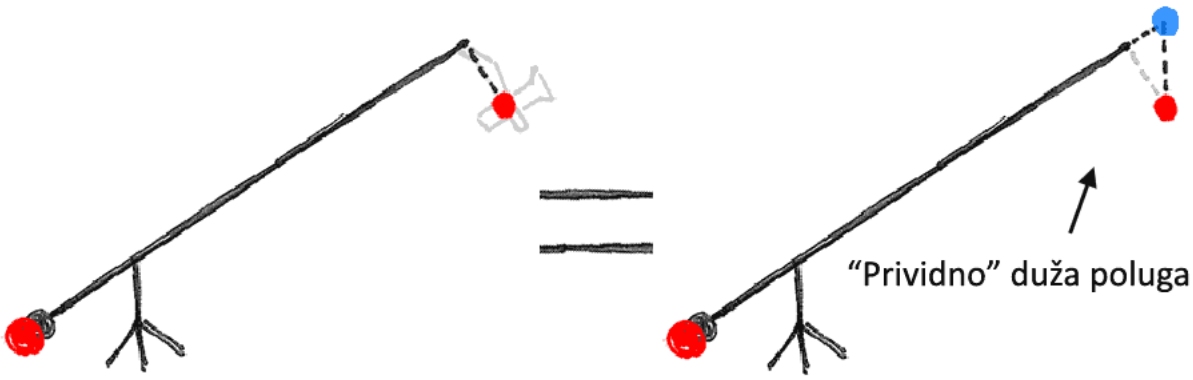
Slika 3.5.4. Ukoliko sada rotiramo ruku vidjeti ćemo da se glava rotira zajedno s njom. Kako to utječe na ravnotežu dizalice?



Slika 3.5.5. Zarotirana ruka s aproksimiranom kamerom.

Slika 3.5.5. Da bi strukturu pojednostavili ucrtamo samo centar mase glave i kamere crvenom točkom i zamislimo da je u njemu koncentrirana sva masa glave i kamere. Tu točku povežemo zamišljenom crtkanom linijom s rukom. Ta je slika istovjetna prethodnoj.

Isto tako ove dvije slike su istovjetne:



Slika 3.5.6. Problem ravnotežnog položaja pri podizanju opterećene ruke.

Slika 3.5.6. Plava točka označava projekciju centra mase glave i kamere na pravac ruke. Sada vidimo da prilikom podizanja prednjeg dijela ruke, dolazi do pomicanja projekcije centra mase kamere i glave, tako da je stvarni krak poluge duži nego što je bio u ravnom položaju. To uzrokuje „prividno“ veći krak poluge prednjeg dijela ruke u odnosu na stražnji, i dovodi do neravnoteže koja uzrokuje padanje prednjeg kraja prema dolje.

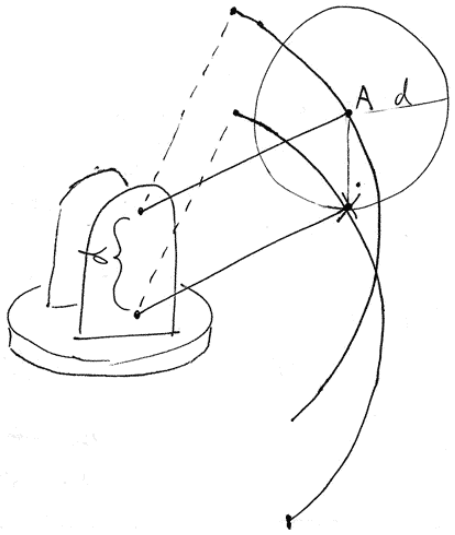
Slika 3.5.7. Spuštanje ruke ispod ravnotežnog položaja.

Slika 3.5.7. Isto tako, ako bi ruku zakrenuli ispod vodoravnog položaja, došlo bi do kraćeg prednjeg kraka poluge, te bi se prednji krak ruke počeo podizati.

Dakle kada bi ruku podignuli i ostavili, ona bi se zbog produljenja poluge pomicala prema dolje dok ne dođe ispod ravnotežnog položaja. Tada bi se zbog skraćivanja poluge počela pomicati prema gore. Tako bi se nastavila njihati oko ravnotežnog položaja, dok se ne bi zaustavila zbog sila trenja.

Opisano njihanje bi se moglo izbjeći tako da se utezi na suprotnom kraku ruke također pomaknu sa pravca koji određuje ruka, ali prema gore, međutim time bi bilo nepraktično rukovati.

Mehanizam koji osigurava konstantnu okomitost robotske glave u odnosu na podlogu rješava sve opisane probleme, zato što se projekcija centra mase glave sa kamerom se bi pomicala po pravcu ruke nego bi uvijek bila na istoj udaljenosti. Okomitost postižemo koristeći pomoćnu cijev. Ukoliko je robotska glava povezana sa stalkom pomoću ruke i pomoćne cijevi, okomitost joj je osigurana na slijedeći način:



Slika 3.5.8. Lukovi svih mogućih položaja glave povezane sa stalkom pomoću ruke i pomoćne cijevi. Okomitost robotske glave je očigledna.

Prvi kraj pomoćne cijevi pričvršćen je na stalak tako da se može rotirati oko spoja. Taj spoj nalazi se ispod spoja ruke sa stalkom na udaljenosti **d** (Slika 3.5.8.).

Drugi kraj pomoćne cijevi pričvršćen je na robotsku glavu, na isti način kao što je spojen i sa stalkom, na udaljenost **d** ispod spoja ruke s glavom.

Spojevi ruke i pomoćne cijevi s robotskom glavom isto su tako podložni rotaciji. Ukoliko nacrtamo lukove koje teoretski mogu zauzeti položaji tih spojeva prilikom rotacije ruke i pomoćne cijevi, dobiti ćemo dva identična luka jedan na udaljenost **d** ispod drugog.

Ukoliko spoj ruke i robotske glave zauzme bilo koji položaj **A** na gornjoj kružnici, spoj pomoćne cijevi i glave mora zauzeti položaj na udaljenosti **d** od toga položaja. Taj spoj mora ležati na donjoj kružnici zbog toga jer je određen rotacijom pomoćne cijevi. Jedina točka koja leži na donjoj kružnici i udaljena je za **d** od točke **A**, nalazi se ispod točke **A** na udaljenosti **d**.

Kako se te dvije točke uvijek nalaze okomito jedna ispod druge, a na njih je pričvršćena glava, slijedi da će robotska glava uvijek biti orijentirana okomito prema podlozi.

Na ovaj način rješava se problem ravnotežnog položaja kamere, te se robotska glava može jednostavnije konstruirati i montirati. Pomoćna cijev također pomaže i u otklanjanju vibracija robotske glave.

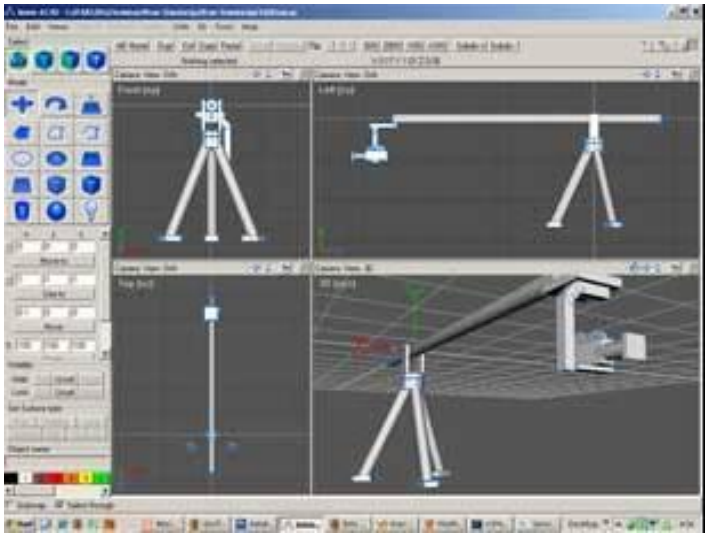
4. Računalni model koji omogućava simulaciju fizički realizirane strukture

Računalni model opisane dizalice u računalu ostvaren je koristeći programski jezik C++ i grafički standard OpenGL.

Na računalnom monitoru prikazuje se trodimenzionalni prikaz dizalice kamere, a korisnik može obavljati slijedeće radnje:

- upravljati rukom, rotirajući je oko stalka – simulacija ručnog pomicanje ruke dizalice u stvarnom svijetu,
- mijenjati orijentaciju kamere na dizalici – predstavlja upravljanje kamerom pomoću upravljačke palice (engl. joystick).
- mijenjati točku iz koje promatra scenu – predstavlja hodanje promatrača oko dizalice,
- koristiti automatsko ciljanje.

4.1. Modeliranje



Slika 4.1.1. Modeliranje dizalice u programu Inivis AC3D

Dizalica je najprije izmodelirana u programu Inivis AC3D koji omogućava spremanje modeliranih objekata u .obj formatu datoteka. Taj format datoteka ima tekstualan i dosta jednostavan opis poligona. Također, takav zapis koristili smo i na laboratorijskim vježbama iz računalne grafike.

4.2. Princip rada računalne aplikacije

Aplikacija učitava objekte rastavljene na pojedine dijelove (stalak, ruka, robotska glava, kamera), te sprema njihov prvotni položaj u matricu koju zovemo originalnom matricom.

Originalna matrica je važna zato jer uvijek čuva prvotnu informaciju o objektima, tako da nema inkrementalne pogreške koja bi mogla nastati zbog nepreciznosti brojeva s posmačnim zarezom, koje koristimo za izračunavanje trenutnih vrhova poligona, prilikom uzastopnih rotacija. Za svaku, pa i najmanju rotaciju, originalnu matricu kopiramo u radnu matricu, koju rotiramo, te ju nakon rotacije vizualno prikazujemo na zaslonu korištenjem OpenGL grafičkog standarda.

Uzevši sve potrebne kutove rotacije koje treba prikazati, problem rotacije rješavamo na slijedeći način:

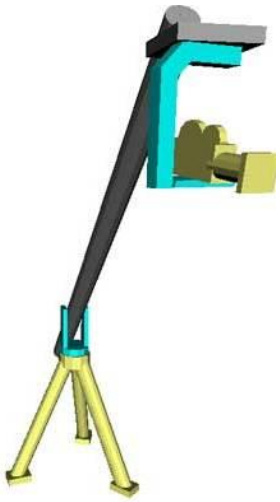
1. robotsku glavu zajedno s kamerom postavimo u ishodište tako da se točka oko koje se kamera rotira u smjeru „gore-dolje“ nalazi u ishodištu,
2. kameru zarotiramo u smjeru „gore-dolje“,
3. glavu (zajedno s kamerom) rotiramo „lijevo-desno“,

4. glavu vratimo na vrh ruke,
5. ruku zajedno sa glavom i kamerom zarotiramo u smjeru „lijevo-desno“,
6. ruku zarotiramo u smjeru „gore-dolje“ i pritom računamo samo položaj glave, s obzirom da je ona uvijek okomita, te da rotacija ruke u smjeru „gore-dolje“ ne rotira glavu kamere.

Dobivene poligone potom prikažemo pomoću OpenGL standarda.

4.3. Korištenje aplikacije

Glavni prozor koji se prikaže pri pokretanju aplikacije sadrži trodimenzionalni prikaz modela dizalice kojom možemo interaktivno upravljati.



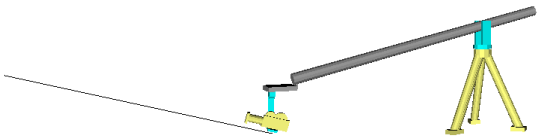
Slika 4.3.1. Prikaz dizalice u glavnom prozoru aplikacije.

Korisničke kontrole:

- Držeći srednji tipku na mišu rotiramo točku iz koje gledamo (očište).
- Tipkama 3 i 4 približavamo i udaljavamo očište.
- Pomicanjem računalnog miša držeći njegovu prvu tipku (obično je na lijevoj strani), rotiramo ruku dizalice.
- Pomicanjem računalnog miša držeći njegovu treću tipku rotiramo glavu sa kamerom.

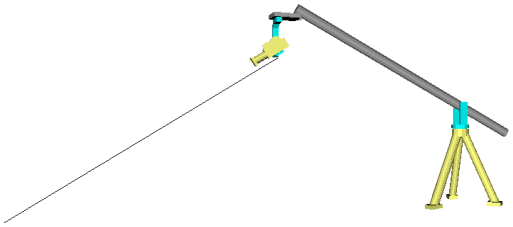
Komande za ciljanje:

- Pomoću navedenih kontrola za upravljanje rukom, namjestimo se na jednu poziciju i stisnemo tipku 1 te na taj način postavimo prvi pravac na kojem leži ciljana točka.



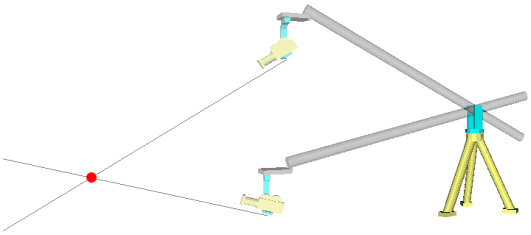
Slika 4.3.2. Odabir prvog pravca.

- Sada pomaknemo ruku na drugu poziciju, zarotiramo kameru, i stisnemo tipku 2, te na taj način odredimo drugi pravac na kojem leži točka.



Slika 4.3.3. Odabir drugog pravca.

Ako pomičemo ruku nakon što smo komandama za ciljanje odredili ciljanu točku, vidjeti ćemo da se kamera automatski zakreće prema zadanoj točci.

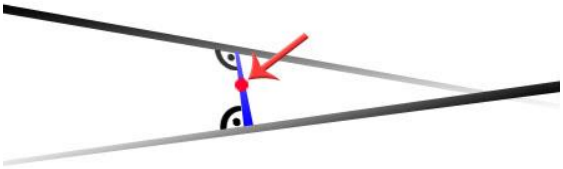


Slika 4.3.4. Ciljana točka prikazana je crvenim krugom na sjecištu dva pravca.

4.4. Izvedbe nekih dijelova aplikacije

Izračun ciljane točke

Slika 4.3.4. iz prethodnog poglavlja prikazuje dva pravca koja se sijeku u prostoru. To sjecište predstavljeno je kao ciljana točka. Međutim, realno je za očekivati da se dva pravca koja korisnik izabere, neće sjeći, nego mimoilaziti. Zato je potrebno pretpostaviti koju točku je korisnik htio naciljati. Najlogičnije je za tu točku uzeti onu, koja je najmanje udaljena od obadva pravca. Matematičkom intuitivnošću možemo pretpostaviti da se radi o točci koja leži na spojnici okomitoj na obadva pravca. Uzeti ćemo točku sa sredine te spojnice.



Slika 4.4.1. Dva mimoilazna pravca u prostoru, te njihova najkraća spojnica (plavom bojom).

Najprije je potrebno postaviti jednadžbe pravaca na slijedeći način:

$$p = \begin{pmatrix} X_0 \\ Y_0 \\ Z_0 \end{pmatrix} + u \bullet \begin{pmatrix} X_{p0} \\ Y_{p0} \\ Z_{p0} \end{pmatrix} \qquad t = \begin{pmatrix} X_1 \\ Y_1 \\ Z_1 \end{pmatrix} + v \bullet \begin{pmatrix} X_{p1} \\ Y_{p1} \\ Z_{p1} \end{pmatrix} \tag{1}$$

Pravci su zapisani u parametarskom obliku (Jednadžbe 1.). Vektor stupci sadrže konstante, a mijenjanjem varijabli **u**, i **v**, „pomičemo“ se po pravcima. Izražavamo formulu za udaljenost dviju točaka koje leže na dva pravca u ovisnosti o njihovim parametrima **u** i **v** (Jednadžba 2.).

$$D = \sqrt{\left(X_0 - X_1 + uX_{p0} - vX_{p1}\right)^2 + \left(Y_0 - Y_1 + uY_{p0} - vY_{p1}\right)^2 + \left(Z_0 - Z_1 + uZ_{p0} - vZ_{p1}\right)^2} \tag{2}$$

Potrebno je pronaći one **u** i **v** parametre za koje udaljenost **D** poprima najmanju vrijednost. Ukoliko jednadžbu 2. parcijalno deriviramo po jednom parametru i izjednačimo ju s nulom (jer tražimo minimum), dobiti ćemo ovisnost toga deriviranog parametra o parametru drugoga pravca (Jednadžba 3.).

$$u = \frac{X_{p0}X_{p1}v + Y_{p0}Y_{p1}v + Z_{p0}Z_{p1}v - X_{p0}X_0 + X_{p0}X_1 - Y_{p0}Y_0 + Y_{p0}Y_1 - Z_{p0}Z_0 + Z_{p0}Z_1}{X_{p0}^{\ 2} + Y_{p0}^{\ 2} + Z_{p0}^{\ 2}} \quad (3)$$

Jednadžba 3. određuje parametar **u** ukoliko je poznat parametar **v**. Parametar **u** izrazili smo pomoću parametra **v** i konstanti. To znači da smo eliminirali jednu nepoznanicu u jednadžbi za najkraću spojnicu (Jednadžba 2.). Uvrštavanjem jednadžbe 3. u jednadžbu 2. dobivamo formulu za dužinu najkraće spojnice pravaca izraženu samo pomoću parametra **v** i konstanti.

$$D = \sqrt{\begin{aligned} &\left(X_0 - X_1 + \frac{X_{p0}X_{p1}v + Y_{p0}Y_{p1}v + Z_{p0}Z_{p1}v - X_{p0}X_0 + X_{p0}X_1 - Y_{p0}Y_0 + Y_{p0}Y_1 - Z_{p0}Z_0 + Z_{p0}Z_1}{X_{p0}^{\ 2} + Y_{p0}^{\ 2} + Z_{p0}^{\ 2}} X_{p0} - vX_{p1} \right)^2 \\ &+ \left(Y_0 - Y_1 + \frac{X_{p0}X_{p1}v + Y_{p0}Y_{p1}v + Z_{p0}Z_{p1}v - X_{p0}X_0 + X_{p0}X_1 - Y_{p0}Y_0 + Y_{p0}Y_1 - Z_{p0}Z_0 + Z_{p0}Z_1}{X_{p0}^{\ 2} + Y_{p0}^{\ 2} + Z_{p0}^{\ 2}} Y_{p0} - vY_{p1} \right)^2 \\ &+ \left(Z_0 - Z_1 + \frac{X_{p0}X_{p1}v + Y_{p0}Y_{p1}v + Z_{p0}Z_{p1}v - X_{p0}X_0 + X_{p0}X_1 - Y_{p0}Y_0 + Y_{p0}Y_1 - Z_{p0}Z_0 + Z_{p0}Z_1}{X_{p0}^{\ 2} + Y_{p0}^{\ 2} + Z_{p0}^{\ 2}} Z_{p0} - vZ_{p1} \right)^2 \end{aligned}} \quad (4)$$

Jednadžba 4. izražava dužinu najkraće spojnice u ovisnosti o parametru **v**. Drugim riječima, za proizvoljnu točku na pravcu **t** pronalazi udaljenost najbliže točke na pravcu **p**.

