Razvoj upravljačke jedinice za 3-osnu glodalicu

Kušec, Krunoslav

Master's thesis / Diplomski rad

2023

Degree Grantor / Ustanova koja je dodijelila akademski / stručni stupanj: **University North / Sveučilište Sjever**

Permanent link / Trajna poveznica: https://urn.nsk.hr/urn:nbn:hr:122:228819

Sveučilište
Sjever

Završni rad br. 101/STR/2023

Razvoj upravljačke jedinice za 3-osnu glodalicu

Krunoslav Kušec, 0314009971

Varaždin, listopad 2023. godine

Sveučilište
Sjever

Diplomski sveučilišni studij Strojarstvo

Diplomski rad br. 101/STR/2023

Razvoj upravljačke jedinice za 3-osnu glodalicu

Student

Krunoslav Kušec, 0314009971

Mentor

doc. dr. sc. Matija Bušić, dipl. ing. stroj.

Varaždin, listopad 2023. godine

Sveučilište Sjever
Sveučilišni centar Varaždin
104. brigade 3, HR–42000 Varaždin

Prijava diplomskog rada

Definiranje teme diplomskog rada i povjerenstva

ODJEL: Odjel za strojarstvo

STUDIJ: diplomski sveučilišni studij Strojarstvo

PRISTUPNIK: Krunoslav Kušec | JMBAG: 0314009971

DATUM: 02.10.2023. | KOLEGIJ: Suvremene proizvodne tehnologije

NASLOV RADA: Razvoj upravljačke jedinice za 3-osnu glodalicu

NASLOV RADA NA ENGL. JEZIKU: Development of a control unit for a 3-axis milling machine

MENTOR: dr. sc. Matija Bušić | ZVANJE: docent

ČLANOVI POVJERENSTVA:

1. doc dr. sc. Jasna Leder Horina, predsjednica povjerenstva
2. doc. dr. sc. Matija Bušić, mentor, član povjerenstva
3. doc. dr. sc. Zlatko Botak, član povjerenstva
4. doc. dr. sc. Tomislav Veliki, zamjenski član
5.

Zadatak diplomskog rada

BROJ: 101/STR/2023

OPIS:

U teoretskom dijelu diplomskog rada, na osnovi podataka iz literature, potrebno je opisati vrste i način rada upravljačkih uređaja na CNC alatnim strojevima. Potrebno je opisati i usporediti mogućnosti CNC glodaćih obradnih centara s obzirom na broj upravljanih osi gibanja alata i/ili obratka. Potrebno je definirati standardne upravljačke dijelove i sklopove koji omogućuju rada CNC glodaći obradnih centara. Potrebno je definirati funkciju interpretera, interpolatora i povratne veze u procesu rada upravljanja CNC glodaćeg obradnog centra.
U eksperimentalnom dijelu diplomskog rada potrebno je preporučiti izvedbu i izraditi model funkcionalnog CNC glodaćeg obradnog centra sa minimalno 3 sinkronizirane radne osi u proizvoljnom računalnom programu za modeliranje. Izraditi simulaciju rada preporučenog modela te usporediti sa drugim mogućim rješenjima izvedbe. Izraditi proceduru implementacije i ispitivanja razvijenog sustava na stvarnom CNC glodaćem obradnom centru. Donijeti vlastiti zaključak o razvijenom sustavu i proceduri implementacije. U radu je potrebno navesti korištenu literaturu i eventualno dobivenu pomoć.

ZADATAK URUČEN: 02.10.2023. | POTPIS MENTORA: M. Bušić

REPUBLIKA HRVATSKA
SVEUČILIŠTE SJEVER
KOPRIVNICA

Predgovor

Izjavljujem da sam diplomski rad izradio samostalno koristeći znanja stečena tijekom studija i uz literaturu navedenu u radu. Zahvaljujem se mentoru doc. dr. sc. Matiji Bušiću na iskazanoj pomoći i stručnim savjetima tijekom izrade diplomskog rada.

Na kraju, najviše se zahvaljujem svojoj obitelji na strpljenju i bezuvjetnoj podršci tijekom cjelokupnog studija.

Krunoslav Kušec

Sažetak

Razvoj upravljačke jedinice za 3-osnu glodalicu je kompleksan inženjerski zadatak koji uključuje integraciju hardvera, softvera i mehaničkih komponenti. Razvoj počinje s razumijevanjem funkcionalnih zahtjeva glodalice [1]. U ovom slučaju, 3-osna glodalica ima sposobnost da se kreće u tri osi: X, Y i Z. Svaka os kontrolirana je zasebnim motorom. Upravljačka jedinica, koja se često naziva i kontroler, mora biti u stanju da precizno upravlja tim motorima kako bi se postigao željeni rezultat obrade.

Hardverski elementi upravljačke jedinice uključuju mikrokontroler ili mikroprocesor, servo upravljač (*engl. driver*), senzore za povratne informacije itd [2]. Mikrokontroler je "mozak" upravljačke jedinice, on interpretira ulazne signale i šalje upute motorima preko upravljača. Senzori pružaju povratne informacije o položaju i brzini pomicanja svake osi, omogućavajući mikrokontroleru da vrši korekcije kako bi osigurao precizno pozicioniranje i vođenje alata. Softverski dio razvoja uključuje programiranje mikrokontrolera. Ovo uključuje implementaciju algoritama za interpretaciju G-koda, univerzalnog jezika za CNC strojeve, interpolatora te kontrolu motora i obradu signala sa senzora [3]. Za potrebe upravljanja i testiranja rada algoritama razvijeno je korisničko sučelje za upravljanje strojem, što je ovdje i učinjeno. Mehaničke komponente uključuju same motore, remenske prijenose i druge komponente koje omogućavaju pogone sve tri osi. Njihov izbor i dizajn su od ključne važnosti za preciznost i pouzdanost stroja.

Kroz iterativni proces dizajna, testiranja i optimizacije, razvija se upravljačka jedinica koja omogućava precizno, pouzdano i efikasno upravljanje 3-osnom glodalicom. Ova vrsta razvoja zahtijeva interdisciplinarni pristup, uključujući znanja i iskustvo iz područja elektronike, računarstva, strojarstva i sl [1]. Glavni zahtjev koji je stavljen pri izradi je implementacija algoritama upravljačke jedinice čiji razvoj je opisan od najniže razine i upravljanje servo sustavom starije proizvodnje koji nema mogućnosti i izbor komunikacijskih protokola kao moderni sustavi, što dodatno otežava sam postupak.

Teorijski dio rada opisuje povijesni razvoj, vrste i načine upravljanja CNC upravljačkih jedinica. Dan je pregled i opis CNC glodalica, te, usporedba mogućnosti CNC glodalica ovisno o broju upravljanih osi. Navedeni su CNC strojevi s obzirom na broj upravljanih osi gibanja alata i/ili obratka. Nadalje, opisani su interpreter, interpolator i petlje povratne veze.

Eksperimentalni dio zaokružuje teorijski dio rada, uz detaljniji opis razvoja interpretera, interpolatora, funkcioniranja i testiranja petlji povratnih petlji i sl. Opisana je simulacija koja prikazuje realni stroj [4] modeliran u programskome alatu za 3D CAD modeliranje [5] i ubačen u programski alat za simuliranje robotskih sustava [6]. Opisano je projektiranje procedure implementacije i testiranja razvijenog sustava na realnom CNC stroju [4].

Na samom kraju rada, napisan je komentar i zaključak na dobivene rezultate eksperimenta. Eksperiment se sastoji od testiranja, odnosno puštanja u rad realnog servo motora, upravljanog pomoću vlastitog razvijenog upravljanja [7]. Prikazano je razvijeno vlastito grafičko sučelje [7] (*engl. GUI - Graphical User Interface*) za prikaz odziva servo motora na zadane ulazne vrijednosti [7]. Također su stvoreni algoritmi kojima se dokazuje, unutar simulacijog okruženja Coppelia Robototics [6], rad istih. U simulacijski alat je ubačen modelirani CNC stroj, istovjetan realnome stroju, na koji je uspješno implementirana razvijena upravljačka jedinica [8].

Prikaz uspješnosti rada testirane upravljačke jedinice prikazano je referencom [7] ili linkom: https://drive.google.com/drive/folders/1Cq4svDRRnwjmpJoTMTzkUB8ATyP4E7f2?usp=drive_link

Ključne riječi: glodanje, 3-osna glodalica, G-kod, M-funkcije, mikrokontroler, servo motor, GUI, simulacija, HMI.

Summary

Development of a control unit for the 3-axis milling machine is a complex engineering task involving the integration of hardware, software and mechanical components in one system. It begins with understanding of the functional requirements of the milling machine [1]. In this case, the 3-axis milling machine has the ability to move in axis directions: X, Y and Z. Each axis is driven by a separate motor. The control unit, often called a controller, must be able to precisely control these motors in order to achieve the desired machining result.

The hardware elements of the control unit include a microcontroller or microprocessor, electronic circuits for controlling the operation of the motor (driver units), sensors for feedback informations and other [2]. The microcontroller is the main part of the control unit, it interprets the input signals and sends instructions to the motors via the driver units. The sensors provide feedback information on the position and speed of movement of each axis, allowing the microcontroller to make corrections to ensure precise positioning. The software part of the development involves programming the microcontroller. This includes the implementation of algorithms for the interpretation of G-code, the universal language for CNC machines, and motor control with sensor signal processing [3]. Also, a user interface for controlling the machine was developed. Mechanical components include the motors, gears and other components that enable the milling machine to be driven in three axes, precisely. Choice and design of these components are of crucial importance for the overall precision and reliability of the CNC machine.

Through an iterative process of design, testing and optimization, a control unit is developed that enables precise, reliable and efficient control of the 3-axis milling machine. This type of development requires an interdisciplinary approach, including knowledge and experience in the fields of electronics, computing and mechanical engineering [1].

The main requirement set during the creation is the implementation of the control unit algorithms whose development is described from the lowest level, and the management of an older production servo system that lacks capabilities and choice of communication protocols like modern systems, which further complicates the process

The theoretical part of the work describes the historical development, types and ways of controlling CNC control units. An overview and description of CNC milling machines is given, as well as a comparison of the possibilities of CNC milling machines depending on the number of controlled axes. Complete list of CNC machines with regard to the number of controlled axes of movement of the tool and/or workpiece are shown. Furthermore, the interpreter, interpolator and feedback loops are described in this thesis.

The experimental part completes the theoretical part of the work, with a more detailed description of the development of the interpreter, interpolator, functioning and testing of feedback loops. A simulation is described with a real machine [4], modeled in a software tool for 3D CAD modeling [5] and inserted into the software tool for simulating mechatronic machines and systems [6]. The design of the procedure of implementation and testing of the developed system on a real CNC machine is described [4].

At the very end of the thesis, a comment and conclusion is written on the obtained results of the experiment. The experiment consists of testing, i.e. putting into operation, a real servo motor, controlled using a self-developed control [7]. Developed graphical interface [7] is shown for presenting the response of the servo motor to the given input values [7]. Algorithms were also created to prove their operation within the Coppelia Robotics [6] simulation environment. A modeled CNC machine, identical to the real machine, was imported into the simulation tool, on which the developed control unit was successfully implemented [8].

Keywords: milling, 3-axis milling machine, G-code, M-functions, microcontroller, servo motor, GUI, simulation, HMI.