Ukoliko tu jednadžbu deriviramo po parametru **v** i izjednačimo sa nulom, dobiti ćemo vrijednost parametra **v**, što znači da točka na pravcu **t** više nije proizvoljna, nego ona za koju je udaljenost najmanja. (Jednadžba 5.).

$$v = \frac{-X_{p0}X_0 + X_{p0}X_1 - Y_{p0}Y_0 + Y_{p0}Y_1 - Z_{p0}Z_0 + Z_{p0}Z_1}{X_{p0}^{\ 2} - X_{p0}X_{p1} + Y_{p0}^{\ 2} - Y_{p0}Y_{p1} + Z_{p0}^{\ 2} - Z_{p0}Z_{p1}} \quad (5)$$

Jednadžba 5. omogućava nam da izračunamo parametar **v** pravca **t**, i tako odredimo točku na pravcu **t** koja omeđuje najkraću spojnicu (uvrštavajući vrijednost **v** u jednadžbu 1.).

Potrebno je izračunati i drugu točku spojnice koja leži na pravcu **p**. To ćemo učiniti tako da dobivenu vrijednost **v**, uvrstimo u jednadžbu 3. i izračunamo parametar **u**.

Parametar **u**, uvrstimo u jednadžbu 1. i dobijemo drugu točku koja omeđuje najkraću spojnicu.

Znajući te dvije točke spojnice, nazovimo ih **A** i **B**, možemo izračunati ciljanu točku. Nju računamo računamo kao polovište spojnice (Jednadžba 6.), tj. kao aritmetičku sredinu točaka **A** i **B**.

$$Ciljana\_tocka = \left(\frac{X_A + X_B}{2} \quad \frac{Y_A + Y_B}{2} \quad \frac{Z_A + Z_B}{2} \right) \tag{6}$$

Opisani izračun ciljane točke potrebno je izvesti samo jednom, i to pri računanju ciljane točke. Nakon toga se izvršava algoritam automatskog zakretanja kamere koji će biti opisan u idućem poglavlju.

Izračun automatskog zakretanja kamere

Za izračune koristimo Kartezijev koordinatni sustav. Kao njegovo ishodište uzimamo zglob koji spaja stalak i ruku.

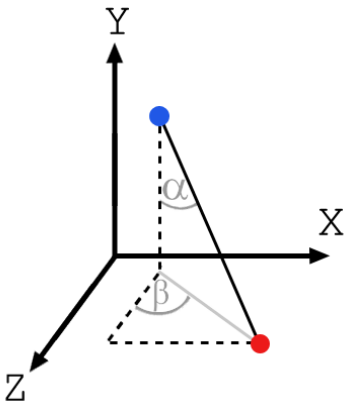
Prvo je potrebno izračunati poziciju robotske glave koja ovisi o rotaciji ruke. Pozicija ciljane točke poznata je već otprije, a njen izračun je opisan u prethodnom poglavlju.

Nakon što saznamo poziciju robotske glave, možemo izračunati potrebnu rotaciju da bi ona bila usmjerena prema ciljanoj točci (Slika 4.4.2.)

Koristimo slijedeće simbole.

X,Y,Z – pozicija robotske glave u Kartezijevom koordinatnom sustavu

X_T,Y_T,Z_T – pozicija ciljane točke u Kartezijevom koordinatnom sustavu

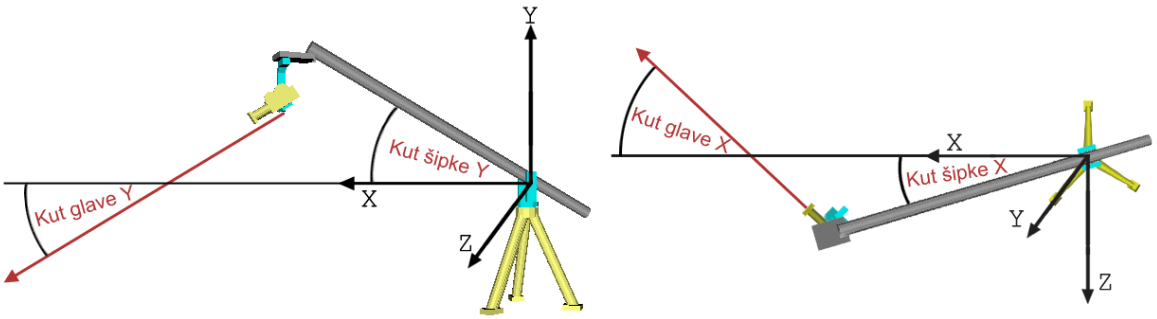


Slika 4.4.2. Prikaz pozicije robotske glave (plavo) i ciljane točke (crveno)

$$\Delta X = X_T - X \qquad \Delta Y = Y_T - Y \qquad \Delta Z = Z_T - Z$$

$$\alpha = \arctan\left(\frac{\Delta Y}{\sqrt{\Delta Z^2 + \Delta X^2}}\right) \qquad \beta = \arctan\left(\frac{\Delta X}{\Delta Z}\right) \tag{7}$$

$$kutglaveY = \alpha \qquad kutglaveX = \beta - kutšipkeX$$

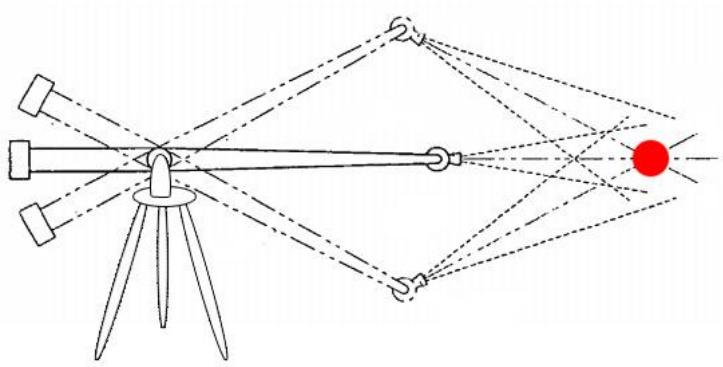


Slika 4.4.3. Prikaz kutova na modelu.

5. Opis specifikacija željenog proizvoda

Proizvod bi trebao udovoljiti slijedećim zahtjevima:

- Potrebno je omogućiti automatizirano okretanje kamere prema objektu snimanja, u ovisnosti o ručnom pomicanju ruke dizalice.
- Pri automatskom okretanju kamere ne smiju se pojavljivati oku vidljivi trzaji kamere, jer bi oni narušili sklad kadra (umjetnički dojam).
- Potrebno je omogućiti memoriranje 10 različitih točaka u prostoru, te jednostavno prebacivanje snimanja s jedne točke na drugu.
- Pri prijelazu sa jedne na drugu točku, postepeno pomicati točku snimanja od točke A do točke B, te ograničiti ubrzanje njenog kretanja (kontinuiranost funkcije ubrzanja) da bi izbjegli vidljive trzaje pri prijelazu.



Slika 5.1. Ilustracija željene funkcionalnosti konačnog proizvoda.

6. Odabir tehnologija

Da bi opisani proizvod uspjeli realizirati potrebno odabrati prikladne tehnologije koje ćemo koristiti pri njegovoj izvedbi. Potrebno je izabrati računalo koje će vršiti aritmetičke proračune, te kontrolirati motore. Osim toga treba izabrati senzore pomoću kojih ćemo očitavati kut pod kojim se nalazi ruka dizalice, te naposljetku način elektroničke kontrole motora. Sve su to područja kojima se mora pomno pristupiti.

6.1. Odabir računala zaduženog za izračune i kontrolu

Da bi odabrali prikladno računalo za izračune, najprije će biti potrebno procijeniti koliko memorijskog prostora i procesorske snage zahtjeva izvršavanje algoritma koji se koristi za automatsko ciljanje. Stoga ćemo ovdje ukratko opisati taj algoritam pseudokodom.

```
CILJANA_TOČKA=ocitaj_ciljanu_točku();
Petlja{
  KUT_RUKE=očitaj_kut_ruke();
  KUT = izračunaj_kut_kamere(KUT_RUKE, CILJANA_TOČKA);
  Pozicioniraj_motore(KUT);
}
```

Izvršavanje jedne ovakve petlje traje oko 100.000 procesorskih taktova na 8-bitnim procesorima koji nemaju koprocesor zadužen za operacije s brojevima s posmačnim zarezom. Memorijski zahtjevi za ovakav kod su reda veličine 10 kilookteta. Da bi se 100.000 taktova izvršilo 30 puta u sekundi, koliko smo predvidjeli da je dovoljno zbog mehaničke vremenske konstante motora od 32 ms, potrebna frekvencija rada procesora je 3Mhz.

Najjednostavnija i najdostupnija računala koja udovoljavaju ovim zahtjevima su 8-bitni mikrokontroleri koji imaju ugrađenu programsku i radnu memoriju na čipu, te razne druge korisne module. Mi smo izabrali 8-bitne Atmel AVR mikrokontrolere zbog njihove široke rasprostranjenosti u industriji i jednostavnosti programiranja.

Frekvencija rada našeg čipa je 16Mhz što znači da će mu izvršavanje ovog koda oduzimati najviše 19% procesorske snage, a to ostavlja prostora za implementaciju mnogih dodatnih popratnih funkcija.

Kratak opis mogućnosti i performansi jednog takvog čipa može se vidjeti na početnoj stranici njegovog pripadajućeg specifikacijskog dokumenta (datasheet) koji se nalazi na kraju dokumenta u Prilogu 1.

6.2. Odabir senzora za očitavanje kuta ruke dizalice

Pri odabiru senzora za očitanje kuta treba voditi računa o slijedećim svojstvima senzora.

- Rezolucija: rezolucija senzora je broj različitih pozicija u krugu koje je senzorom moguće očitati. Najčešće je taj broj potencija broja 2. Na primjer 13-bitni senzor ima 2^{13} podjeljaka, što znači da može razabrati 8192 različite pozicije u krugu.

- Preciznost: preciznost senzora predstavlja ravnomjernost gustoće rasporeda podjeljaka po krugu. Može se tumačiti i kao najveće moguće odstupanje očitanog kuta od stvarnog kuta. Izražava se u stupnjevima. Na primjer, preciznost od ±0.5° na 13-bitnom senzoru znači da taj senzor, iako ima 8192 podjeljka, gdje svaki podjeljak predstavlja ~0.04° može imati grešku očitanja od ±0.5°, što je oko 11 podjeljaka. Svaki senzor je u pravilu monoton, što znači da stvarni pomak senzora u jednu stranu ne može rezultirati očitavanjem pomaka u drugu stranu.

- Tip senzora: senzore rotacije mogli bi podijeliti na apsolutne i inkrementalne senzore. Razlika između ta dva tipa je slijedeća. I jedan i drugi tip imaju neki položaj na kojem očitavaju kut od nula stupnjeva. Apsolutnim senzorima taj je položaj konstantan i određen je mehaničkom konstrukcijom, dok je na inkrementalnim senzorima nulti položaj onaj položaj na kojem se senzor nalazio prilikom dolaska napajanja, ili točnije, u svakom očitanju očitavamo samo njegov pomak od prethodnog mjerenja i onda te pomake programski zbrajamo.

Prva odluka koja se nameće je – trebamo li postaviti apsolutne ili inkrementalne senzore između stalka i ruke.

Ukoliko glava uvijek ne bi bila okomita prema podlozi, bilo bi sasvim svejedno kakve ćemo senzore koristiti, ali budući da ona to je, tada je apsolutnost očitanja kuta bitna za izračune. Osim toga, ako bi recimo, koristeći inkrementalne senzore, naciljali neku točku, i spremili ju u stalnu memoriju, nakon gašenja i paljenja uređaja došlo bi do poništavanja početnog kuta, te uređaj više ne bi ciljao prema istoj točci.

Dakle trebaju nam apsolutni senzori na obadvije osi.

Dalje, potrebno je odlučiti kolika rezolucija senzora nam je potrebna. To izračunamo na slijedeći način. Prvo odredimo koliki najmanji pomak kamere u milimetrima koji želimo biti u mogućnosti registrirati. Dalje, uzevši u obzir opseg ruke, vidimo koliko takav opseg sadrži tih najmanjih dijelova. Broj koji dobijemo biti će donja granica broja podjeljaka u krugu senzora.

Napravimo i taj izračun. Želimo li moći registrirati najmanji pomak od 5mm po svakoj osi, a dužina ruke je 6 metara, tada dobivamo slijedeće podatke:

Opseg = 2 • π • 6 = 37.7 m
Broj podjeljaka = Opseg / 0.005 m = 7540 podjeljaka

Da bi dobili broj potrebnih bitova senzora izračunamo logaritam po bazi dva od broja podjeljaka i dobiveni rezultat zaokružimo na prvi veći broj.

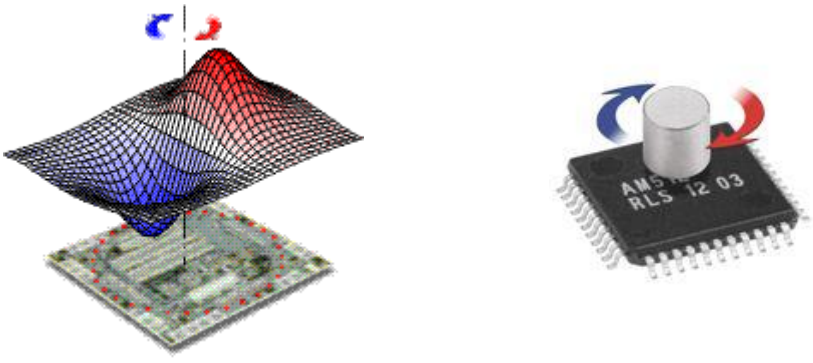
$$\lceil \mathrm{Log}_2\,(7540) \rceil = \lceil 12.88 \rceil = 13\,.$$

Znači potreban nam je 13 bitni apsolutni senzor. Istraživanjem tržišta došli smo do slijedećeg senzora kao optimalnog rješenja:

RM22 je magnetski senzor u obliku čipa koji ima kružno postavljene hal-sonde na površini silicija. Funkcija hal-sondi je iz gustoće magnetskog polja očitati kut pod kojim je okrenut cilindrično polarizirani magnet koji se nalazi iznad čipa (Slika 6.2.1.).

Senzor ima slijedeće karakteristike:

- 13-bitnu rezoluciju
- Preciznost od ±0.5°
- Odličnu imunost na smetnje, te serijsku komunikaciju (RS422A) pogodnu za rad sa mikrokontrolerima.
- IP68 zaštitu kućišta (kompletna zaštita od prašine, kontakta, te uranjanja u vodu) koja je korisna jer se dizalica koristi na otvorenim terenima, te postoji mogućnost prokišnjavanja i utjecaja vlage.
- Između okretnog dijela (magneta) i statičnog (čipa) ne mora biti kontakta, što je olakotna okolnost pri ugradnji.



Slika 6.2.1. Čip s halovim sondama postavljenim u krug i ilustracija magnetskog polja.

6.3. Odabir senzora za očitavanje kuta kamere

Na glavi kamere koristimo motore za ugrađenim inkrementalnim senzorima, koji nam služe kao pomoć pri kontroli brzine. Iste te inkrementalne senzore možemo koristiti i za određivanje kuta pod kojim se kamera trenutno nalazi. Da bi ih mogli koristiti u tu svrhu, prilikom paljenja uređaja potrebno je robotsku glavu usmjeriti ravno i podesiti nulti položaj (kut od nula stupnjeva).

Inkrementalni senzori vrlo su jednostavne izvedbe. Sastoje se od okruglog 7-polnog magneta, te dva hall senzora pod kutom od 90°. U jednom krugu svaki od tih hall senzora napravi 7 impulsa, što, ukoliko brojimo padajuće i rastuće bridove daje rezoluciju od 28 pozicija u jednom okretu osovine motora. Čini se malo, ali ako uzmemo u obzir redukciju prijenosa motora od 1:700, to nam daje 19600 impulsa po okretu glave, što je veća preciznost nego RM22 koji očitava kut ruke.

Novija verzija uređaja imati će, uz inkrementalne, i apsolutne senzore ugrađene i na glavi. Funkcija apsolutnih senzora biti će automatsko podešavanje nultog kuta, i na njima neće biti potrebna neka visoka rezolucija, dok će funkcija inkrementalnih biti poboljšanje rezolucije te kontrola brzina.

Razlog takve izvedbe je taj da prilikom paljenja uređaja ne moramo podešavati nulti kut robotske glave. Mogli bi se zapitati, pa zašto onda ne koristimo samo apsolutne senzore na glavi?

Razlog leži u ovim činjenicama: Senzorima rotacije cijena je uglavnom proporcionalna njihovoj rezoluciji, a apsolutne senzore nema smisla montirati na osovinu motora, jer oni raspoznaju apsolutni kut samo unutar jednog kruga. Dakle ukoliko imamo prijenosni omjer od 1:500 na motoru tada su nam za istu rezoluciju potrebni 500 puta precizniji apsolutni senzori od inkrementalnih. Radi kvalitetne servo kontrole motora, potrebno je na motorima imati ugrađene inkrementalne senzore s kojih je najlakše i najbrže očitavati brzinu vrtnje motora. Kad već imamo takve inkrementalne senzore, koji imaju jako dobru krajnju preciznost (prvenstveno iz razloga što su montirani na osovinu motora), možemo ih koristiti i za određivanje kuta robotske glave, a jedini problem ostaje određivanje njenog nultog kuta pri paljenju uređaja. Taj problem riješimo tako da na glavu stavimo neke jednostavne apsolutne senzore skromne rezolucije, koji će služiti samo tome, da pri paljenju odrede nulti kut glave.

6.4. Implementacija proizvoda

Za implementaciju proizvoda povezano je 7 čipova koji međusobno komuniciraju i imaju podijeljene zadaće. Svi čipovi su, radi jednostavnosti, iste arhitekture: 8-bitni Atmel AVR mikrokontroleri.

Svakom čipu radi jednostavnosti nedjenuti ćemo kratko ime koje jednostavno dočarava njegovu funkciju. Njihova imena su: “Glavni”, “MotorX”, “MotorY”, “kamera kontrola”, “Focus/iris”, “Tipke” i “Led”.

Najprije ćemo ukratko opisati njihove funkcije:

- MotorX i MotorY:

  Kontroliraju brzinu i poziciju motora pojedine osi na robotskoj glavi. Na njih su spojeni inkrementalni senzori s osovina motora pomoću kojih je određen kut robotske glave. Od “glavnog” čipa primaju pozicije na koje bi trebalo pozicionirati motore, te na temelju toga, koristeći PID algoritam, koji ćemo opisati naknadno, procjenjuju brzinu i računaju napon koji će aplicirati na motore.

- Kamera kontrola:

  Kontrolira zoom i ostale kontrole na gotovo svim vrstama kamera. Kontrola kamere prima zadaće od “glavnog” čipa i izvršava ih.

- Fokus/iris:

  Kontrolira fokus i iris na profesionalnim lećama za kamere. Ovaj čip povezan je s dva servo motora montirana na leću kamere. Servo motori zauzimaju željeni kut koji korenspondira sa fokusnom dubinom, te svjetloćom slike. Komunikacija između ovog čipa, i čipova ugrađenih na servo motore odvija se vremenskom dužinom pozitivnog električnog impulsa. Svakih 20 ms ovaj čip šalje impuls dužine od 1 ms do 2 ms. Ukoliko impuls traje 1 ms motor će zauzeti kut od 0°, a ukoliko traje 2ms tada će zauzeti maksimalni kut od 240°. Naravno, moguće je zauzeti i međukutove, npr. želimo li motor pozicionirati na sredinu, tada pošaljemo impuls dužine 1.5ms. Ovaj čip komande također prima od “glavnog” čipa.

- Tipke:

  Budući da na korisničkoj konzoli imamo 40 tipki (Slika 6.4.1.), jedan čip zadužili smo da obavlja zadaću samo očitavanja tipki. Iako bi i glavni čip mogao obavljati ovu funkciju koja za razliku od ostalih nije aplikacija u stvarnom vremenu, poseban čip dodan je zbog nedostatka pinova na glavnome čipu.

Slika 6.4.1. Korisnička konzola.

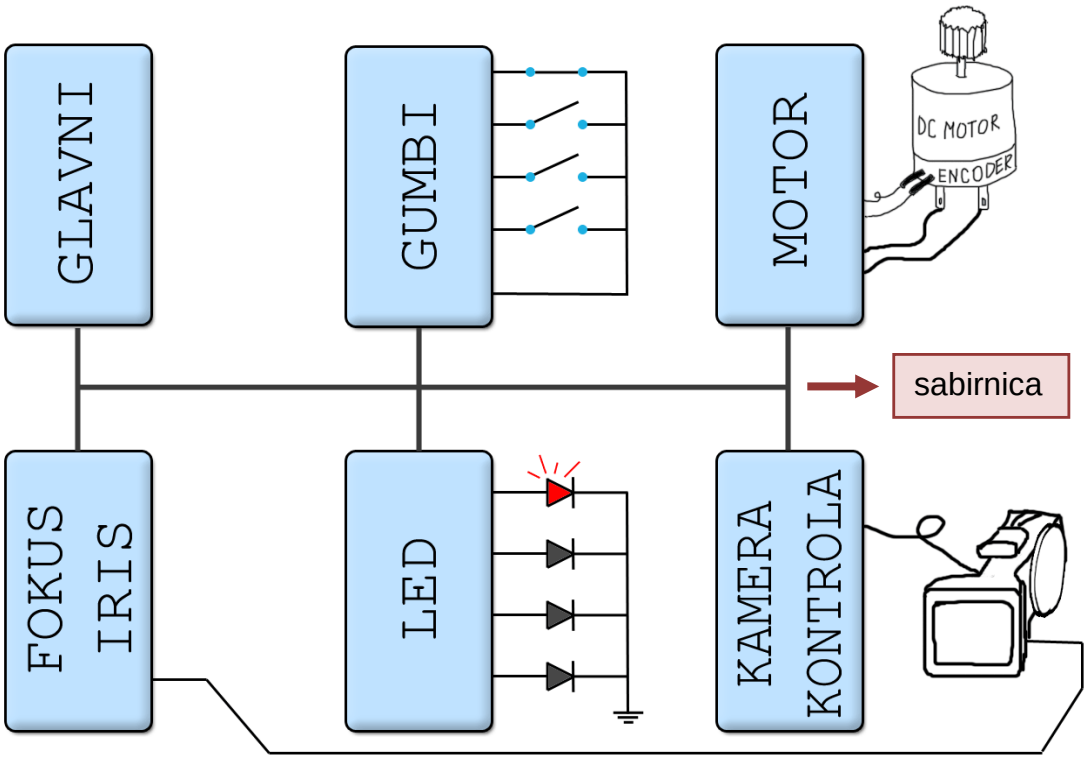
- Ledice:

  svaka tipka ima svoju pripadajuću svjetleću diodu, koju glavni čip treba biti u mogućnosti upaliti ili ugasiti. Kako je za svaku svjetleću diodu potreban jedan izlazni pin, za ovu namjenu također smo dodali jedan odvojen čip. “Glavni” čip ovome čipu šalje naredbe. Iako za ovu namjenu postoje posebni multipleksorski čipovi, radi boljeg uklapanja u aplikaciju, te spajanja na istu podatkovnu sabirnicu i tu smo koristili AVR miktrokontroler, koji trenutno na tržištu čak nije niti skuplji od multipleksorskih čipova namijenjenih za paljenje ledica.

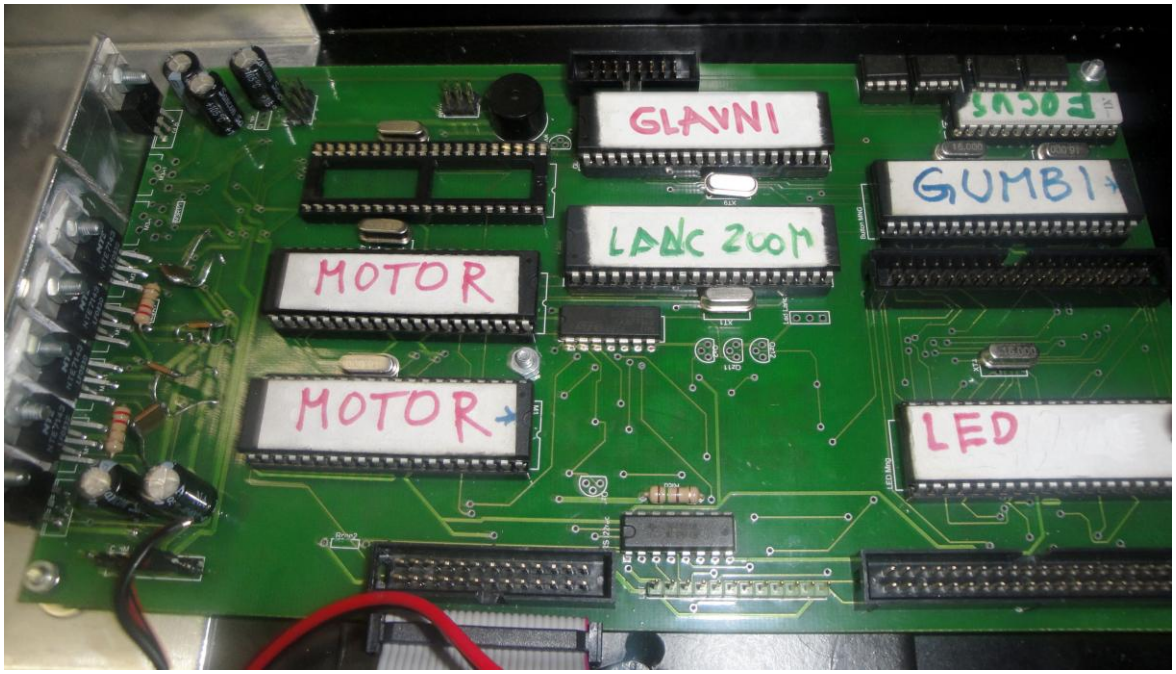
- Glavni:

  Ovaj čip je mozak cijelog sustava. On očitava senzore rotacije, od čipa “Tipke” očitava koja je tipka stisnuta, te koristeći deterministički konačni automat, određuje što će se ispisati na zaslonu, koje će se svjetleće diode upaliti, te kakvo će biti ponašanje sustava. Također, određuje koje kontrole će slati kameri preko čipa „kamera kontrola“ i kojom brzinom će se kretati motori komuniciranjem sa čipom „Motor X/Y“. Obavlja i trigonometrijske izračune potrebne za automatsko ciljanje zadane točke. Trolinijskom sabirnicom povezan je sa svakim čipom na štampanoj pločici, te s njima komunicira dvosmjernom komunikacijom koristeći ugrađeno SPI sučelje (engl. Serial peripheral interface), što je najbrži oblik komunikacije između korištenih mikrokontrolera.

Svi opisani čipovi nalaze se na istoj štampanoj pločici.



Slika 6.4.2. Pojednostavljena shema povezivanja i funkcionalnosti čipova na štampanoj pločici.



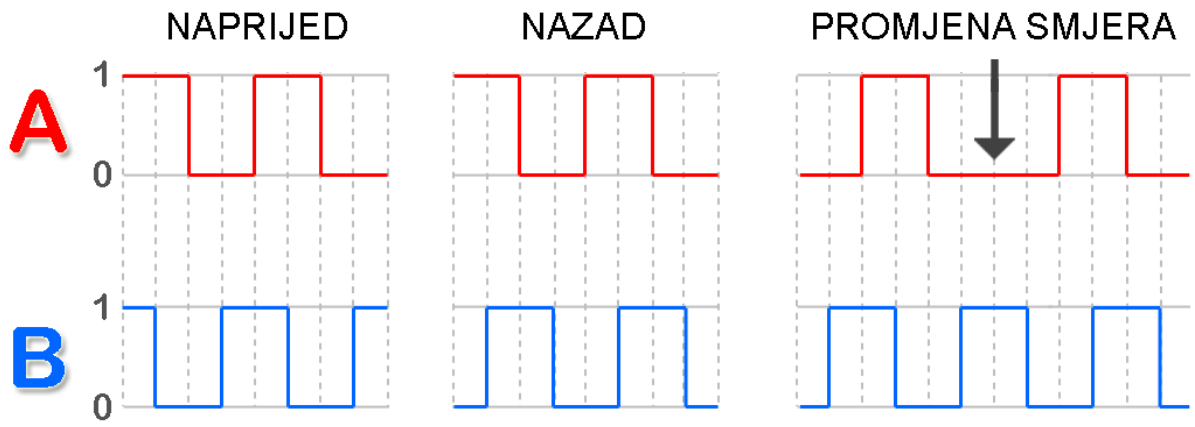
Slika 6.4.3. Fotografija štampane pločice koja implementira shemu sa slike 6.4.1.

7. Implementacija pojedinih modula

7.1. Implementacija čipa za kontrolu brzine – „Motor“

Brojanje impulsa

Ranije smo spomenuli da je na motoru postavljen inkrementalni senzor rotacije. Ukratko ćemo opisati izlaze takvoga senzora. Senzor ima dva izlaza A i B, koji imaju dva moguća stanja napona. Svaka promjena napona na tim izlazima, korespondira rotaciji osovine motora. Kao što prikazuje Slika 7.1.1., mi smo te naponske razine označili sa „1“ i „0“. Jedna električka perioda podrazumijeva 2 promjene napona na svakom kanalu. To je ukupno 4 naponske promjene u jednoj električkoj rotaciji. Fizička rotacija osovine motora za 360° sadrži 7 električkih rotacija, tj. 28 promjena napona. Imajući to na umu možemo reći da naš inkrementalni senzor ima rezoluciju od 28 podjeljaka u mehaničkom krugu senzora.



Slika 7.1.1. Značenje funkcijskih oblika dvokanalnog inkrementalnog senzora rotacije.

Da bi pomoću mikrokontrolera mogli raspoznati 4 pozicije unutar jedne električke rotacije potrebno je biti u mogućnosti brzo reagirati na promjenu napona na ulaznom pinu.

Naš Atmel AVR ATMega16 mikrokontroler, ima dva specijalna pina (*engl. external interrupt pin*) koji mogu raspoznati rastuće i padajuće bridove napona, te pri dolasku jednoga od njih aktivirati prekidni potprogram. Te pinove koristimo tako da na njih spojimo dva kanala senzora. Prekidni program koji broji impulse je vrlo jednostavan. Napisati ćemo njegov pseudokod:

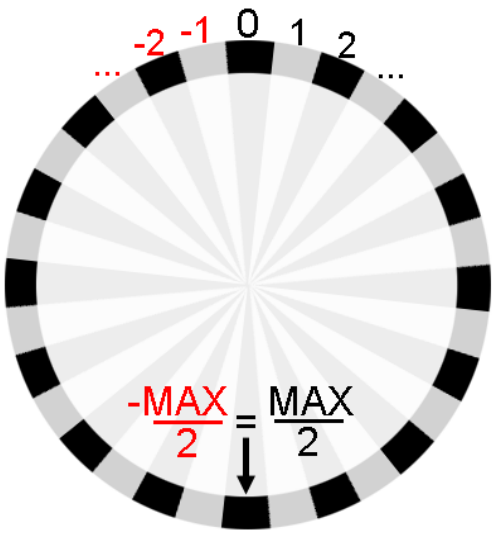
```
Ako je (stariKanalA == noviKanalB) tada  {Brojac++}
                                   inače {Brojac--}

stariKanalA=noviKanalA;
Normaliziraj(Brojac);
```

Objašnjenje pseudokoda: Pogledavši sliku 7.1.1. vidjeti ćemo da ukoliko kanal 'B' pomaknemo za četvrtinu periode „u desno“, tada je on ili jednak kanalu 'A', ili inverzan. Ukoliko su jednaki tada se motor vrti naprijed, a ukoliko su inverzni tada se radi o rotaciji u nazad. Ovime smo objasnili način brojanja koji je opisan u prva tri retka pseudokoda. Preostalo je još objasniti funkciju normaliziraj.

Zamislimo da se motor konstantno vrti u jednu stranu. Prije ili poslije, varijabla brojač došla bi do neke svoje najveće vrijednosti, i nakon toga, ukoliko bi ju još povećavali, dogodila bi se neka vrsta greške. Zato je potrebno ograničiti vrijednosti varijable brojač na neke koje su unutar mogućih vrijednosti programskih varijabli. To ćemo najbolje učiniti, ako izračunamo koliko impulsa sadrži jedan kompletni okret robotske glave, te tu vrijednost uzmemo za najveću vrijednost varijable brojač. Izračunajmo broj impulsa u jednom okretu glave:

$$impulsa\_u\_rotaciji\_osovine\_motora = 7 \bullet 4 = 28$$
$$prijenosni\_omjer = 648$$
$$impulsa\_u\_rotaciji\_glave = 18144$$

Ukoliko koristimo i negativne brojeve, tada za prikaz možemo upotrijebiti brojevnu kružnicu (Slika 7.1.2.).



Slika 7.1.2. Način brojanja impulsa prezentiran brojevnom kružnicom.

Pseudokod funkcije normaliziraj:

```
Funkcija Normaliziraj(brojac)
   Ako (brojac > MAX/2) vrati brojac - MAX;
   Ako (brojac < MAX/2) vrati brojac + MAX;
   vrati brojac
```

Kontrola kuta rotacije i brzine motora – PID kontroler

Iz prethodnog poglavlja vidjeli smo da možemo brojati impulse enkodera i tako znati relativnu poziciju na kojoj se motor nalazi. Ukoliko bi, osim impulsa, mogli mjeriti i vrijeme proteklo između njih, tada bi bili u mogućnosti računati i kutnu brzinu kojom se okreće motor u datom trenutku. Čip koji koristimo za kontrolu ima tu mogućnost.

Tu brzinu računamo na slijedeći način:

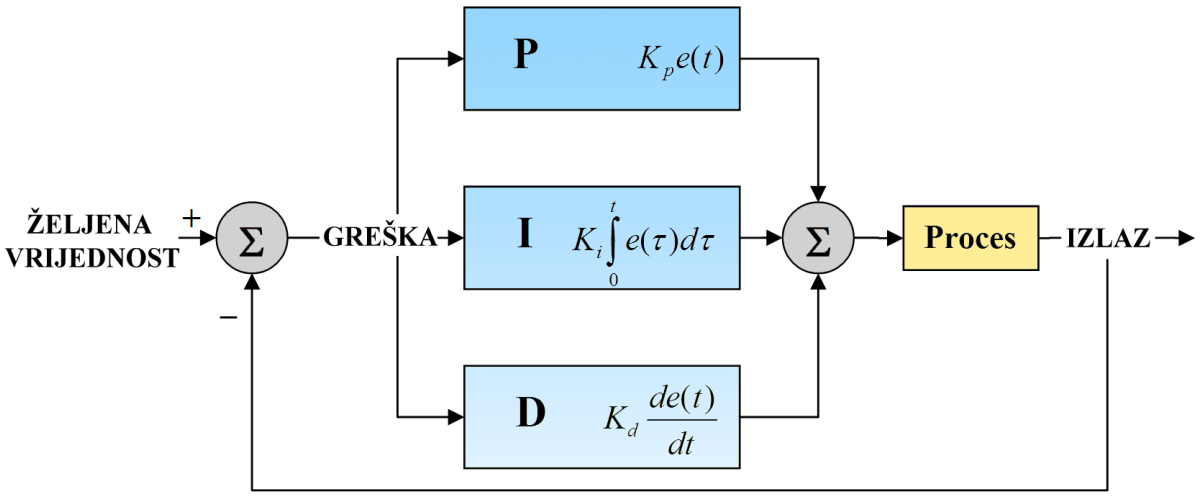
$$v = \frac{broj\_impulsa}{proteklo\_vrijeme}$$

Na robotskoj glavi koristimo istosmjerne elektromotore, čija kontrola je relativno jednostavna. Brzina motora direktno je proporcionalna naponu na motoru, a obrnuto proporcionalna opterećenju motora.

Želimo li ostvariti kvalitetnu kontrolu brzine, tako da brzina bude neovisna o opterećenju motora (koje varira ovisno o težini kamere) i drugim faktorima, tada moramo koristiti kontrolu s povratnom vezom (engl*. closed-loop control*).

Najraširenij algoritam u industriji koji implementira tu vrstu kontrole je proporcionalno-integralno-derivativni kontroler, popularno zvani PID kontroler.

Zadaća tog kontrolera je maksimalno približiti željenu vrijednost procesne varijable (u našem slučaju kut rotacije motora) i njenu stvarnu (izmjerenu) vrijednost. Drugim riječima, minimizirati pogrešku.



Slika 7.1.3. Princip rada PID kontrolera.

Slika 7.1.3. ilustrira princip rada PID kontrolera. Naša procesna varijabla je kut rotacije motora. Sustav konstantno mjeri kut rotacije motora, te računa grešku označenu slovom *e* (skraćeno od engl. error) te na temelju te greške računa tri komponente korektivnog napona. Nabrojati ćemo i objasniti te tri komponente.

Proporcionalna komponenta – P

P komponenta najvažnija je od sve tri komponente u PID kontroleru jer svojim iznosom najviše doprinosi ukupnoj vrijednosti procesne varijable. Postoje PID kontroleri koji koriste samo ovu komponentu, a takvi se nazivaju P kontroleri.