Popis korištenih kratica

AC-DC	izmjenična struja - istosmjerna struja struja (engl. Alternating Current – Direct Current)
API	Programsko sučelje aplikacije (engl. Application Programming Interface)
APT	automatski programirani alat (engl. Automatically Programmed Tool)
Acc/Dec	ubrzanje/usporavanje (engl. Acceleration/Deceleration)
DNC	izravno numeričko upravljanje (engl. Direct Numerical Control)
CCW	suprotno smjeru gibanja kazaljke na satu (engl. Counter Clock Wise)
CW	u smjeru gibanja kazaljke na satu (engl. Clock Wise)
CAD	računalno potpomognut dizajn (engl. Computer-Aided Design)
CAI	računalno potpomognuta inspekcija (engl. Computer-Aided Inspection)
CAM	računalno potpomognuta proizvodnja (engl. Computer-Aided Manufacturing)
CAPP	računalno potpomognuto planiranje procesa (engl. Computer-Aided Process Planning)
CNC	računalno numeričko upravljanje (engl. Computer Numerical Control)
GUI	grafičko korisničko sučelje (engl. Graphical User Interface)
HMI	sučelje čovjeka i stroja (engl. Human Machine Interface)
FIFO	prvi-ušao-prvi-izašao (engl. First-In-First-Out)
FMS	fleksibilni proizvodni sustav (engl. Flexible Manufacturing System)
FBD	funkcijski blok dijagram (engl. Function Block Diagram)
SISO	jedan ulaz jedan izlaz (engl. Single Input Single Output)
RPM	okretaja u minuti (engl. Revolutions Per Minute)
PID	proporcionalni integralni derivacijski (engl. Proportional Integral Derivative)
VME	okruženje virtualnog stroja (engl. Virtual Machine Environment)
S/W	softver (engl. Software)
TPG	generiranje putanje alata (engl. Tool Path Generation)
FPGA	niz vrata koji se može programirati poljem (engl. Field-Programmable-Gate Array
PC-based	kontroler integriran (temeljen) na računalu (engl. Personal Computer-Based)
PLC	programirajući logički upravljač (engl. Programmable Logic Controller)
DSP	procesor digitalnog signala (engl. Digital Signal Processor)

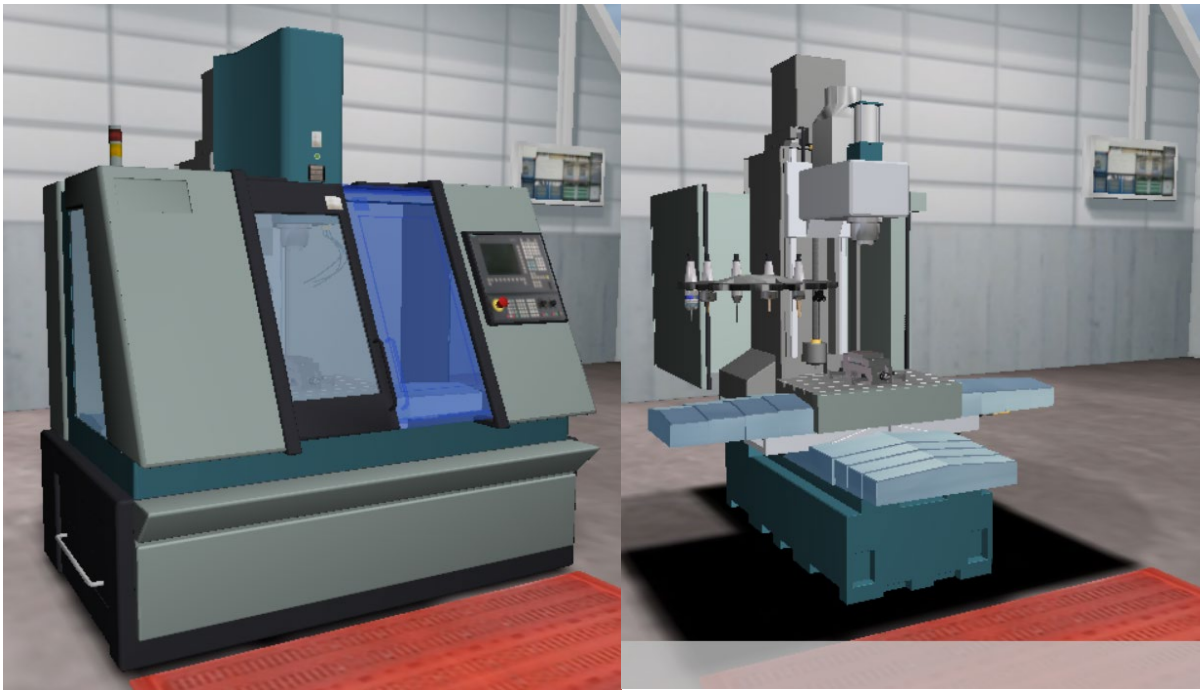
Popis oznaka

X	Pomak po X osi
Y	Pomak po Y osi
V	Brzina osi – općenito brzina
F	Brzina osi – posmak
t	Vrijeme
F_1	Brzina prvog profila
F_2	Brzina drugog profila
T_{A1}	Vrijeme ubrzanja prvog profila brzina
T_{C1}	Vrijeme konstantne brzine prvog profila brzina
T_{D1}	Vrijeme usporenja prvog profila brzina
T_{A2}	Vrijeme ubrzanja drugog profila brzina
T_{C2}	Vrijeme konstantne brzine drugog profila brzina
T_{D2}	Vrijeme usporenja drugog profila brzina
R	Radijus gibanja između dva profila
V_x	Brzina gibanja po X osi
V_y	Brzina gibanja po Y osi
T_a	Period ubrzanja
T_c	Period konstantne brzine
T_d	Period usporenja
a	ubrznja i usporenje
t_1	Vrijeme do kraja ubrzanja
t_2	Vrijeme do kraja konstantne brzine
t_3	Vrijeme do kraja usporenja
p_1	Prijeđeni put do t_1
p_2	Prijeđeni put do t_2
p	Prijeđeni ukupni put do t_3
w_1	Kutna brzina segmenta ubrzanja
w_2	Kutna brzina segmenta konstantne brzine
w_3	Kutna brzina segmenta usporenja

Sadržaj

1. Uvod

Tema ovog rada je razvoj upravljačke jedinice za 3-osnu glodalicu. To uključuje nekoliko ključnih koraka. Jedan od važnijih, tj. glavnih koraka u razvoju upravljačke jedinice je svakako analiza zahtjeva stavljenih pred projektanta. Prvi korak je analiza zahtjeva i potreba za upravljanjem 3-osne glodalice [1].