Komponentu računamo na slijedeći način:

$P = K_p \bullet e(t)$

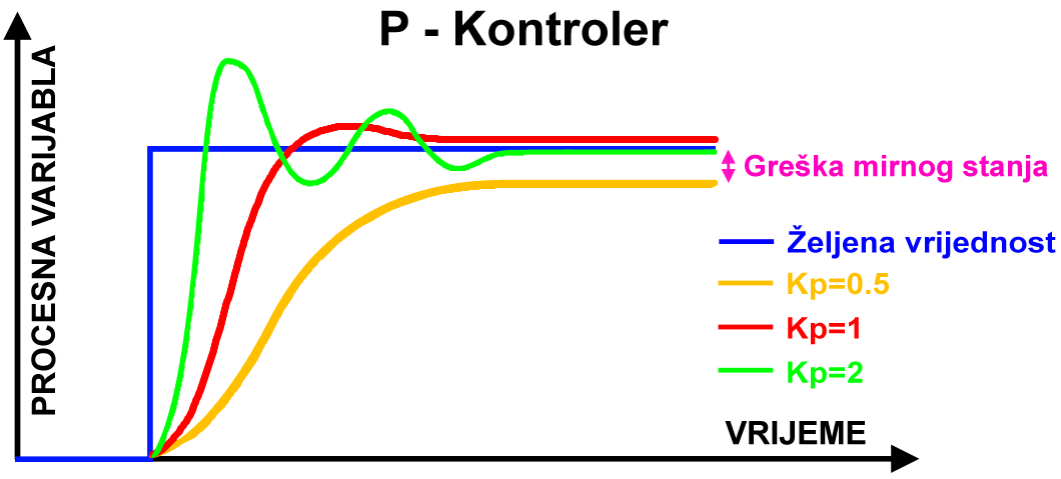
P: Proporcionalna komponenta

K_p: Proporcionalno pojačanje

e: Greška = Željena pozicija − stvarna pozicija

t: Trenutno vrijeme

P komponenta linearno je proporcionalna o iznosu greške. Njen utjecaj možemo pojačavati ili smanjivati promjenom parametra pojačanja Kp. Prevelike vrijednosti parametra Kp rezultiraju velikim promjenama na izlazu za danu promjenu u grešci, što naposlijetku rezultira nestabilnošću sustava i oscilacijama. Ukoliko je Kp premali, tada je reakcija sustava preslaba i često prespora. Pri odabiru vrijednosti Kp, često se primjenjuje pravilo da pronađemo iznos pojačanja na kojemu sustav postaje nestabilan, a tada Kp postavimo na polovicu te vrijednosti.



Slika 7.1.4. Reakcija na impulsnu pobudu ovisna o različitim vrijednostima Kp.

Integralna komponenta – I

Integralna komponenta druga je po važnosti od tri komponente. PID kontroleri koji koriste samo P i I komponentu su najčešći u primjeni, te se nazivaju PI kontroleri.

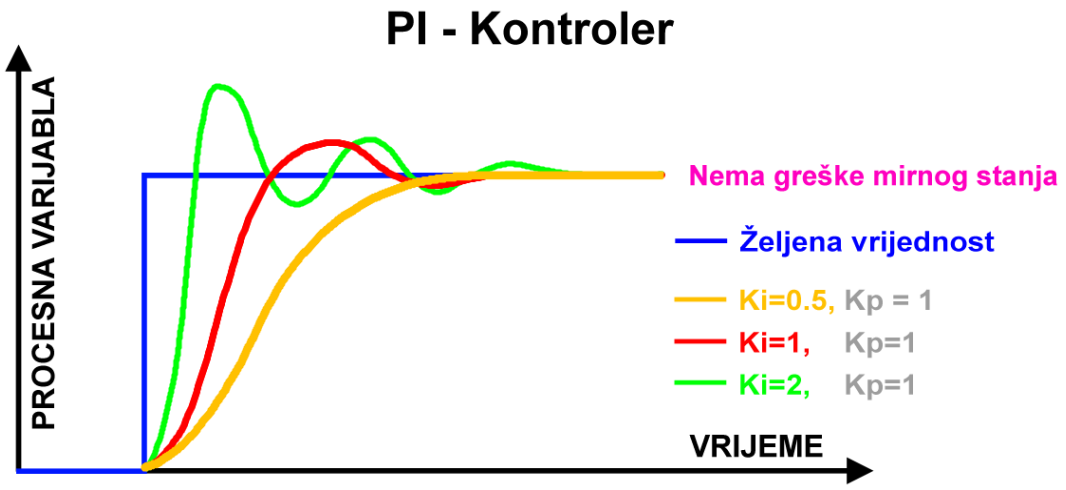
Računamo ju ovako:

$$I = K_i \bullet \int_0^t e(\tau) \bullet d\tau$$

I: Integralna komponenta
K_i: Integralno pojačanje
e: Greška = Željena pozicija − stvarna pozicija
t: Trenutno vrijeme
τ: integracijska varijabla

Doprinos integralne komponente proporcionalan je kako veličini pogreške, tako i njenom trajanju. Zbrajanjem greški tokom vremena dobivamo akumuliranu grešku koja je trebala biti ispravljena u prošlosti. Ta akumulirana greška množi se konstantom integralnog pojačanja *Ki*, a mijenjanjem njenog iznosa mijenjamo ukupni doprinos integralne komponente u PID kontroleru.

Integralna komponenta ubrzava postizanje željenog iznosa procesne varijable, te eliminira stalnu grešku mirnog stanja koja se pojavljuje u jednostavnijim P kontrolerima. Kako integralna komponenta reagira na akumulirane greške iz prošlosti, vrlo je vjerojatno da će uzrokovati oscilacije oko željene vrijednosti procesne varijable zbog čega je bitno pažljivo podesiti vrijednost pojačanja *Ki*.



Slika 7.1.5. Reakcija na impulsnu pobudu ovisna o različitim vrijednostima K_i.

Derivativna komponenta – D

Derivativna komponenta predstavlja promjenu pogreške kroz vrijeme (prva derivacija greške u odnosu na vrijeme).

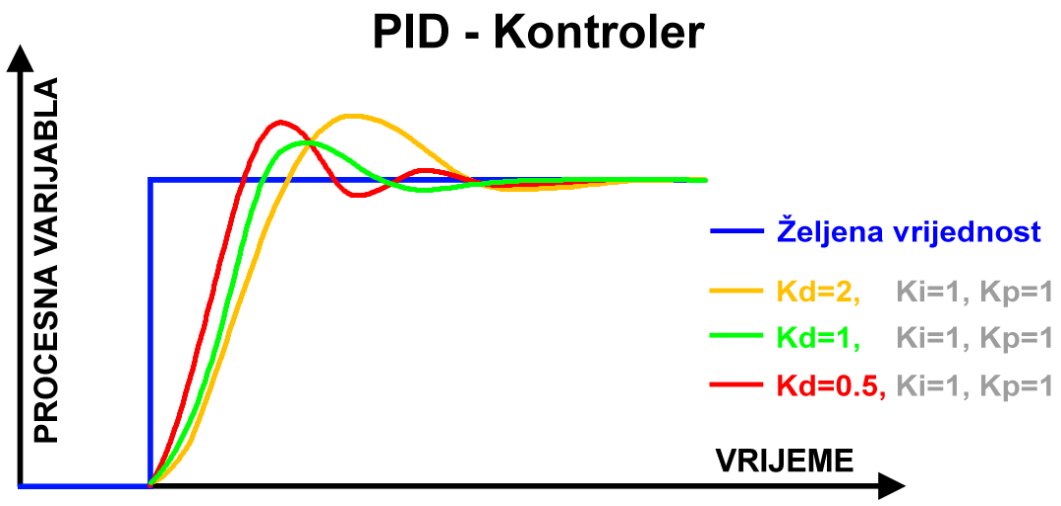
Računa se na slijedeći način:

$$D_{IZLAZ} = K_d \bullet \frac{d}{dt} e(t)$$

D: Derivativna komponenta
K_d: Derivativno pojačanje
e: Greška = Željena pozicija − stvarna pozicija
t: Trenutno vrijeme

Iako derivativna komponenta usporava postizanje željene vrijednosti, ona se koristi u svrhu reduciranja prekovrijednosti nastalih zbog integralne komponente, te poboljšanja stabilnosti procesa.

Deriviranje signala pojačava šum pa je ova komponenta izrazito osjetljiva na šum u grešci, te proces može postati astabilan ako je šum i derivativno pojačanje dovoljno veliko.



Slika 7.1.6. Reakcija na impulsnu pobudu ovisna o različitim vrijednostima Kd.

Nakon što smo opisali tri komponente PID kontrolera, možemo napisati i implementaciju našeg kontrolera kojeg koristimo.

Pseudokod PID kontrolera:

```
Petlja {
  Greska = Zeljena_Pozicija - Stvarna_pozicija;

  P  =  Greska;
  I +=  Greska;
  D  =  Greska - Prijasnja_Greska;

  Prijasnja_Greska = Greska;

  Napon = P * Kp + I * Ki + D * Kd;
}
```

Procesna varijabla PID kontrolera je pozicija na kojoj se nalazi motor. Stvarnu poziciju mjerimo pomoću impulsa koje dobivamo senzora rotacije, a željenu poziciju dobivamo iz „Glavnog“ čipa.

Želimo li da se motor vrti konstantnom brzinom, potrebno je iz „Glavnog“ čipa, stalno uvećavati željeni kut motora za isti iznos.

Da bi izbjegli oscilacije oko željenog kuta potrebno je vrijednosti pojačanja navedene tri komponente uskladiti tako da pri impulsnoj pobudi nema oscilacija.

Druga nepoželjna stvar koju možemo zamisliti je slijedeća: Budući da iz „glavnog“ čipa šaljemo željeni kut motora, u diskretnim pomacima, moguće je i da će se motor pomicati na taj način: zauzme poziciju, pa stane, i tako stalno. Da bi to izbjegli, potrebno je iz „Glavnog“ čipa dovoljno učestalo slati novi željeni kut, tako da se motor konstantno zakreće prema željenom kutu. Poželjno je da taj vremenski period bude reda veličine vremenske konstante motora (vrijeme potrebno da motor reagira na promjenu napona).

Gdje je varijabla koja označava proteklo vrijeme?

Iako čip ima sklop za mjerenje vremena, mi ga ne koristimo, jer novu, željenu poziciju primamo 30 puta u sekundi, što znači i 30 prolazaka kroz petlju u sekundi. To znači da bi varijabla proteklog vremena bila uvijek ista, i to oko 33ms. Budući da je proteklo vrijeme konstantno, tada bi grešku uvijek množili ili dijelili s konstantom, što znači da nam trebaju samo konstante pojačanja. Zato smo u pseudokodu varijablu vrijeme zanemarili. Također, na taj način postigli smo značajno ubrzanje algoritma jer smo smanjili broj operacija nad brojevima s posmačnim zarezom.

Elektronika koja čipu „Motor“ omogućuje kontrolu istosmjernih motora

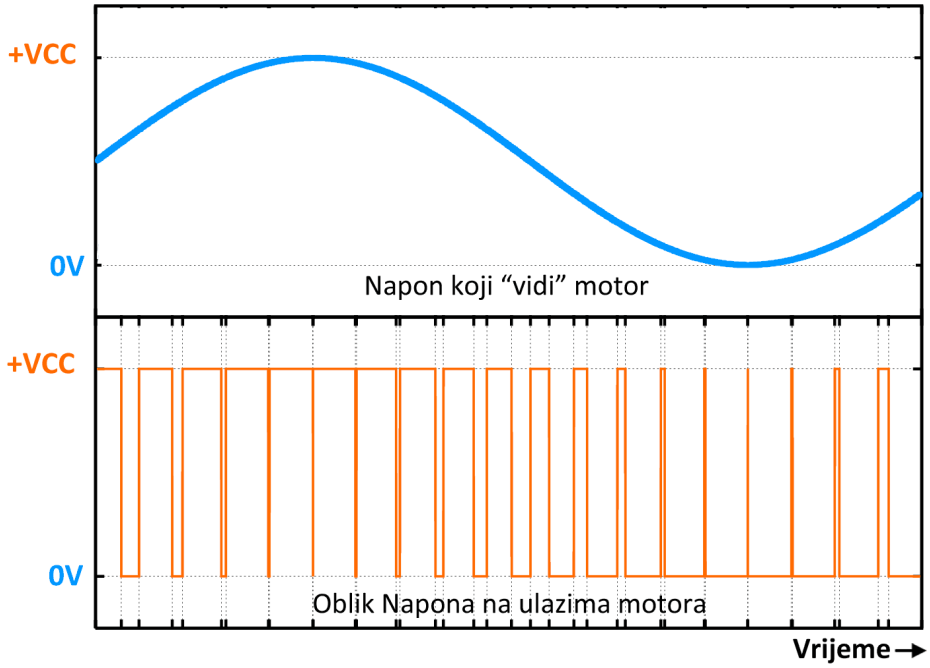
Istosmjerni motor upravljan je električnim naponom između svoja dva ulaza. Ukoliko je taj napon pozitivan, motor se vrti u jednu stranu, a ukoliko je negativan, tada se vrti u suprotnu stranu.

Ovisno o visini tog napona motor se vrti brže ili sporije, što znači da ukoliko motor spojimo na iznos 20% od nazivnog napona motora, da će se tada motor vrtiti 5 puta sporije nego što bi se vrtio kada bi ga spojili na nazivni napon.

Naša elektronika ima za zadatak izlaze iz mikrokontrolera koji mogu zauzeti diskretne naponske razine od 0V i 5V, transformirati na potrebne ulazne naponske razine motora tako da se na motoru ulazne vrijednosti kreću od -24V do +24V.

Postoje razne metode kontrole istosmjernih motora mikrokontrolerima, a jedna od najzastupljenijih je kontrola širinom impulsa koja radi na slijedeći način:

Unutar vrlo kratke vremenske periode spajamo motor na puni napon napajanja i odspajamo ga sa njega. Brzina motora proporcionalna je omjeru vremena u kojem je motor bio spojen na napajanje i ukupnom trajanju ciklusa. Frekvencija ponavljanja takvih impulsa je reda veličine 10 kHz, što je dovoljno velik iznos, da bi motor, induktivitetom svojih zavojnica „ispeglao“ struju, te „vidio“ te digitalne naponske impulse, kao analogni napon.

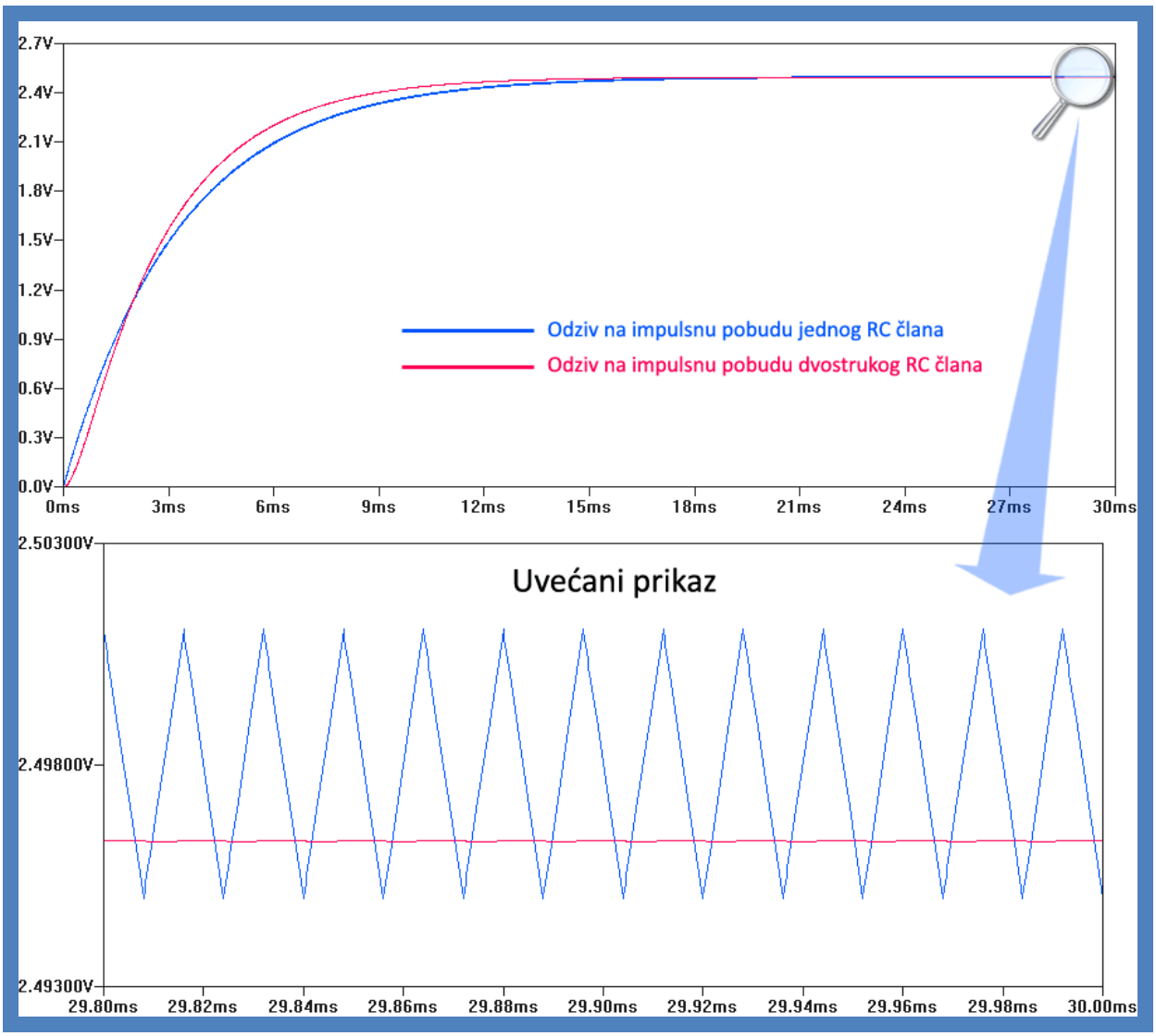


Slika 7.1.7. Pojednostavljen prikaz kontrole motora širinom naponskog impulsa.

Ipak, mi nismo koristili taj način zato što bi visoke frekvencije paljenja i gašenja napajanja, sa značajnim iznosima struja (reda veličine 1 A), uzrokovale smetnje na video signalu, te ostalim signalnim kablovima koji su zajedno provedeni po ruci dizalice. Međutim, koristili smo metodu širine impulsa koje smo RC filterima prvo

pretvorili u analogni napon da bi eliminirali putovanje visokih frekvencija kablovima. Izlaz iz RC filtera spojili smo na pojačalo napona i snage, jer struja iz mikrokontrolera ne smije preći iznos od 20mA, a motori zahtjevaju i do 2A.