Slika 1.1 CNC vertikalna glodalica s 3 osi (prikaz sa oklopom-lijevo i bez oklopa-desno) [3]

Ovo uključuje identifikaciju funkcionalnosti koje će upravljačka jedinica morati podržavati, kao što su kretanje alata i/ili radnog stola u tri osi, upravljanje brzinama, sigurnosne mehanizme, sučelje za rad s operaterom (*GUI* engl. *Graphical User Interface*) i slično. Nakon analize zahtjeva, slijedi projektiranje elektroničkog sklopa upravljačke jedinice. To uključuje odabir odgovarajućih komponenti kao što su upravljački mikrokontroler, servo upravljači motora, senzori, prekidači, sučelja za rad s računalom itd. Elektronički sklop treba biti dizajniran tako da podržava potrebne funkcionalnosti i interakcije s mehaničkim dijelovima glodalice. Iako kod razvoja jednog CNC stroja treba obuhvatiti sve spomenute elemente, u ovom slučaju s obzirom na već postojeći stroj sa mehanički ispravnim komponentama zahtjeva izradu i prilagodbu upravljačke jedinice. Takav zahtjev prilagodbe za jedan sustav je u mnogo slučajeva kompleksniji nego izrade iz početka.

Početni korak je razvoj upravljačkog softvera koji će upravljati radom glodalice. To uključuje pisanje koda koji će interpretirati G-kod naredbe, interpolirati putanju alata i kontrolirati motore

za kretanje alata i/ili radnog stola u tri osi, upravljati brzinama, obraditi ulazne signale s prekidača i senzora, te osigurati sigurnosne mehanizme kao što su zaustavljanje u hitnim situacijama i sl. Da bi rad sa strojem bio što jednostavniji, bitno je imati sučelje za komunikaciju stroja s operaterom. Svrha sučelja nije samo interakcija sa strojem na operativnoj razini, u smislu općenitog rada sa CNC strojem. Interakcija i komunikacija služe i dodatno za potrebe prikupljanja informacija potrebnih za ispitivanje i proučavanje rada razvijenog programa. Upravljačka jedinica treba imati sučelje koje omogućuje unos naredbi, praćenje statusa operacija i pružanje povratnih informacija. Sučelje može biti u obliku tipkovnice, ili, zaslona osjetljivog na dodir. Operater bi trebao moći upisati G-kod naredbe ili koristiti druge naredbe za kontrolu glodalice, kao što su npr. M-kod naredbe.

Slika 1.2 CNC upravljačka jedinica Siemens Sinumerik 840D [3]

Nakon razvoja upravljačke jedinice, slijedi ispitivanje i optimizacija. U ovom koraku provjeravaju se funkcionalnosti upravljačke jedinice kako bi se osiguralo da ispravno interpretira i izvršava G-kod naredbe, pravilno kontrolira gibanja motora, pruža očekivane rezultate obrade i odgovara na ulazne naredbe i signale. Kada je upravljačka jedinica testirana i optimizirana, slijedi integracija s mehaničkim dijelom glodalice. Izravno testiranje algoritama rada upravljačke jedinice je opasno zbog rizika oštećenja pogonskog sustava, inicijalna provjera algoritama se provodi u simulaciji. Simulacija također pruža bolji uvid u gibanje i proučavanje gibanja osi stroja.

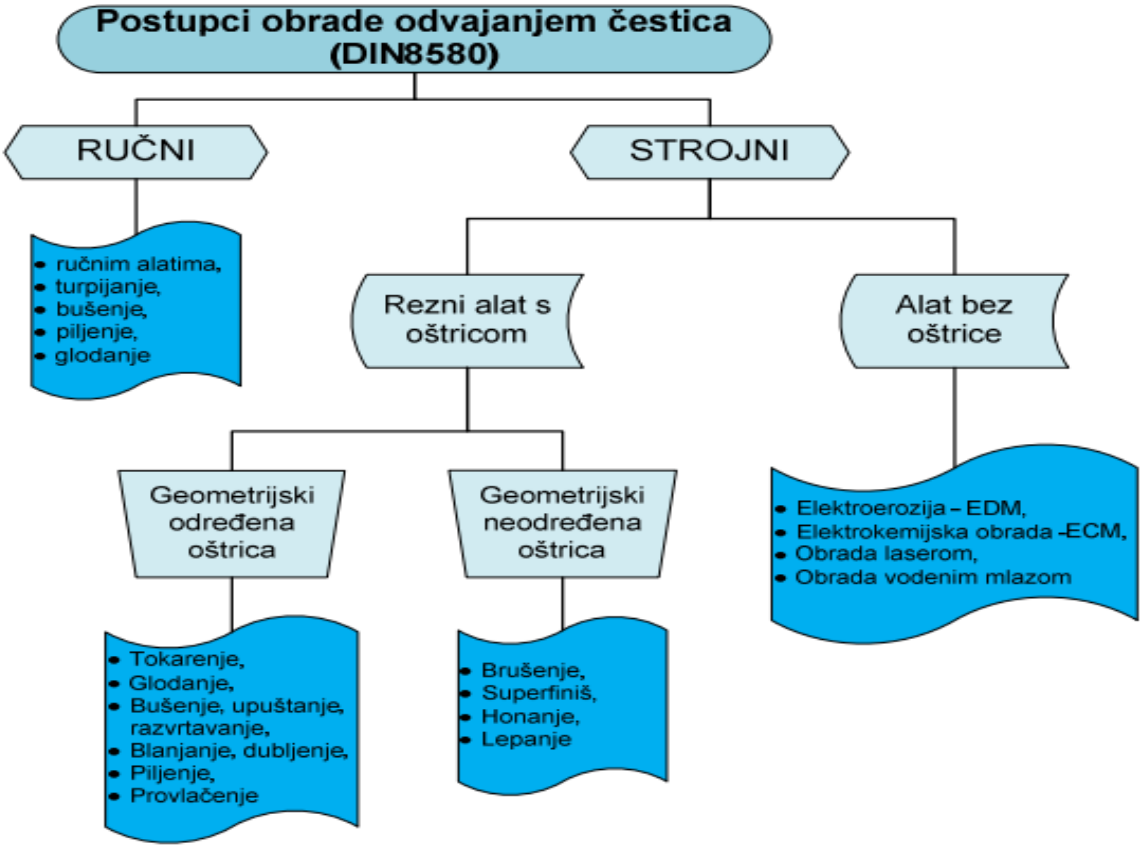
Slika 1.3 CNC vertikalna glodalica s 3 osi na koju će se implementirati razvijena upravljačka jedinica[4]