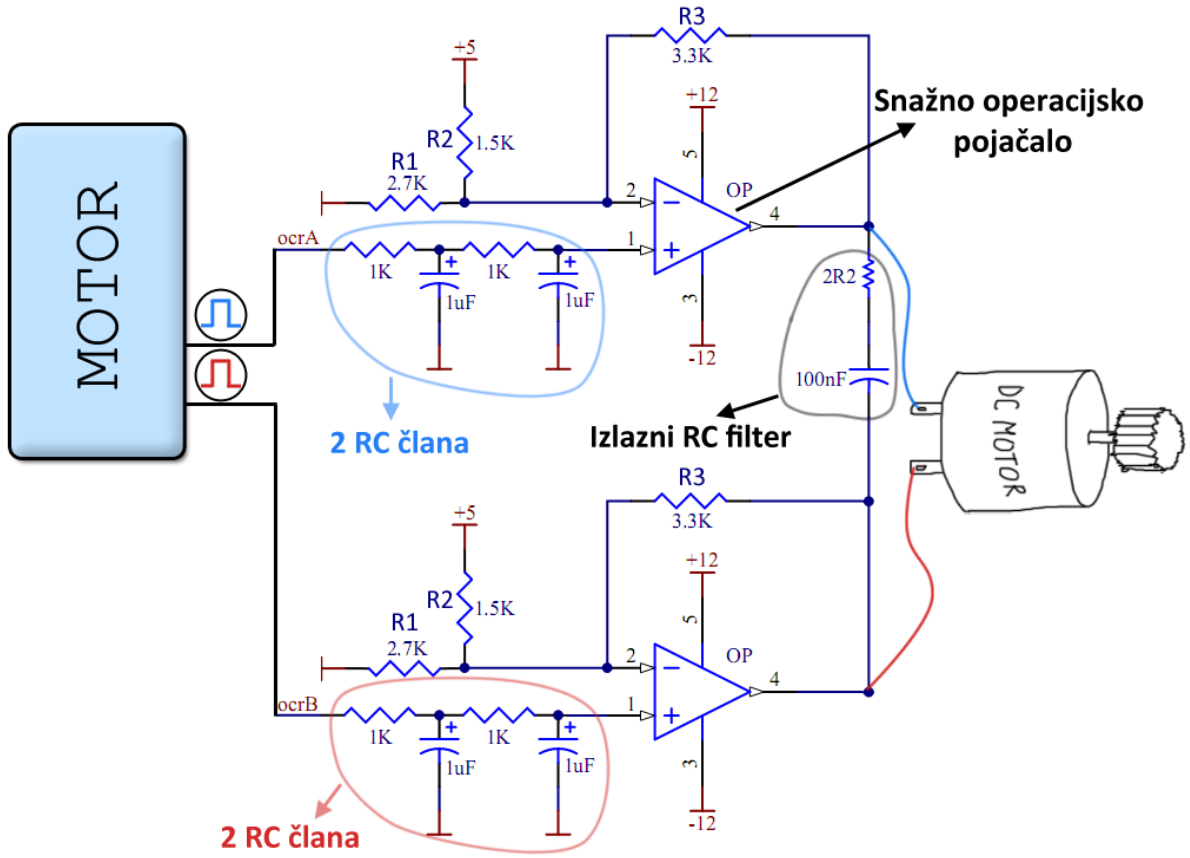
Za filtriranje pravokutnog signala iz mikrokontrolera RC filterom, koristili smo dva RC člana umjesto jednog. Netko bi se mogao zapitati zašto jednostavno nismo povećali vremensku konstantu RC članu da eliminiramo smetnje. Međutim, koristeći dva RC člana, dobivamo puno bolje filtriranje, bez značajnog povećanja vremenske konstante. Slika 7.1.8. prikazuje usporedbu dva različita RC filtera sa istom vremenskom konstantom. Prvi filter (plava boja na slici) sastoji se samo od jednog RC člana, a drugi filter (crvena boja) sastoji se od dva RC člana. Simulacija je obavljena elementima s realnim svojstvima u programu LTSpice/SwitcherCAD III.



Slika 7.1.8. Prikaz odziva dva različita filtra s istom vremenskom konstantom.

Time smo objasnili digitalno-analognu pretvorbu koju koristimo. Potrebno je još prikazati na koji način pojačavamo struju iz mikrokontrolera, te prilagođavamo naponske razine.

Mikrokontroler ima sklop koji generira širinu impulsa. Pozitivna naponska razina tog impulsa ista je kao napon napajanja čipa (5V) a negativna ista kao i masa čipa (0V). Motori su predviđeni da rade na naponu od 24V. Budući da u našoj konzoli imamo simetrično napajanje znači da na svakom ulazu motora moramo osigurati vrijednosti od -12V do +12V. Ukoliko želimo ići punom brzinom naprijed, na jedan izlaz postaviti ćemo -12V, a na drugi +12V, što će ukupno dati 24V.

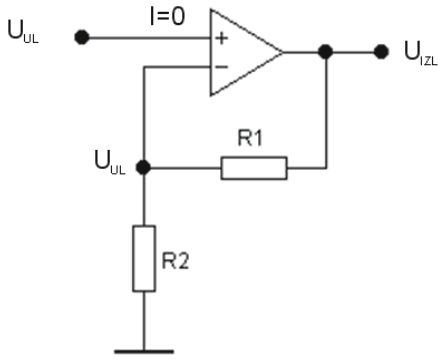


Slika 7.1.9. Shematski prikaz elektronike zadužene za pretvorbu širine impulsa u analogni napon te njegovo pojačanje od 0V-5V u napon -24V do +24V.

Kao što prikazuje Slika 7.1.9. na izlaze iz čipa koji daju širinu impulsa spojeni su RC filteri, koji impulse pretvaraju u analogni napon iznosa od 0V do 5V. Taj napon potrebno je još samo transformirati u napon od -12V do +12V. Za tu namjenu koristimo operacijska pojačala, u neinvertirajućem spoju.

Sva operacijska pojačala imaju ograničenje da ne mogu na izlazu poprimiti puni napon napajanja. Pojačala koja mi koristimo, ukoliko su spojena na napajanje od ±12V, na izlazu mogu poprimiti ±11V, što daje ukupni napon na motoru ±22V. To ćemo uzeti u obzir pri podešavanju željenog pojačanja, u smislu da niti ne pokušavamo postaviti puni iznos napajanja.

Da bi objasnili zašto smo operacijska pojačala spojili kao što prikazuje Slika 7.1.9. prvo ćemo prikazati najobičniji neinvertirajući spoj te ga opisati jednadžbama.

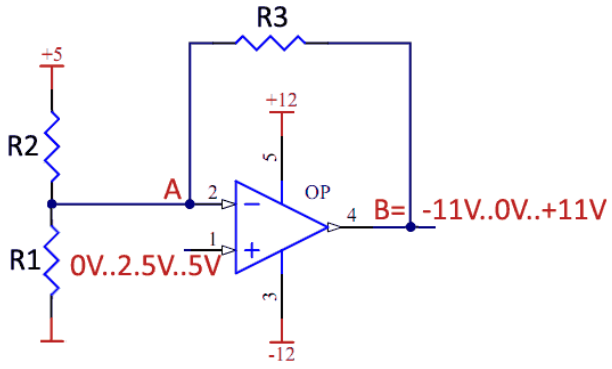


Slika 7.1.10. Shema neinvertirajućeg spoja operacijskog pojačala.

$$U_{UL} = U_{IZL} \cdot \frac{R2}{R2 + R1}$$
$$A = \frac{U_{IZL}}{U_{UL}} = \frac{R2}{R2 + R1} = 1 + \frac{R2}{R1} \qquad (8)$$

U_{UL} – ulazni napon pojačala
A – pojačanje napona, omjer izlaznog i ulaznog napona

Jedini nedostatak u gornjem spoju je što na izlazu ne može biti napon niži od mase, a u našem spoju imamo taj zahtjev. Prikažimo sada i naš spoj u kojem smo dodali otpornik R2 prema 5V, kojem je cilj malo podignuti potencijal negativnog ulaza pojačala, tako da bi on mogao u nekim slučajevima postati viši od potencijala na pozitivnom ulazu, da bi se na izlazu mogao formirati negativni napon.

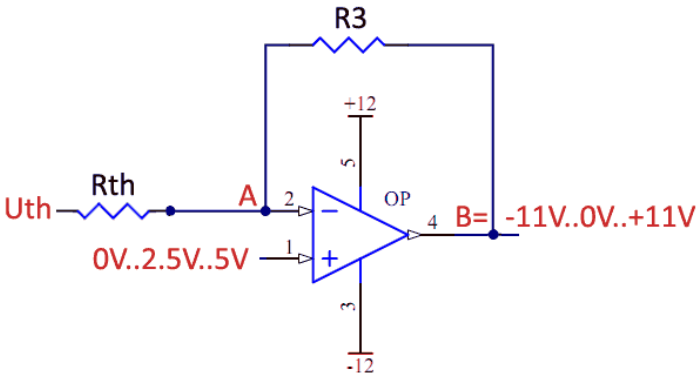


Slika 7.1.11. Shema neinvertirajućeg spoja operacijskog pojačala koji koristimo u aplikaciji.

Potrebno je još izračunati potrebne vrijednosti otpora. Postavimo najprije početne zahtjeve. Otporno dijelilo sastavljeno od otpora R1 i R2 možemo promatrati i po Theveninovom teoremu kao realni naponski izvor čiji napon izvora, i unutrašnji otpor računamo ovako.

$$U_{TH} = \frac{R1}{R1 + R2} \bullet 5V \qquad R_{TH} = R1 \,||\, R2 \qquad (9)$$

Nacrtajmo i ekvivalentnu shemu spoja sa theveninovim izvorom:



Slika 7.1.12. Ekvivalentna shema shemi na slici 7.1.11.

Želimo da napon na izlazu (u točki B) bude 0V, kada je na ulazu (i točki A) 2.5V. U tom slučaju struja teče od točke A prema točki B i na otporu R3 uzrokuje pad napona od 2.5V. Ta ista struja teče i otporom R_{TH}. Dobivamo:

$$U_A = U_B + (U_{TH} - U_B) \cdot \frac{R3}{R3 + R_{TH}}$$

Budući da je U_B=0, dobivamo:

$$U_A = U_{TH} \cdot \frac{R3}{R3 + R_{TH}} \Rightarrow U_{TH} = U_A \cdot \frac{R3 + R_{TH}}{R3} = 2.5V \cdot \frac{R3 + R_{TH}}{R3}$$

$$U_{TH} = 2.5V \cdot \frac{R3 + R_{TH}}{R3} \qquad (10)$$

Ako je napon na ulazu +5V, tada na izlazu napon treba biti +11V. Napišimo jednadžbe i za taj slučaj.

$$U_A = U_B + (U_{TH} - U_B) \cdot \frac{R3}{R3 + R_{TH}} \Rightarrow 5V = 11V + (U_{TH} - 11V) \cdot \frac{R3}{R3 + R_{TH}} \qquad (11)$$

$$5V = 11V + (2.5V \cdot \frac{R3 + R_{TH}}{R3} - 11V) \cdot \frac{R3}{R3 + R_{TH}} = 11V + 2.5V - 11V \cdot \frac{R3}{R3 + R_{TH}} \qquad (10 \text{ i } 11)$$

$$-8.5V = -11V \cdot \frac{R3}{R3 + R_{TH}} \Rightarrow \frac{R3 + R_{TH}}{R3} = \frac{11}{8.5} \Rightarrow \frac{R_{TH}}{R3} = \frac{2.5}{8.5} = \frac{5}{17}$$

$$\frac{R_{TH}}{R3} = \frac{5}{17} \qquad (12)$$

$$U_{TH} = 2.5V \cdot \frac{R3 + R_{TH}}{R3} = 2.5V(1 + \frac{R_{TH}}{R3}) = 2.5V(1 + \frac{5}{17}) = 2.5V \cdot \frac{22}{17} = \frac{55V}{17} \qquad (10 \text{ i } 12)$$

$$U_{TH} = \frac{55\text{V}}{17} \tag{13}$$

$$\frac{55\text{V}}{17} = \frac{R1}{R1+R2}\cdot 5V \rightarrow \frac{11}{17} = \frac{R1}{R1+R2} \rightarrow \frac{17}{11} = \frac{R1+R2}{R1} = 1+\frac{R2}{R1} \rightarrow \frac{R2}{R1} = \frac{6}{11} \tag{9 i 13}$$

$$\frac{R2}{R1} = \frac{6}{11} \Rightarrow R2 = 6k, R1 = 11k \tag{14}$$

Dobili smo omjer otpora R2 i R1, sada napišimo ponovo jednadžbu 12.

$$\frac{R_{TH}}{R3} = \frac{5}{17} \tag{12}$$

$$\frac{R1 \,||\, R2}{R3} = \frac{5}{17} \Rightarrow \frac{6k\cdot 11k}{(6k+11k)\cdot R3} = \frac{5}{17} \Rightarrow \frac{66k^2}{k\cdot R3} = 5 \Rightarrow \frac{66k}{R3} = 5 \tag{12 i 14 i 9}$$

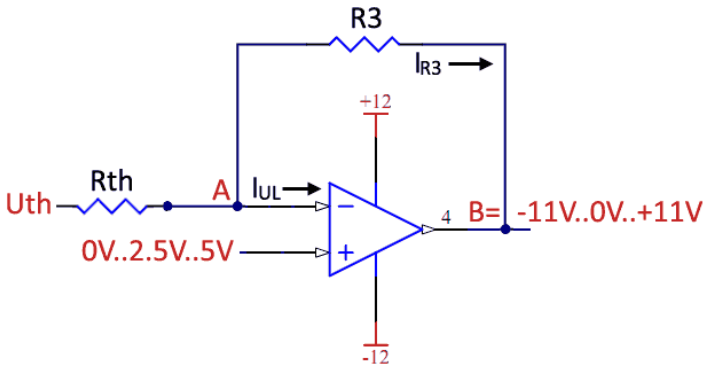
$$R3 = \frac{66k}{5} = 13.2k \tag{15}$$

Sada imamo omjere sva tri otpora bitna za pravilno pojačanje.

$$R1 : R2 : R3 = 11 : 6 : 13.2 = 1.83 : 1 : 2.2$$

Pri odabiru otpornika odabirati ćemo one veličine koje u omjerima najmanje odstupaju od zadanih omjera. Potrebno je još otprilike odrediti apsolutnu veličinu barem jednog otpora.

Generalno, trebali bi koristiti što veće otpore, da smanjimo potrošnju električne energije u povratnoj vezi. Ipak, otpori moraju biti i dovoljno mali, tako da parazitna struja I_{UL} koja ulazi u negativni ulaz pojačala bude zanemariva. (Slika 7.1.13.).



Slika 7.1.13. Shema sa slike 7.1.12, sa ucrtanom parazitnom ulaznom strujom.

Ta struja tipičnog je iznosa 0.2μA. Razmotriti ćemo iznos struje I_{R3} za razliku napona na izlazu i ulazu pojačala od 1V:

$$I_{R3} = \frac{U_{AB} = 1V}{R3} \tag{16}$$

Ta struja mora biti za nekoliko redova veličine veća od I_{UL} čiji je iznos 0.2μA, stoga za minimalni iznos struje I_{R3} odabiremo 0.2mA (1000 puta veći iznos). Veličinu otpora R3 računamo pomoću jednadžbe 16 i dobivamo R3=5K. Taj iznos predstavlja okvirnu gornju granicu veličine otpora.

Na tržištu se ne može kupiti baš bilo koja veličina otpornika, pa ćemo morati izabrati neke iznose koji najmanje odstupaju od ovih omjera, a nalaze se unutar zadanih okvira.

Istraživanjem tržišta izabrali smo slijedeće vrijednosti otpora:

R1=2.7K; R2=1.5K; R3=3.3K

Njihovi omjeri odnose se kao: 1.8 : 1 : 2.2, te ne odstupaju previše od željenih omjera (1.83 : 1 : 2.2). Jedino veličina otpora R1 neznatno odstupa od željenog omjera.

Zbog toga što se koriste dva potpuno simetrična kanala u pojačalu snage, možemo biti sigurni da će nulti položaj (u kojem se motor ne okreće) biti stabilan.

Odstupanja u vrijednosti otpornika od izračunatih omjera mogu samo utjecati na ukupno pojačanje, dakle maksimalnu brzinu motora, te na narušavanje simetričnosti izlaznog napona što bi uzrokovalo da se motor u jednu stranu može vrtiti malo brže nego u drugu.

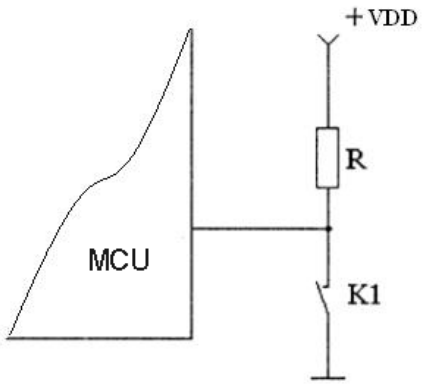
R1 odstupa samo 1.6%, tako da su promjene u ukupnom pojačanju i simetričnosti izlaznog napona neznatne.

Još je potrebno reći da je izlazni RC filter (Slika 7.1.9.) postavljen kako bi pomogao apsorbirati naponske šiljke koje generira istosmjerni motor zbog komutacije četkicama.

7.2. Čip za očitavanje tipkovnice – „Tipke“

Da bi lakše objasnili programsku implementaciju čipa za očitavanje tipki, najprije ćemo napraviti kratki uvod u električne sheme povezivanja tipki sa mikrokontrolerima. Početi ćemo od najjednostavnije sheme, navesti njene nedostatke, te nakon toga opisati shemu koju koristimo.

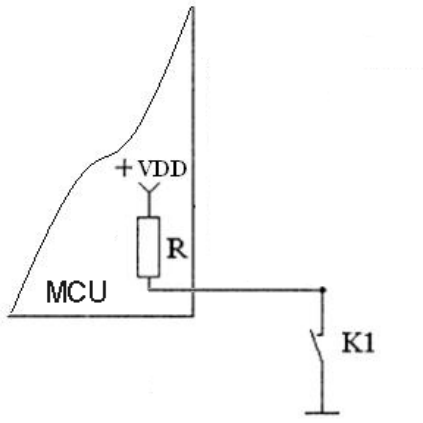
Najjednostavniji način povezivanja tipki sa mikrokontrolerom je slijedeći. Tipku jednim kontaktom spojimo na masu, a drugim na otpornik koji se spaja na napajanje čipa. Kontakt tipke koji se spaja sa otpornikom spojimo sa ulaznim pinom mikrokontrolera (Slika 7.2.1.).



Slika 7.2.1. Električna shema povezivanja tipki sa miktrokontrolerom.

Zadaća otpornika R koji se spaja na napajanje čipa je da, dok tipka nije stisnuta, i kontakt sa masom nije ostvaren, stalno održava napon ulaznog pina na visokoj razini. Kada se tipka stisne, napon na ulaznom pinu biti će jednak masi jer je otpor prema njoj zanemariv. Otpusti li se tipka i nestane kontakta prema masi, tada se napon na ulaznom pinu ponovno vraća na visoku razinu.

Otpornici koji se koriste za ovu namjenu nazivaju se „pull-up“ otpornici, a neki mikrokontroleri ih imaju već ugrađene u sebi, te ih po želji mogu programski uključiti.



Slika 7.2.2. Povezivanje tipki sa miktrokontrolerom sa ugrađenim pull-up otpornikom.