Na slici 1.3 prikazan je stvarni CNC stroj koji će kasnije poslužiti za implementaciju razvijene upravljačke jedinice.

Na [10] se može vidjeti kako to izgleda, na sličnome primjeru. Referenca se odnosi na cijelo trajanje videa, prikaz cijelog CNC sustava. Prikazani sustav je manje složenosti zbog korštenja koračnih motora koji su jednostavniji za upravljanje, te zahtjevaju manje komponenti. Ispitivanje upravljanja realnim servo motorom je prikazano ovdje [7]. Prikaz ispitivanja odziva servo motora na razvijene algoritme upravljanja[8]. Reference se odnose na cijelo trajanje video zapisa, gdje se na demonstrativan način prikazuje rad razvijenog sustava.

2. Upravljačka jedinica CNC strojeva

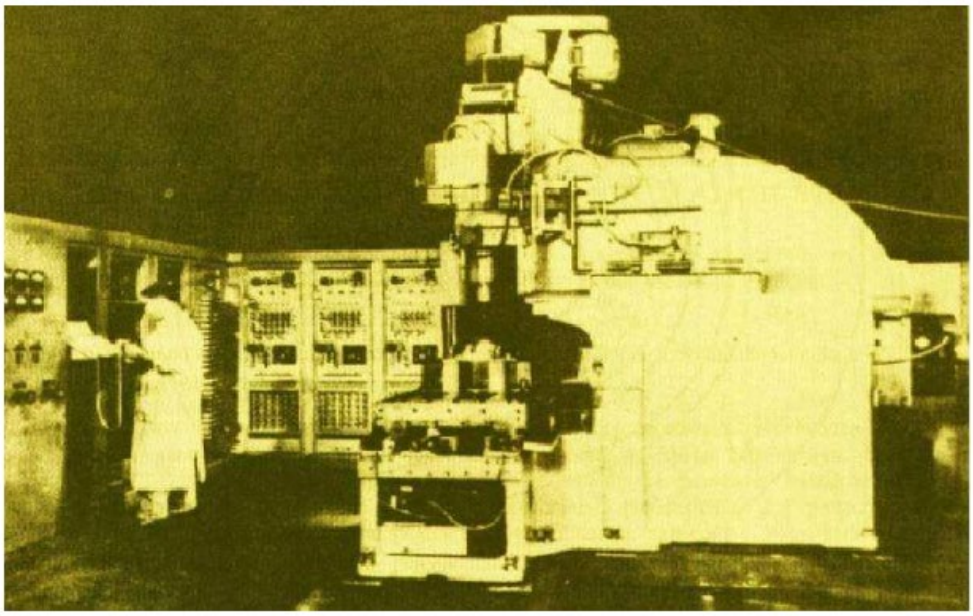
Današnja suvremena industrija zahtjeva kontinuirano ulaganje u inovacije i nadogradnju postojećih proizvodnih procesa, kao proizvodnih sustava i prateće tehnologije i na taj način kvalitete rada. Cilj svakoga proizvodnoga procesa je postizanje konkurentnosti na tržištu, brzina i kvaliteta proizvodnje, te smanjenje samih troškova proizvodnje. U proizvodnome procesu veliku ulogu imaju CNC strojevi, koji su uz pomoć računala omogućili veću konkurentnost na tržištu, bržu proizvodnju te i u određenoj mjeri i smanjenje troškova proizvodnje. CNC (*engl. Computer Numerical Control*), je računalom upravljani numerički alatni stroj koji je direktna nadogradnja starije verzije numerički upravljanog alatnog stroja NC (*engl. Numerical Control*). CNC strojevi predstavljaju automatizirane alatne strojeve čije upravljanje proizlazi uz pomoć naredbi prethodno kodiranih, a te kodirane naredbe se sastoje od kombinacije brojeva, određenih slova abecede kao i simbola zapisanih na takav način da jedinica za kontrolu strojeva može razumjeti i izvršiti zadanu naredbu. Naredbe se pišu logičkim redoslijedom u unaprijed standardiziranom obliku. Skup svih naredbi potrebnih da bi se provela određena obrada na izratku naziva se CNC program [11]. U današnjim proizvodnim sustavima se najčešće koristi CAD/CAM tehnologija (*engl. Computer Aided Design/Computer Aided Manufacture*) koja kombinira konstruiranje i razvoj proizvoda u nekom 3D alatu s programom obrade na CNC stroju, također u istom tom 3D alatu. CNC stroj spada u kategoriju alatnih strojeva, tj. strojeva za oblikovanje izrađevina od različitih materijala obradom odvajanja čestica. Iako se slična upravljanja osi koriste i kod drugih procesa kao aditivnih tehnologija, tehnologija deformacija itd. Alatni strojevi imaju mogućnost oblikovanja izrađevina na više načina: odvajanjem čestica, deformacijom, odrezivanjem, primjenom elektriciteta i ultrazvuka ili različitim kemijskim postupcima. Ono što je karakteristično za suvremene alatne strojeve, pa time i za CNC strojeve je da mogu proizvesti izrađevine s viskom ponovljivošću i točnošću [12].