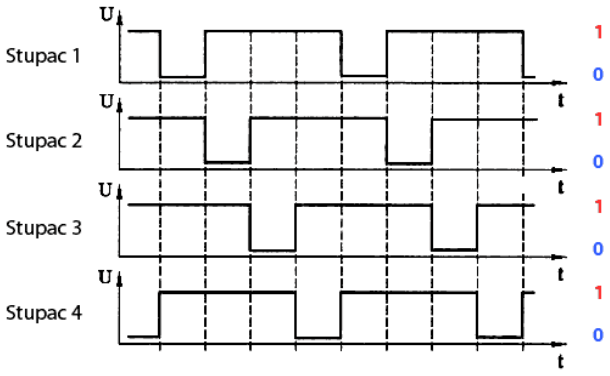
AVR mikrokontroleri koje smo izabrali imaju „pull-up“ otpornike ugrađene u čipu. Shema spajanja sa takvim mikrokontrolerima je vrlo jednostavna. Tipku jednim kontaktom spojimo na masu, a drugim direktno na ulazni pin mikrokontrolera (Slika 7.2.2.). Ukoliko je tipka pritisnuta sa ulaznog pina očitavamo nulu, a ukoliko je otpuštena, očitavamo jedinicu.

Objasnili smo najjednostavniju metodu spajanja tipki sa mikrokontrolerima, međutim njezin nedostatak je što svaka tipka zahtjeva po jedan ulazni pin na čipu. Veličina čipa sa brojem nožica znatno raste, a i cijena, pa je poželjno štedjeti na broju nožica.

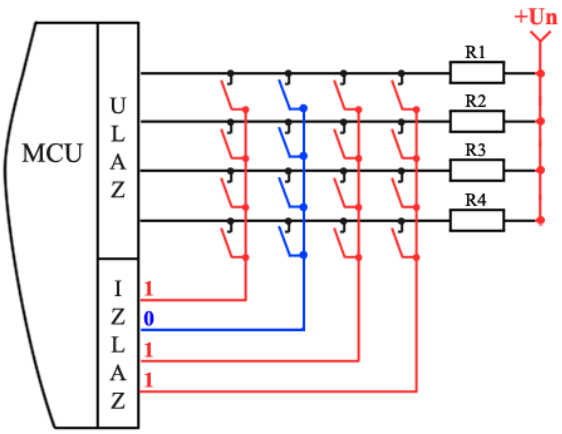
Metoda „šetajuće jedinice“

Da bi uštedjeli na broju nožica čipa upotrijebiti ćemo metodu „šetajuće jedinice“. Ta metoda koristi tipkovnicu organiziranu u mrežu u kojoj su pola nožica izlazne a pola ulazne. Iako se metoda zove šetajuća jedinica, u našoj implementaciji šetati će nula umjesto jedinice.

Tipkovnica je organizirana u mrežu, a očitava se tako da se u petlji očitavaju jedan po jedan stupac tipki. Svaki izlazni pin predstavlja jedan stupac tipki. U svakom koraku petlje čip postavlja sve svoje izlaze na '1', osim jednoga izlaza koji postavi na '0', gdje '1' predstavlja visoki naponski nivo, a '0' niski (Slika 7.2.3.).



Slika 7.2.3. Vrijednosti izlaznih pinova mikrokontrolera u pojedinim prolascima kroz petlju.



Slika 7.2.4. Ilustracija metode šetajuće jedinice sa 16 tipki organiziranih u mrežu 4x4.

U svakom prolazu petlje čip očitava sve redove (ulazne pinove) i provjerava je li koji ulazni pin očitao vrijednost '0'. Ukoliko niti jedna tipka u stupcu nije pritisnuta, tada će svi ulazni pinovi biti postavljeni na '1'.

Ukoliko je pritisnuta tipka, tada na barem jednom ulaznom pinu očitava vrijednost '0'. Budući da znamo koji je stupac postavljen na '0', možemo odrediti koja je tipka pritisnuta križajući redak i stupac.

Na taj način moguće je povezati mikrokontroler sa više tipki nego što ima ulazno-izlaznih pinova. Sa 16 pinova, moguće je je upravljati tipkovnicom od 8 redova i 8 stupaca i tako očitavati 64 tipki.

Slika 7.2.4. prikazuje implementaciju metode u našoj aplikaciji, sa napomenom da umjesto vanjskih „pull-up“ otpornika koristimo one ugrađene na čipu.

Oni su na slici ucrtani kao vanjski radi lakšeg objašnjenja zašto u našoj implementaciji šeta nula umjesto jedinice. Kada bi šetala jedinica, tada bi otpornike jednim krajem trebali spojiti na masu, umjesto na napon napajanja. Takvi otpornici nazivaju se „pull-down“ otpornici. Budući da na čipu nemamo ugrađene „pull-down“ otpornike nego „pull-up“, zamijenili smo naponske nivoe u metodi šetajuće jedinice tako da koristimo manje vanjskih komponenata.

U našoj aplikaciji koristimo 40 tipki na tipkovnici, pa smo odlučili koristiti metodu šetajuće jedinice u mreži od 5 izlaznih i 8 ulaznih pinova.

Budući da su pinovi na mikrokontroleru grupirani u grupe od 8 pinova (engl. PORT), koji se mogu očitati u jednom vremenskom taktu, najjednostavnije je na ulazu imati 8 ulaznih pinova. Na taj način čak koristimo i manje pinova nego da smo koristili mrežu 7 x 7.

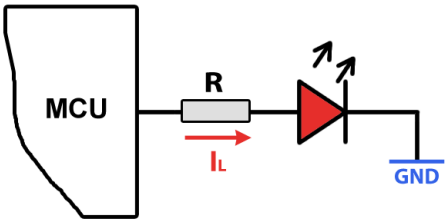
Pseudokod programa je slijedeći:

```
Petlja {
   Tipka=0;
   Za (Stupac = 0 do 4) {
      Izlazni_port= 0b11111 - (1<<Stupac);
      //postavlja 5 izlaznih pinova
      Očitaj Ulazni_port; //očitava 8 ulaznih pinova
      Ako je Ulazni_port različit od 0xFF tada {
            Tipka=Stupac*8+redni_broj_aktivnog_pina;
            izađi;
            }
   }
   šalji Tipka;
}
```

7.3. Implementacija čipa za paljenje svjetlećih dioda – „Ledice“

Na konzoli imamo 30 LED dioda koje je potrebno biti u mogućnosti programski paliti i gasiti. „Glavni“ čip odlučuje koje će svjetleće diode biti upaljene, a koje ugašene, a čip „Ledice“ ih pali i gasi.

Najprije ćemo objasniti električnu shemu potrebnu za paljenje i gašenje svjetlećih dioda mikrokontrolerom.



Slika 7.3.1. Svjetleća dioda spojena na izlazni pin mikrokontrolera.

Svaka svjetleća dioda je dizajnirana da radi na unaprijed određenom naponu, i da kroz nju prolazi unaprijed određena struja.

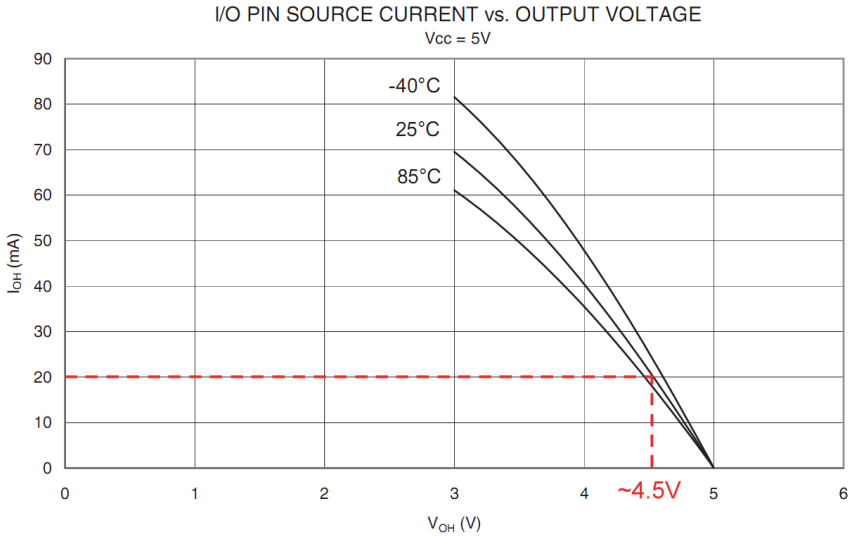
Na primjer, svjetleće diode koje koristimo predviđene su za rad na naponu od 1.8 V, te da provode struju iznosa 20mA. Međutim, izlazni pin mikrokontrolera može zauzeti samo dvije naponske razine, 0V, i +5V, pa je potrebno prilagoditi naponsku razinu. Da bi prilagodili naponske razine, u seriju sa LED diodom dodajemo otpornik (Slika 7.3.1.).

Funkcija otpornika R sa slike je preuzeti „višak“ napona, i tako dovesti svjetleću diodu u njeno predviđeno područje rada. Drugim riječima, otpornik štiti od pregaranja svjtleću diodu i izlazni pina čipa.

Kada bi izlazni pin mikrokontrolera bio idealan naponski izvor tada bi iznos otpornika računali na slijedeći način.

$$R = \frac{V_{CC} - V_{LED}}{I_L} = \frac{5V - 1.8V}{20mA} = 160\Omega$$

Budući da izlazni pin mikrokontrolera nije idalan naponski izvor izračunat ćemo napon koji on daje pri struji od 20mA. Takvu informaciju možemo iščitati iz specifikacija mikrokontrolera (Slika 7.3.2.).



Slika 7.3.2. Napon na izlaznim pinovima u ovisnosti o temperaturi i struji koju daju.

Uzevši u obzir pad napona na izlaznom pinu, iznos otpora računamo na slijedeći način.

$$R = \frac{V_{CC}(20mA) - V_{LED}}{I_L} = \frac{4.5V - 1.8V}{20mA} = 135\Omega$$

Vidimo da uračunavanjem realnog pada napona na izlaznom pinu iznos potrebnog otpora pada sa 160Ω na 135Ω, čime bi se dalo zaključiti da je iznos unutarnjeg otpora izlaznog pina ~25Ω. Iako su svjetleće diode dosta robusne, i nije bitno ovako precizno računanje otpora, mi smo ga naveli, da bi na primjeru pokazali ponašanje izlaznog pina kao realnog naponskog izvora, te praktično korištenje dokumentacije čipa.

Ono što je bitno za reći je da je za kontrolu svake svjetleće diode potreban jedan odvojeni pin mikrokontrolera. Ukoliko mikrokontroler želi upaliti svjetleću diodu, potrebno je postaviti visoki naponski nivo na tom izlaznom pinu, a ukoliko ju želi ugasiti potrebno je postaviti niski naponski nivo.

Komunikacija čipa „Ledice“ i „Glavnog“ čipa

Komunikacija čipa „Ledice“ sa „Glavnim“ čipom vrlo je jednostavna. „Glavni“ čip uzastopno šalje oktete podataka. Svaki oktet podataka predstavlja skupinu od 8 svjetlećih dioda. „Glavni“ čip najprije šalje sinkronizacijski oktet sa svim jedinicama, koji služi usklađivanju ova dva čipa u brojanju okteta. Nakon toga šalju se 4 okteta koja sadržavaju informacije o paljenju 32 svjetleće diode. Time je opisan ciklus slanja informacija od „Glavnog“ čipa, čipu „Ledice“.

Budući da je stabilnost sustava na prvom mjestu u aplikaciji, svi su čipovi, osim „Glavnog“ dizajnirani na način da ne ometaju nastavak rada sustava ukoliko oni iz nekog nepredviđenog razloga prestanu raditi, nego se samo isključuje područje rada za koje su oni zaduženi. Tako čip „Ledice“ možemo tokom rada konzole čak i izvaditi

iz tiskane pločice, a da se ništa ne promijeni u radu, osim što će se svjetleće diode ugasiti. Ukoliko ga nakon toga ponovo uključimo svjetleće diode će se ponovno upaliti i sustav će nastaviti raditi kao da se ništa nije dogodilo.

Budući da je konzola zamišljena na taj način da nikad nije upaljeno svih 8 svjetlećih dioda u grupi, nikad se neće dogoditi, da se podatkovni oktet krivo protumači kao sinkronizacijski, ili obrnuto.

Pseudokod rada čipa „Ledice“:

```
Prekidni potprogram primi_oktet {
   Oktet=primljeni_oktet;
   Ako je (Oktet == 0b11111111) tada postavi Brojač=0;
   Inače {
      Ako je (Brojač == 0) PORTA=Oktet;
      Ako je (Brojač == 1) PORTB=Oktet;
      Ako je (Brojač == 2) PORTC=Oktet;
      Ako je (Brojač == 3) PORTD=Oktet;
      Brojač ++; //Povećaj brojač
   }
}
```

Već smo spomenuli da je za komunikaciju zadužen poseban sklop unutar mikrokontrolera, tako da samu komunikaciju nije bilo potrebno programski implementirati. Sklop u čipu „Ledice“ podešen je da radi na taj način, da kada primi oktet podataka, krene izvršavati prekidni potprogram.

U prekidnom potprogramu nalazi se cijeli aktivni programski kod kao što se može vidjeti u gore napisanom pseudokodu.

Objašnjenje pseudokoda:

Varijabla brojač broji oktete, te prema tome određuje na koju se grupu svjetlećih dioda odnosi koji dolazni oktet. Pri dolasku sinkronizacijskog okteta, varijabla brojač se poništava na početnu vrijednost 0.

Dakle jedan ciklus postavljanja stanja svjetlećih dioda sastoji se od slanja 5 okteta „Glavnog“ čipa ka čipu „Ledice“. Od toga je jedan oktet sinkronizacijski, a četiri podatkovna.

7.4. Implementacija funkcija u „Glavnom“ čipu

Kao što smo već ranije spomenuli „Glavni“ čip je mozak cijelog sustava. Najveći dio programskog koda sadržan je baš u „glavnom“ čipu.

Dok je strojni kod za ostale mikrokontrolere zauzimao u prosjeku do 2 kilookteta programske memorije na čipu, strojni kod „Glavnog“ čipa veličine je oko 30 kilookteta, zbog čega smo u implementaciji izabrali čip AVR ATMega32 koji ima ugrađeno 32 kilookteta programske memorije.

Zbog veličine programskog koda, koji je u izvornom obliku pisan u programskom jeziku C, te zauzima oko 3.000 tisuće linija koda, opisati ćemo samo bitnije i zanimljivije dijelove. Podastrijet ćemo ih matematičkim jednadžbama, fizičkim formulama, a implementaciju opisati pseudokodom.

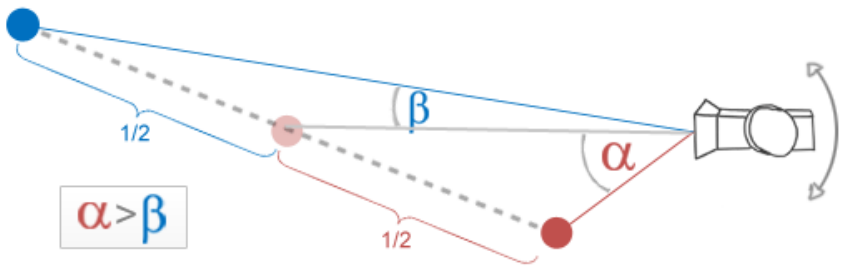
Realizacija glatkih prijelaza sa jedne točke ciljanja na drugu

U poglavlju 6.1. već smo psudokodom opisali algoritam zadužen za ciljanje zadane točke, a u poglavlju 4.4. matematičke funkcije zadužene za izračun ciljane točke, te za automatsko zakretanje robotske glave prema istoj. Tako da ćemo u ovom poglavlju opisati što se događa kada korisnik tijekom ciljanja jedne točke, izabere neku drugu točku koju želi automatski ciljati.

Kamera se mora lagano i neprimjetno okrenuti prema novoj ciljanoj točki. Nagle promjene brzine (trzaji), su nepoželjne zbog narušavanja sklada kadra.

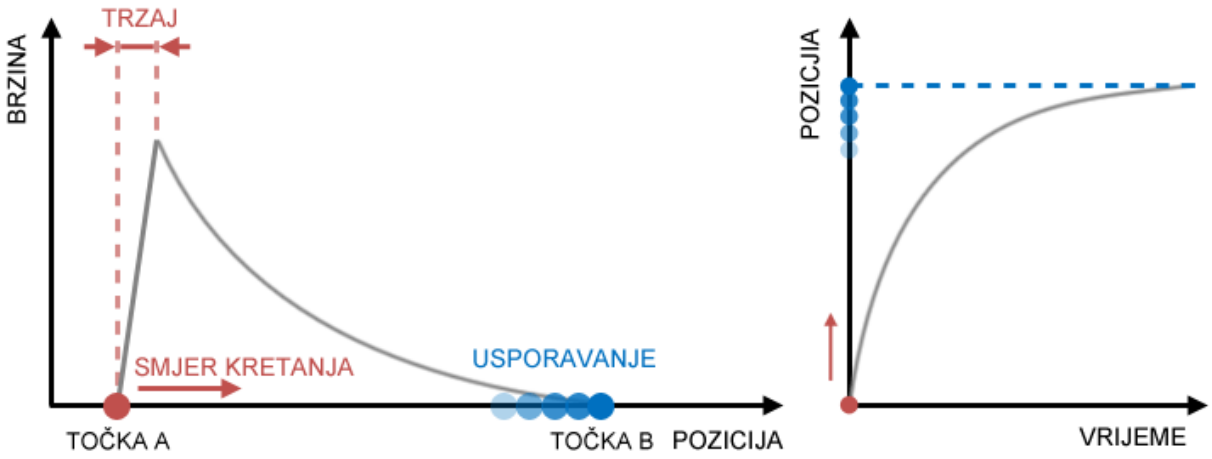
Osim toga, želimo kameru okretati od jedne točke prema drugoj, neovisno o pomicanju ruke dizalice. Dakle želimo da se ciljana točka pomiče od prve točke ka drugoj.

Najjednostavniji način koji se odmah nameće kao rješenje vjerojatno je linearno interpolirati ciljanu točku od prve točke ka drugoj, međutim takav pristup ima svojih nedostataka koje ćemo opisati.



Slika 7.4.1. Nelinearnost brzine kamere u odnosu na pomicanje ciljane točke linearnom interpolacijom.

Slika 7.4.1. prikazuje kameru koja se zakreće od crvene točke prema plavoj točci. Plava točka znatno je više udaljena od kamere nego crvena. Ukoliko bi se ciljana točka linearno pomicala po pravcu koji spaja te dvije točke, tada bi se, u početku gibanja, kamera pomicala znatno brže nego pri kraju gibanja (kut α znatno je veći od kuta β). Osim toga, pri kretanju bi dolazilo do vidljivog trzaja (Slika 7.4.2.).



Slika 7.4.2. Graf brzine i pozicije glave pri prijelazu ciljanja sa točke A na točku B realiziran jednostavnom linearnom interpolacijom između tih točaka.

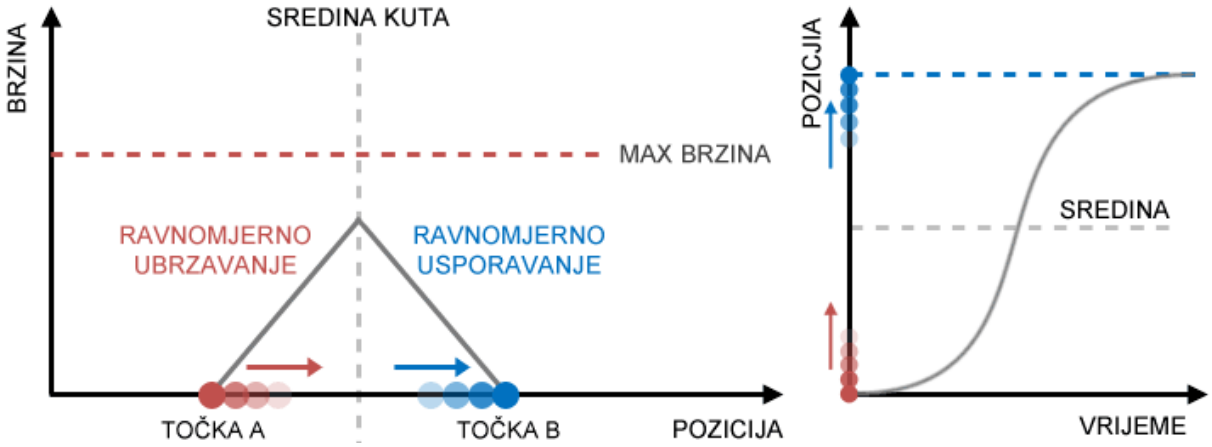
Do trzaja pri kretanju dolazi zato jer je točka A puno bliža kameri od točke B (Slika 7.4.1.). Ukoliko bi situacija bila obrnuta, do trzaja bi dolazilo na kraju prijalaza. Ukoliko bi obadvije točke bile jednako udaljene od kamere, tada bi do trzaja dolazilo i pri kretanju i pri stajanju. Da bi izbjegli ovakvo neravnomjerno i nepredvidljivo kretanje, moramo prvo naći neki linearni model pomicanja međutočke, kojim ćemo biti u mogućnosti postići konstantnu brzinu okretanja motora.

Najjednostavniji linearni model pomicanja ciljane točke možemo dobiti tako da izračunamo početni i krajnji **kut** kamere, te u međukoracima računamo željene pozicije motora dobivene linearnom interpolacijom između ta **dva kuta.**

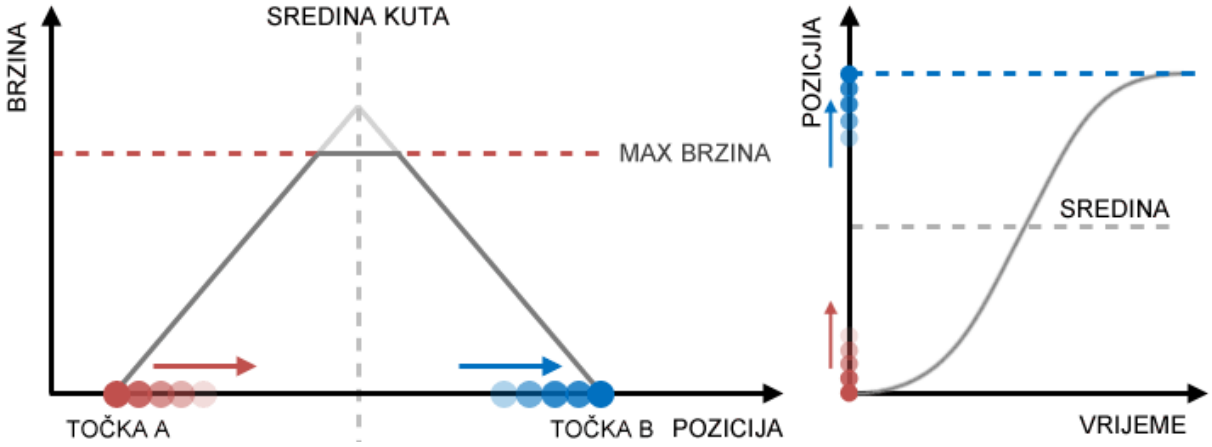
Slika 7.4.3. Graf brzine i pozicije glave pri prijelazu ciljanja sa točke A na točku B realiziran jednostavnom linearnom interpolacijom između početnih kutova.

Vidimo da ovim pristupom imamo ravnomjernu brzinu prijelaza, međutim i na početku i na kraju prijelaza dolazi do trzaja, tako da i taj princip nije zadovoljavajući.

Potrebno je postići pomicanje od prve točke ka drugoj, ali na taj način, da brzina nije cijelo vrijeme konstantna, jer to uzrokuje trzaje na početku i kraju kretnje, nego da ciljana međutočka polagano ubrzava krenuvši od potpunog mirovanja na točki A, te prije dolaska na željenu točku lagano usporava do ponovnog mirovanja. Pod stanjem mirovanja mislimo na mirovanje točke ciljanja, a ne na mirovanje robotske glave. Moguće je da se robotska glava i dalje kreće ukoliko se ruka dizalice pomiče, da bi kompenzirala pomak ruke. Željeni model opisuju Slika 7.4.4 i Slika 7.4.5.



Slika 7.4.4. Graf brzine i pozicije glave pri glatkom prijelazu sa jedne točke ciljanja na drugu.



Slika 7.4.5. Graf brzine i pozicije glave pri glatkom prijelazu sa jedne točke ciljanja na drugu, u slučaju jako udaljenih točaka kada se postiže maksimalna brzina.

Slika 7.4.4. i Slika 7.4.5. opisuju željeno ponašanje sustava. Pri početku prijelaza međutočka ubrzava od točke A do točke B jednolikim ubrzanjem. Kada pozicija međutočke pređe polovicu željenog **kuta**, tada se program prebacuje u inverzni način rada, te započinje usporavanje. Iznos usporavanja jednak je iznosu ubrzanja ali je suprotnog predznaka.

Da bi uspjeli izvesti ovakvu implementaciju opisanu slikama: Slika 7.4.4. i Slika 7.4.5., potrebno je pomno razmotriti sustav.

Navesti ćemo nekoliko premisa koje vode do ispravnog rješenja:

- Očito je da je potrebno do polovice kuta koristiti model sa ubrzanjem, a nakon polovice model sa usporavanjem.
- Slijedi da je potrebno izračunati kut, te njegovu polovicu.
- U svakom koraku pomicanja međutočke potrebno je ponovno izračunati kut, te njegovu polovicu, zato jer se oni mijenjaju pomicanjem ruke.
- Budući da se iznos kuta i njegove polovice mijenja, moguć bi bio slijedeći scenarij. Negdje na sredini prijelaza kut robotske glave pređe polovicu kuta između točke A i B, te se prebaci iz modela ubrzavanja u model usporavanja. Nakon toga pomakne se ruka dizalice te se promijene se kutovi između krajnjih točaka A i B, a njihovo polovište se također pomakne prema bivšoj točki B. Zbog pomaka polovišta, sadašnja međutočka je opet u prvoj polovici kuta, te se sustav ponovno vrati u model ubrzavanja. Takav razvoj događaja bi potencijalno prouzročilo trzaj u kretnji. Da bi to izbjegli, nećemo pamtiti apsolutni iznos kuta, nego **postotak od ukupnog kuta između točke A i B**. Kada taj postotak postane veći od 50%, tada se odvija prebacivanje u model usporavanja.
- Međutim, ukoliko bi samo koristili postotak kuta pri izračunu, tada bi vremensko trajanje prijelaza bilo uvijek isto (postojala bi vremenska konstanta prijelaza), a udaljenost točaka bi znatno utjecala na iznos ubrzanja. To znači da bi prijelaz za točke koje su vrlo blizu, bio nepotrebno prespor, a da bi za vrlo udaljene točke bio prebrz. Dakle u računu moramo koristiti i postotak i apsolutni kut.
- Brzina u svakom trenutku mora biti određena apsolutnim udaljenošću kuta između međutočke do točke A, ili točke B (manji kut od ta dva kuta).
- U jednom ciklusu prijelaza računa se ukupni kut između točke A i B, te na osnovu postotka izračunatog u prethodnom ciklusu računamo apsolutni kut međutočke iz prethodnog ciklusa (koji je malo različit zbog pomaka ruke), te na osnovu toga međukuta računamo novi međukut.
- Opisani postupak opisuje prijelaz za jednu os rotacije kamere. Iznos međukuta druge osi računamo tako da je u postotku od ukupnog kuta uvijek isti kao i međukut prve osi. Kao prvu os odabiremo onu os koja u početku prijelaza ima veću razliku u početnom i završnom kutu, radi bolje preciznosti.

To rekaviši potrebno je još odrediti poveznicu između brzine i pređenog puta motora. Cilj je postići linearno ubrzanje. Napišimo fizičke formule za brzinu i put u ovisnosti o vremenu, pri konstantnom ubrzanju.

$$v = \int a \cdot dt = a \cdot t \tag{17}$$

$$s = \int v \cdot dt = \iint a \cdot dt^2 = \int a \cdot t \cdot dt = \frac{a \cdot t^2}{2} \tag{18}$$

Želimo li dobiti ovisnost brzine o putu pri konstantnom ubrzanju, potrebno je eliminirati nepoznanicu vrijeme iz jednadžbi 10 i 11.

$$v = a \cdot t \Rightarrow t = \frac{v}{a} \tag{19}$$

$$s = \frac{a \cdot t^2}{2} \Rightarrow t = \sqrt{\frac{2 \cdot s}{a}} \tag{20}$$

$$\frac{v}{a} = \sqrt{\frac{2 \cdot s}{a}} \Rightarrow v = \sqrt{2a \cdot s} \tag{19 i 20}$$

$$v = \sqrt{2a \cdot s} = \sqrt{2a} \cdot \sqrt{s} \tag{21}$$

$$v = K \cdot \sqrt{s} \tag{22}$$

Jednadžba 20 dobivena je izjednačavanjem jednadžbi 18 i 19 i eliminacijom nepoznanice vrijeme. Jednadžba 21 uvodi jednu konstantu **K** umjesto svih konstanti iz formule 20. Ta konstanta određuje iznos ubrzanja.

Vidimo dakle da brzina mora biti proporcionalna korijenu udaljenosti ukoliko želimo konstantno ubrzanje pri kretanju.

Pseudokod glatkog prijelaza za jednu os:

```
Petlja {
     Kut A = Izračunaj_kut(Točka A);
     Kut B = Izračunaj_kut(Točka B);
     Razlika = Kut B - Kut A;

     Ako je Postotak veći od 0.5(50%) tada
          Udaljenost = Postotak * Razlika;
     Inače
          Udaljenost = (1 - Postotak) * Razlika;

     Brzina = Korijen (Udaljenost) * Konstanta ubrzanja;
     Ako je (Brzina veća od Najveće brzine)
          tada ograniči Brzinu na Najveću brzinu.
     Ako je (Postotak >= 1) tada
          Postotak = 1;
          Pozicija = Kut B;
     Inače Pozicija = Kut A + Postotak * Razlika + Brzina;

     Pozicioniraj_motor(Pozicija);

     Postotak = Postotak + Brzina / Razlika;
}
```

Namjena pojedinih varijabli:

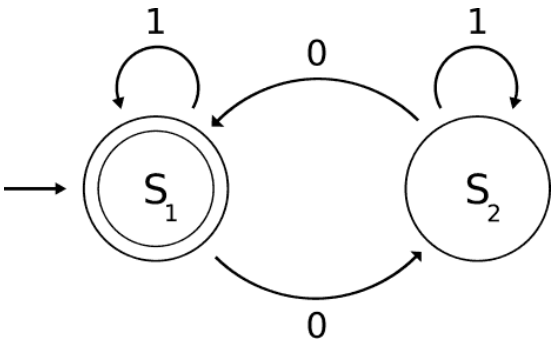
Točka A – *Početna točka*
Točka B – *Završna točka*
Kut A – *početna pozicija motora*
Kut B – *završna pozicija motora*
Razlika – *put koji motor mora preći da dođe od početne do završne pozicije motora*
Postotak – *Postotak prijeđenog puta motora, od ukupno potrebnog puta koji je potrebno preći od točke A do B.*
Udaljenost – *varijabla koja određuje udaljenost međutočke od bliže od dviju krajnjih točaka. Ova varijabla služi za izračun brzine.*
Brzina – *Brzina koja linearno raste sa udaljavanjem od početne točke A, te opada približavanjem završnoj točci B. Brzina predstavlja pomak u impulsima motora tokom 30ms.*
Pozicioniraj_motor() – *funkcija koja šalje čipu „motor" željenu poziciju motora.*

Implementacija izbornika za upravljanje sustavom

Izbornik je implementiran kao deterministički konačni automat (DKA). DKA se sastoji od 49 stanja. Ovisno o pritisnutoj tipki ili pomaku upravljačke palice automat prelazi iz jednog stanja u drugo.

Zbog veličine cijelog automata, nećemo ga cijelog prikazati, nego ćemo pseudokodom prikazati njegovu implementaciju, a nakon toga grafički prikazati samo stanja automata zadužena za automatsko ciljanje.

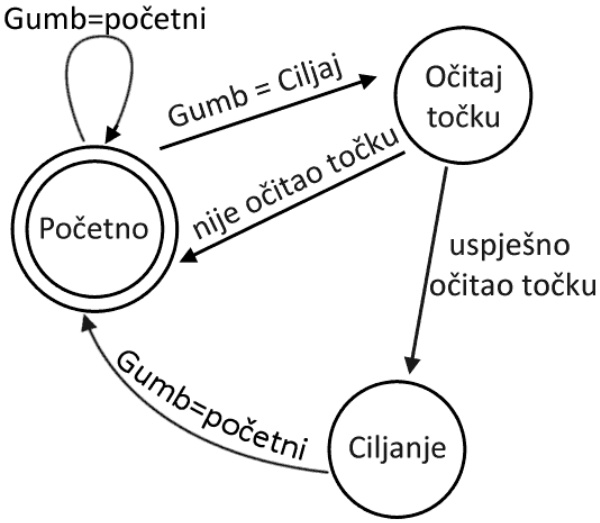
Najprije ćemo ukratko opisati jedan jednostavan deterministički konačni automat.



Slika 7.4.6. Jednostavni deterministički konačni automat sa dva stanja.

Prikazani automat (Slika 7.4.6) sastoji se od dva stanja (S1 i S2) i dva ulazna znaka (0 i 1). U početku se automat nalazi u početnom stanju S1 koje je označeno dvostrukom kružnicom. Dolaskom ulaznog znaka, u našem slučaju to je pritisak na tipku, automat prelazi iz jednog stanja u drugo, ili ostaje u istom stanju.

Prikažimo sada i dio DKA koji je zadužen za upravljanje sustavom ciljanja.



Slika 7.4.7. Dio DKA koji je zadužen za upravljanje sustavom ciljanja.

Pseudokod implementacije DKA kojeg prikazuje Slika 7.4.7. :

```
Petlja {
  Čekaj_vremenski_signal();
   //čeka signal prekidnog potprograma
   //koji dolazi signal svakih 33 ms
   Upali potrebne svjetleće diode();
   //Komunicira sa čipom „Ledice"
  TIPKA = Očitaj_tipke;
  //komunicira sa čipom „Tipke"
  Ako je (STANJE == POCETNO) Tada {
     Izvrši potprogram „Početni"
     Ako je (TIPKA == POCETAK) tada STANJE=POCETNO
     Ako je (TIPKA == CILJAJ) tada  STANJE=OCITAJ_TOCKU
   }
   Ako je (STANJE == OCITAJ_TOCKU) Tada {
     Izvrši potprogram „Očitaj točku"
     Ako je potprogram uspio STANJE=CILJANJE
                         inače STANJE=POCETNO
   }
  Ako je (STANJE == CILJANJE) Tada {
     Izvrši potprogram „Ciljaj_tocku"
     Ako je (TIPKA == POCETAK) tada STANJE=POCETNO
   }
}
```

8. Opis funkcija konzole

Slika 8.1. Upravljačka konzola.

Vidimo da je konzola podijeljena u dvije sekcije. Lijeva sekcija zadužena je za memoriranje pokreta, a desna za automatsko ciljanje te kretanje po izborniku. Između te dvije sekcije nalazi se upravljačka palica koja služi za ručno upravljanje motorima.

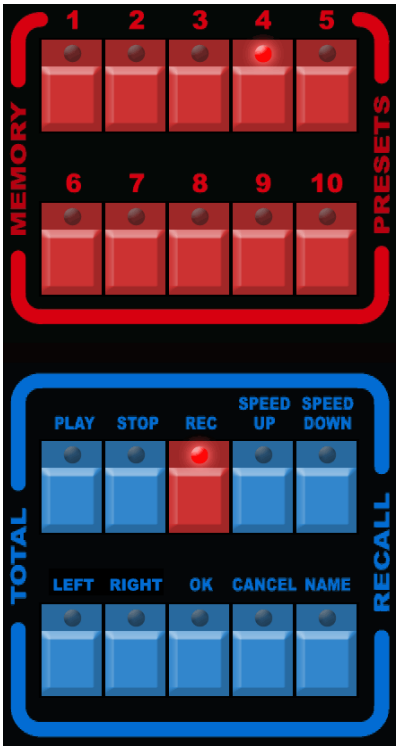
Svaka od dvije sekcije podijeljena je na gornji i na donji dio.

Slika 8.2. Pogled na upravljačku konzolu iz ptičje perspektive.

Opisati ćemo zasebno funkcije tipki na lijevoj i desnoj sekciji.

Lijeva sekcija

U gornjem dijelu lijeve sekcije nalazi se 10 tipki koje predstavljaju 10 memorijskih mjesta za snimanje pokreta. Te tipke su crvene boje radi lakšeg razlikovanja. U jednom trenutku odabrana je samo jedna od 10 lokacija kao aktivna, te iznad nje svijetli pripadajuća svjetleća dioda.



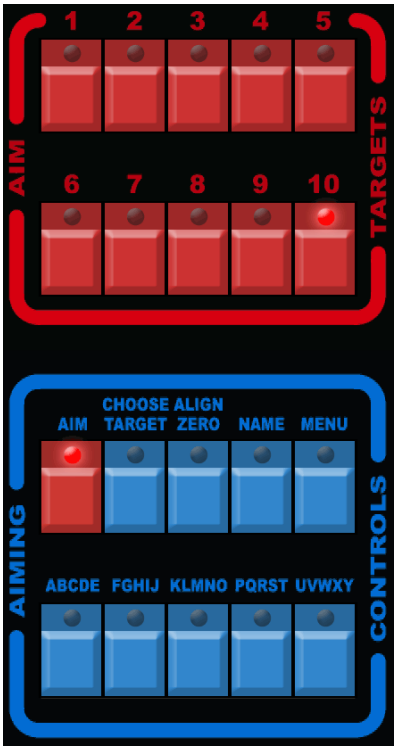
Slika 8.3. Lijeva sekcija konzole.