Slika 2.1 Prikaz postupaka obrade odvajanjem čestica [13]

2.1. Povijest upravljačkih jedinica

Povijest numerički upravljanih alatnih strojeva seže nekoliko desetljeća unatrag, točnije značajniji napredak u ovom području je bio 1950.-ih godina, gdje je prvi alatni stroj takve vrste napravljen u Americi početkom 1950.-ih godina na prestižnom sveučilištu MIT (*engl. Massachusetts Institute of Technology*). Tada su se počeli pojavljivati prvi eksperimentalni sustavi za kontrolu pokreta alata pomoću računalnih sustava. U to vrijeme su se koristili računalni sustavi velikih dimenzija koji su bili skupi i rezervirani za industrijske primjene. Ono što je karakteristično za ove alatne strojeve je da su se tada počeli pojavljivati prvi eksperimentalni sustavi za kontrolu pokreta alata pomoću računalnih sustava.



Slika 2.2 Prvi NC stroj [11]

Važno je za naglasiti da je razvoju alatnog stroja na MIT-u prethodio razvoj drugih uređaja (kao što su sustavi za automatsko zvonjenje zvona, razvitak stroja za pletenje i tkanje upravljan bušenom vrpcom, razvoj digitalnog računala, razvoj pneumatike i hidraulike i sl.) koji su direktno i indirektno utjecali na razvoj NC-a.

Tijekom 1960-ih godina, razvoj CNC tehnologije se nastavio. Pojavljuje se direktno numeričko upravljanje gdje su upravljačke jedinice postale sofisticiranije, koristeći računalne sustave koji su imali veće kapacitete kao i mogućnost obrade s više osi. Razvoj je bio usmjeren na povećanje preciznosti i točnosti CNC strojeva. Time je proizvodni proces znatno skraćen.

U 1970.-im godinama, CNC tehnologija je postala sveprisutna u industriji. Upravljačke jedinice su postajale manje i kompaktnije, koristeći tehnologiju mikroračunala. Računalni sustavi su bili programabilni, omogućavajući fleksibilnost u izvođenju različitih operacija.

1980.-ih godina CNC tehnologija je doživjela značajan napredak. Upravljačke jedinice su postale još manje, brže i učinkovitije. Razvijeni su novi softverski alati za generiranje G-koda i simulaciju CNC operacija. Sustavi za komunikaciju s računalima postali su standardizirani, što je olakšalo integraciju CNC strojeva u proizvodne linije.

U 1990.-ima CNC tehnologija je postala još naprednija. Upravljačke jedinice su se razvijale u smjeru sve veće brzine obrade, bolje točnosti i veće pouzdanosti. Napredni softveri za programiranje i simulaciju omogućili su brže i učinkovitije postavljanje CNC operacija.

U 2000.-ima do danas upravljačke jedinice za CNC strojeve nastavile su se razvijati u korak napretkom računalnih tehnologija. Moderni CNC strojevi koriste napredne mikrokontrolere, digitalne servo pogone i sofisticirane softverske alate. Integracija senzora i sustava za nadzor

omogućuje praćenje obrade u realnom vremenu i optimizaciju obrade. Kao i korištenje sve prisutnije umjetne inteligencije.

Danas su upravljačke jedinice za CNC strojeve visoko napredne, s mogućnošću upravljanja više osi, automatizacijom promjene alata, komunikacijom s drugim strojevima i sustavima, te naprednim softverskim sučeljima za programiranje i nadzor. Napredak u CNC tehnologiji nastavlja se, s fokusom na poboljšanje produktivnosti, točnosti i automatizacije CNC strojeva.

Slika 2.3 Moderni obradni sustav [14]

2.2. Vrste upravljačkih jedinica

Upravljačka jedinica se sastoji od više konstruktivnih dijelova gdje jezgru čini računalo, odnosno procesorska jedinica koja izvršava sve logičke odluke. Postoje različite vrste upravljačkih jedinica za CNC strojeve, a izbor ovisi o vrsti stroja i specifičnim potrebama aplikacije. Također, upravljačka jedinica se nalazi na samom stroju u obliku ekrana i tipki te je potreban softver za rad CNC stroja.

Slika 2.4 Upravljačke jedinice CNC strojeva [14]

Jedna od vrsta upravljačkih jedinica je upravljačka jedinica na bazi PC-a, osobnog „standardnog“ računala. U ovom tipu upravljačke jedinice, CNC funkcionalnost se temelji na osobnom računalu tj. PC-u. Softver se instalira na PC-u, koji komunicira s vanjskim servo upravljačima i senzorima putem odgovarajućih sučelja. Upravljačke jedinice na bazi PC-a često imaju napredne mogućnosti programiranja i simulacije i obično se koriste kod jednostavnijih CNC strojeva. Iskorištava se velika procesorska moć računala te napredne mogućnosti grafičkog prikaza. U nekim slučajevima potrebna je instalacija operativnog sustava koji osigurava izvođenje operacija u stvarnom vremenu (engl. *Real Time*) pojam vezan za prekidanje svih nepotrebnih procesa i bavljenje samo procesima vezanim za upravljanje, te osiguravanje da će svaka vremenska iteracija biti izvršena u definiranom vremenskom intervalu.

Zatim postoje upravljačke jedinice na bazi mikrokontrolera. Karakteristično za ovu vrstu upravljačkih jedinica je da koriste mikrokontrolere kao središnje procesore za upravljanje CNC strojem. Mikrokontroleri su integrirani u samu upravljačku jedinicu i kontroliraju kretanje motora, obradu G-koda i upravljanje ostalim funkcionalnostima. Ovi sustavi su često kompaktni i ekonomični. Mikrokontroler kao procesorski uređaj puno lakše i pouzdanije osigurava izvođenje operacija u definiranim vremenskim intervalima.

PLC-bazirana upravljačka jedinica često se koristi za CNC strojeve s jednostavnijim zahtjevima. PLC je industrijsko računalo koje se koristi za kontrolu i automatizaciju procesa. Upravljačke jedinice na bazi PLC-a mogu biti robustne, pouzdane i lako prilagodljive. Ponekad se ovi uređaji nazivaju i PLC uređaji za gibanje (*engl. Motion PLC*). Automatizirani proces zahtjeva PLC kao centralni uređaj za donošenje odluka, korištenjem upravljanja na bazi PLC-a, jedan uređaj upravlja CNC strojem i lokalnim automatiziranim procesom koji je vezan za CNC stroj.

Upravljačka jedinica sa DSP-om (*engl. Digital Signal Processor*). DSP upravljačke jedinice koriste digitalne signale za obradu i kontrolu CNC operacija. DSP je poseban tip procesora koji je

optimiziran za brzu obradu signala. Ove upravljačke jedinice mogu pružiti visoku brzinu izvođenja i precizno kretanje motora.

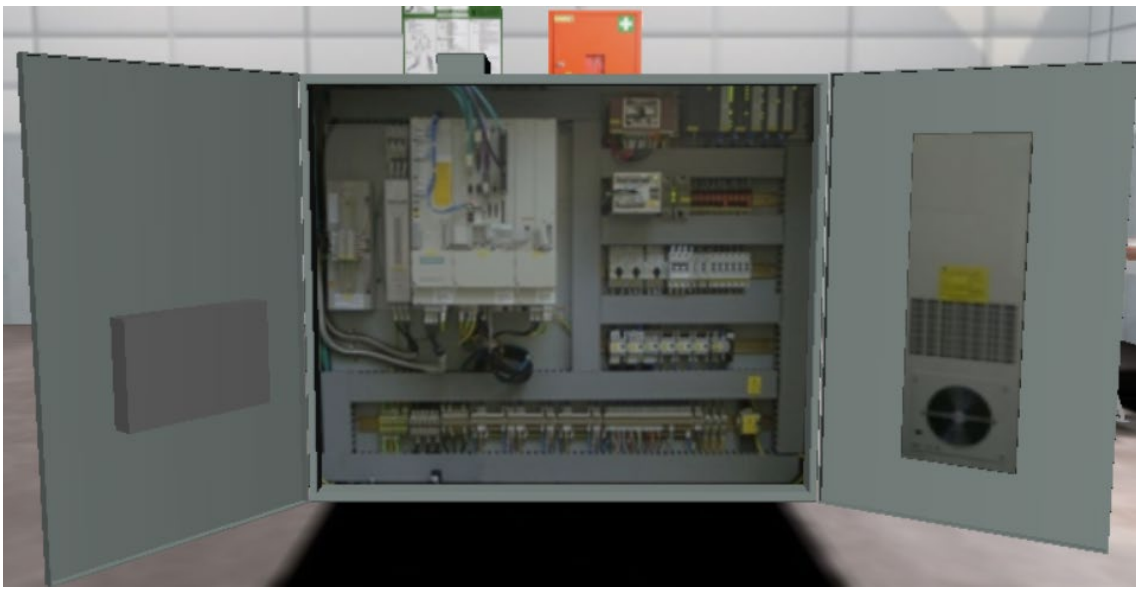
Upravljačka jedinica FPGA (*engl. Field-Programmable Gate Array*) koristi FPGA čipove koji se mogu programirati za izvršavanje specifičnih funkcija. FPGA-ovi pružaju visoku fleksibilnost i mogu se konfigurirati za podršku različitim CNC operacijama. Ove upravljačke jedinice često imaju visoku brzinu izvođenja i mogu biti vrlo prilagodljive.

Ovdje je navedeno nekoliko primjera koji prikazuju neke od glavnih vrsta upravljačkih jedinica, ali također postoji i širok spektar drugih upravljačkih jedinica koje se koriste u CNC strojevima, svaka s vlastitim karakteristikama. Odabir odgovarajuće vrste upravljačke jedinice ovisi o zahtjevima aplikacije, performansama i funkcionalnostima koje su potrebne za CNC operacije.

Bitno je naglasiti da moderne upravljačke jedinice koriste kombinacije navedenih vrsta, gdje su određeni dijelovi upravljačke jedinice čak smješteni na servo upravljač. Te je ponekad i teško razlučiti kojoj vrsti pripada pojedina upravljačka jedinica.