U donjem dijelu lijeve sekcije nalaze se kontrole za snimanje i ponavljanje pokreta. Navesti ćemo namjenu pojedinih tipki donjeg dijela ove sekcije:

1. Play – Ponavlja snimljeni pokret kamere sa aktivne memorijske lokacije.
2. Stop – Prekida izvođenje snimljenog pokreta.
3. Rec – Započinje snimanje pokreta na aktivnu memorijsku lokaciju.
4. Speed up – povećava brzinu izvođenja snimljenog pokreta. Omogućuje da neki pokret snimimo polako i precizno, te da ga poslije ubrzamo.
5. Speed down – smanjuje brzinu izvođenja snimljenog pokreta.
6. Left – Pomicanje u izborniku u lijevo.
7. Right – Pomicanje u izborniku u desno.
8. Ok – Potvrda akcije. Na primjer ukoliko želimo snimiti pokret na lokaciju na kojoj je već snimljen neki pokret biti će potrebno akciju potvrditi pritiskom ove tipke.
9. Cancel – Odustajanje od akcije.
10. Name – Davanje imena pojedinoj memoriji.

Desna sekcija

U gornjem dijelu desne sekcije nalazi se 10 tipki koje predstavljaju 10 različitih ciljanih točaka. U jednom trenutku odabrana je samo jedna od 10 točaka kao aktivna, te iznad nje svijetli pripadajuća svjetleća dioda. Tipke gornjeg dijela obojane crvenom bojom kao i u lijevoj sekciji.



Slika 8.4. Desna sekcija konzole.

Donji dio desne sekcije sadrži kontrole memoriranje ciljanih točaka, prebacivanje u stanje automatskog prećenja, te upravljanje izbornikom.

Navesti ćemo namjenu pojedinih tipki donjeg dijela ove sekcije:

1. AIM – Ulazi u stanje automatiziranog praćenja točke.
2. Choose target – Odabir ciljane točke. Izbornik korisnika navodi da točku nacilja dva puta iz dva različita kuta.
3. Align zero – Podešava nulti kut kamere.
4. Name – Daje ime pojedinoj od 10 točaka ciljanja.
5. Menu – Ulazi u glavni izbornik.
6. ABCDE – Pritiskom na ovu tipku odabiremo neko od pet slova iz imena tipke. Koristi se pri davanju imena točkama, i ima princip rada, kao i pisanje poruka na mobilnim telefonima.
7. – 10. Imaju istu namjenu kao i tipka 6.

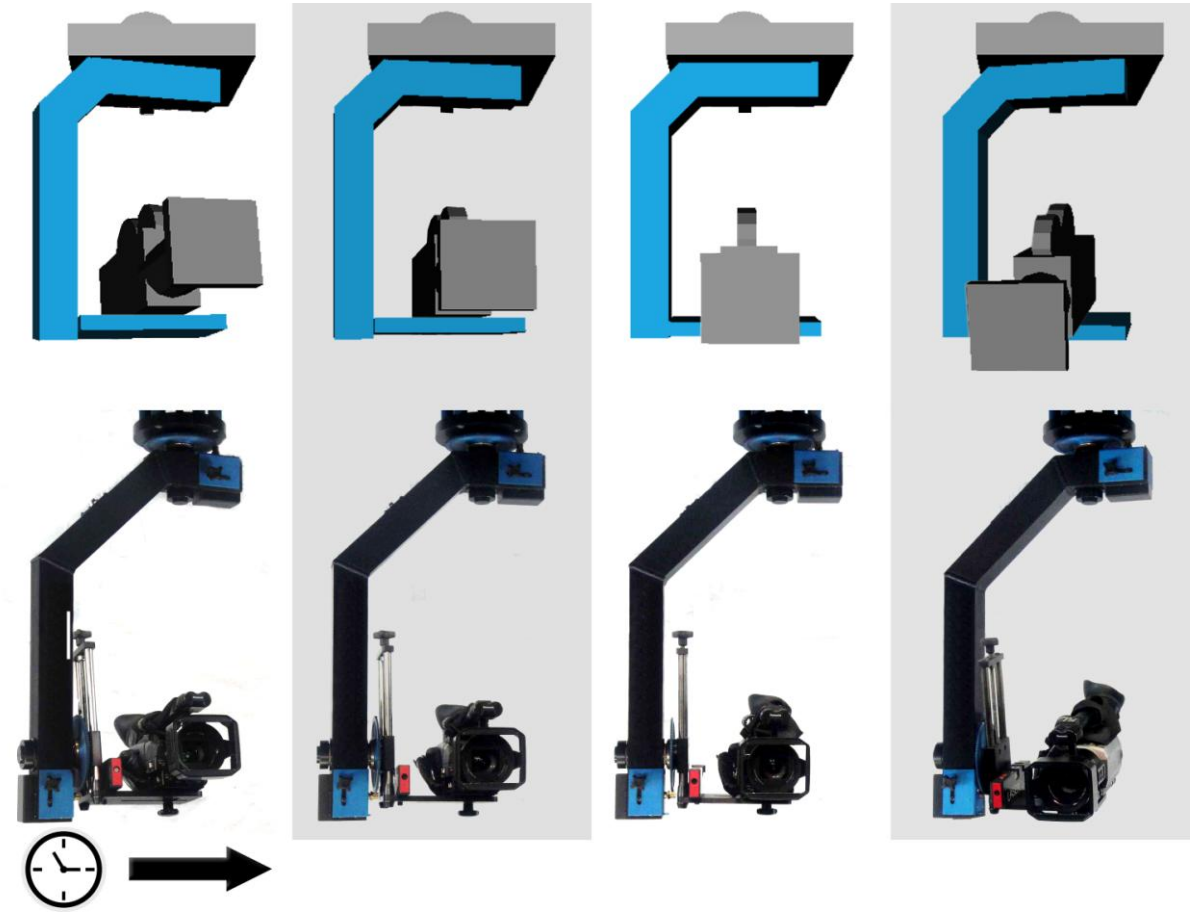
9. Usporedba stvarnog sustava i računalnog modela

U računalnom modelu svi zglobovi imaju isti broj stupnjeva slobode kao i stvarni proizvod, te su svi segmenti jednake dužine, ali ipak, izrađeni model ne odgovara u potpunosti stvarnom modelu.

Najveća razlika se očituje u tome što model ne uzima u obzir vrijeme potrebno za reakciju elektromotora, niti tromost tijela koja sprečava nagle pokrete ruke i robotske glave. Stvarni model koristi PID algoritam za zauzimanje željenog kuta glave, dok računalni model jednostavno „zauzme“ tu poziciju.

Računalni model možemo promatrati i kao impulsnu pobudu koja trenutno zauzima željenu vrijednost, a koju smo crtali u poglavlju 7.1. gdje smo opisivali PID kontroler.

Zbog navedenih razloga u stvarnosti se pojavljuju odstupanja, ili kašnjenja, u orjentaciji kamere stvarnog sustava u odnosu na računalni model. Cilj je u konačnom proizvodu te razlike smanjiti što je više moguće.



Slika 9.1. Usporedba kretnji robotske glave računalnog modela i stvarnog sustava.

10. Zaključak

Opisali smo razvojni put elektroničkog uređaja za automatizirano upravljanje robotskom glavom dizalice za kameru. Neveli smo i opisali korištene tehnologije, te opisali sheme povezivanja komponenata. Prikazali smo pseudokode korištenih algoritama, koje smo podastrijeli matematičkom i teorijskom podlogom.

Najkompleksniji dio razvoja opisanog uređaja bila je izrada aplikacije u stvarnom vremenu automatiziranog upravljanja jer je dosta teško uskladiti 2 zahtjeva koja su kontradiktorna jedan drugome: što manje odstupanje od ciljane točke, i kretnje bez trzaja.

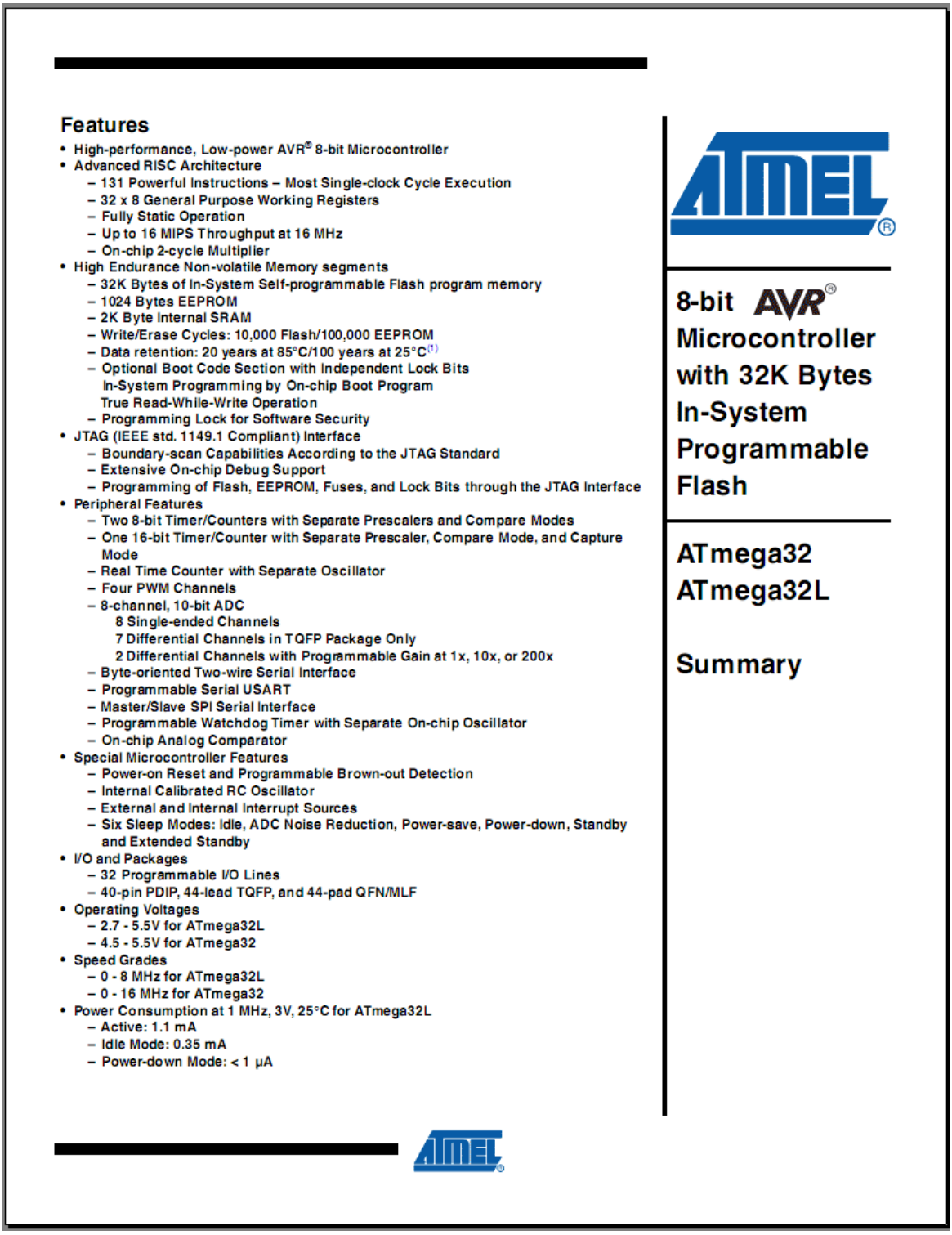
Drugi značajan problem je kontrola vrlo malih brzina, koje su na granici fizičkih mogućnosti motora. Najveća poboljšanja sustava u budućnosti očekujemo baš u tom segmentu, korištenjem motora veće snage, te korištenjem inkrementalnih senzora na motoru visoke rezolucije, koji bi omogućili kvalitetniju kontrolu brzine.

Najveće poboljšanje u jednostavnosti korištenja sustava očekujemo ugradnjom apsolutnih senzora na robotsku glavu, te ugradnjom mogućnosti pomicanja ciljane točke tokom automatiziranog ciljanja.

Opisani proizvod nije primjenom ograničen samo na dizalicu za kameru, nego se, uz prilagodbu, može koristiti i za druge namjene. Na primjer, mogli bi ga koristiti i u solarnim elektranama za pozicioniranje solarnih ploča prema suncu, ili u zvjezdarnicama pri promatranju pojedinih zvijezda. Možemo ga koristiti u bilo kojem sustavu u kojem se jedan objekt treba okretati prema drugome, ukoliko poziciju tih objekata možemo automatizirano računati ili dohvaćati.

11. Prilozi

Prilog 1 - Specifikacija mikrokontrolera Atmel AVR ATMega32



Features

- High-performance, Low-power AVR® 8-bit Microcontroller
- Advanced RISC Architecture
  - 131 Powerful Instructions – Most Single-clock Cycle Execution
  - 32 x 8 General Purpose Working Registers
  - Fully Static Operation
  - Up to 16 MIPS Throughput at 16 MHz
  - On-chip 2-cycle Multiplier
- High Endurance Non-volatile Memory segments
  - 32K Bytes of In-System Self-programmable Flash program memory
  - 1024 Bytes EEPROM
  - 2K Byte Internal SRAM
  - Write/Erase Cycles: 10,000 Flash/100,000 EEPROM
  - Data retention: 20 years at 85°C/100 years at 25°C(1)
  - Optional Boot Code Section with Independent Lock Bits
    In-System Programming by On-chip Boot Program
    True Read-While-Write Operation
  - Programming Lock for Software Security
- JTAG (IEEE std. 1149.1 Compliant) Interface
  - Boundary-scan Capabilities According to the JTAG Standard
  - Extensive On-chip Debug Support
  - Programming of Flash, EEPROM, Fuses, and Lock Bits through the JTAG Interface
- Peripheral Features
  - Two 8-bit Timer/Counters with Separate Prescalers and Compare Modes
  - One 16-bit Timer/Counter with Separate Prescaler, Compare Mode, and Capture Mode
  - Real Time Counter with Separate Oscillator
  - Four PWM Channels
  - 8-channel, 10-bit ADC
    8 Single-ended Channels
    7 Differential Channels in TQFP Package Only
    2 Differential Channels with Programmable Gain at 1x, 10x, or 200x
  - Byte-oriented Two-wire Serial Interface
  - Programmable Serial USART
  - Master/Slave SPI Serial Interface
  - Programmable Watchdog Timer with Separate On-chip Oscillator
  - On-chip Analog Comparator
- Special Microcontroller Features
  - Power-on Reset and Programmable Brown-out Detection
  - Internal Calibrated RC Oscillator
  - External and Internal Interrupt Sources
  - Six Sleep Modes: Idle, ADC Noise Reduction, Power-save, Power-down, Standby and Extended Standby
- I/O and Packages
  - 32 Programmable I/O Lines
  - 40-pin PDIP, 44-lead TQFP, and 44-pad QFN/MLF
- Operating Voltages
  - 2.7 - 5.5V for ATmega32L
  - 4.5 - 5.5V for ATmega32
- Speed Grades
  - 0 - 8 MHz for ATmega32L
  - 0 - 16 MHz for ATmega32
- Power Consumption at 1 MHz, 3V, 25°C for ATmega32L
  - Active: 1.1 mA
  - Idle Mode: 0.35 mA
  - Power-down Mode: < 1 µA

8-bit AVR® Microcontroller with 32K Bytes In-System Programmable Flash

ATmega32
ATmega32L

Summary

Slika 11.1. Specifikacija mikrokontrolera Atmel AVR ATMega32

12. Literatura

[1] Laboratorijske vježbe iz računalne grafike, Željka Mihajlović

[2] Petar Biljanović, „Elektronički sklopovi“, Školska Knjiga, Zagreb, 2001.

[3] Aleksandar Szabó, „Impulsna i digitalna elektronika“, Zagreb 2000.

[4] Siniša srbljić, „Jezični procesori 1“, Element, Zagreb 2003.

[5] Gabro Smiljanović, „Osnove digitalnih računala“, Školska knjiga, Zagreb

[6] Elezović Neven, Žubrinić Darko, Aglić: „Linearna algebra“, Element, Zagreb 2009.

[7] Ribarić Slobodan: „Naprednije arhitekture mikroprocesora“, Element, Zagreb

[8] Glut specifikacija:
http://www.opengl.org/documentation/specs/glut/spec3/spec3.html

[9] http://en.wikipedia.org/

[10] Forum AVR entuzijasta:
http://www.avrfreaks.net/

Zahvale

Zahvaljujem svojoj mentorici prof.dr.sc. Željki Mihajlović na savjetima, vodstvu i entuzijazmu kojeg je pokazala pri izradi ovog diplomskog rada.

Hvala Dragi Markoviću što je svojom inovativnošću, idejama, iskustvom, te širokim znanjem elektronike, koje je nesebično dijelio, dao velik doprinos realizaciji ovog projekta.

Zahvaljujem i Ivici Štritofu, na njegovim inovativnim izvedbama najsloženijih mehaničkih rješenja, bez kojih ne bi bilo moguće implementirati ovaj proizvod.

Posebnu zahvalu upućujem Senadu Galijaševiću koji je svojom pedantnošću, idejama i iskustvom, dao doprinos razvoju svakog dijela ovog proizvoda, koji bez njega ne bi ugledao svjetlost dana.

Hvala Matijasu Antunoviću, koji mi je svojim savjetima i uputama, olakšao rad sa AVR mikrokontrolerima.

Hvala Jeleni Galijašević na fotografiranju dizalice za potrebe ovoga rada.

Zahvaljujem i gospodinu Slobodanu Staniću, vlasniku obrta „Tiplon“, na izradi vrhunskih tiskanih pločica, kao i na specijalnom studentskom popustu pri izradi tiskanih pločica za ovaj diplomski rad.

Zahvaljujem gospodinu Sergiu Fabrisu, djelatniku slovenske tvrtke „RLS d.o.o.“, koji je svoj entuzijazam prema ovom projektu iskazao poklanjanjem prvog magnetskog senzora rotacije.

Hvala Ivoru Feriću na stručnim savjetima glede poboljšanja ovog projekta u budućnosti.

Hvala Marti Krnić, na besprijekornim prijevodima dijelova ovoga rada na engleski jezik.

Sažetak

(Simulacija upravljanja i ostvarivanje prikaza kinematičkog sustava)

Rad opisuje razvoj elektroničkog uređaja za automatizirano upravljanje robotskom glavom dizalice za kameru. Navedene su i opisane korištene tehnologije. Prikazane su sheme povezivanja elektroničkih komponenata. Dana je teorijska i matematička podloga algoritmima koje koristimo u implementaciji programske podrške sustava. Napravljen je i računalni model sustava koristeći programski jezik C++ i grafički standard OpenGL.

Ključne riječi: kinematičke strukture, automatizirano upravljanje, ciljanje objekata, kontrola istosmjernih motora, PID kontroler, aplikacije u stvarnom vremenu.

Abstract

(Control simulation and the implementation of the kinematic system representation)

The thesis describes the development of an electronic device for automatized control of the robotic head of the video camera elevator, as well as the methods and technologies used in the process. Electronic components' schemes and theoretical and mathematical elaboration of the algorithms used in implementation of the system's software are also provided. A computer model of the system was also made and it was developed using C++ programming language together with the OpenGL graphics standard.

Keywords : kinematic structures, automatized control, aiming objects, DC motor control, PID controller, real time applications