2.3. Način rada upravljačkih jedinica

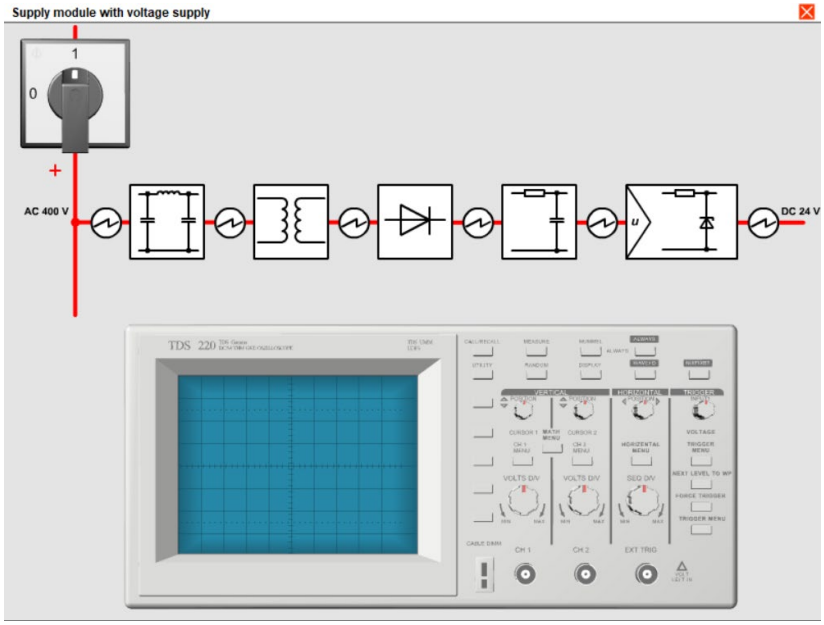
Upravljačke jedinice CNC strojeva rade na temelju unaprijed definiranih programa i naredbi za izvođenje specifičnih CNC operacija. Način rada upravljačkih jedinica obično uključuje korake kao što su pripreme programa, prijenos programa, interpretacija G-koda, kontrola motora, obrada operacija, sigurnosti i nadzora te povratnih informacija. Program CNC operacija se generira pomoću CAM softvera, a može se upisivati i ručno u upravljačku jedinicu. CAM softver pretvara nacrte ili modele u niz G-kod naredbi. G-kod sadrži informacije o kretanju alata, brzinama, pozicioniranju i ostalim parametrima potrebnih za izvođenje CNC operacija. Sljedeći korak, nakon generiranja programa, je prijenos koda koji sadrži naredbe. G-kod se prenosi na upravljačku jedinicu CNC stroja. Prijenos se obično vrši putem računalne veze, kao što su USB, Ethernet ili serijska veza.



Slika 2.5 Upravljački ormar CNC stroja [3]

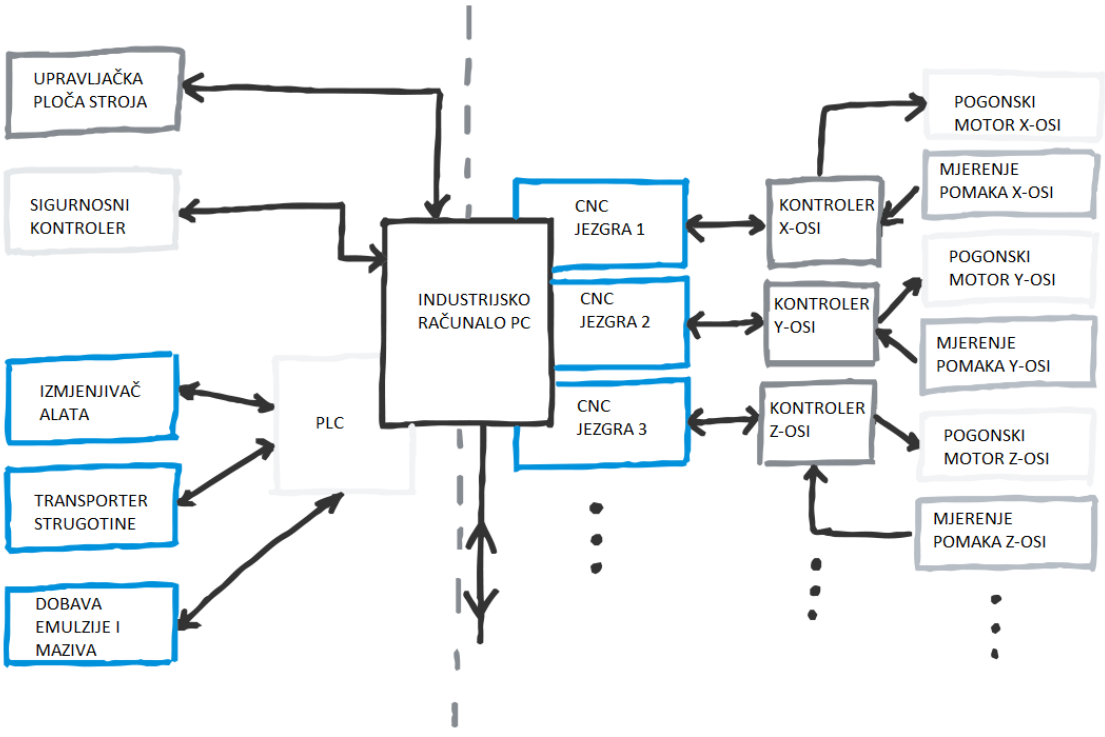
Nakon prijenosa G-koda na upravljačku jedinicu, CNC stroj je spreman izvršiti operacije prema programu. Potom, upravljačka jedinica CNC stroja interpretira G-kod program i pretvara ga u odgovarajuće naredbe za pokretanje motora, kontrolu brzine i ostale operacije. Upravljačka jedinica "čita" svaku naredbu iz G-koda i izvršava ih s obzirom na definirane parametre. Zatim na red dolazi glavna funkcija upravljačke jedinice, a to je kontroliranje motora stroja kako bi se postigli željena gibanja alata ili stola stroja. Ovisno o CNC stroju, mogu se koristiti različite vrste motora, poput servo motora ili koračnih motora. Tijekom izvođenja CNC operacija, upravljačka jedinica kontrolira kretanje alata ili stola stroja u skladu s G-kod programom. To uključuje linearno ili kružno kretanje, promjene brzina, pozicioniranje na određene točke i druge definirane radnje.

Sigurnost i nadzor su izrazito bitni u radu CNC stroja. Upravljačka jedinica također nadzire sigurnosne aspekte CNC operacija. To može uključivati zaštitne mehanizme kao što su hitni zaustavi, prepoznavanje prepreka, zaštita od sudara i slično.



Slika 2.6 Blokovska shema napajanja stroja sa osciloskopom za testiranje [3]

Upravljačka jedinica osigurava da se operacije izvode sigurno i prema postavljenim parametrima. Povratne informacije i sučelje za komunikaciju s operaterom su također pod kontrolom upravljačke jedinice. Ona pruža povratne informacije o statusu operacija i stanju stroja. To može uključivati informacije o brzinama, pozicioniranju, izvršenim operacijama i drugim relevantnim podacima.



Slika 2.7 Shematski prikaz načina rada upravljačkog stroja [14]

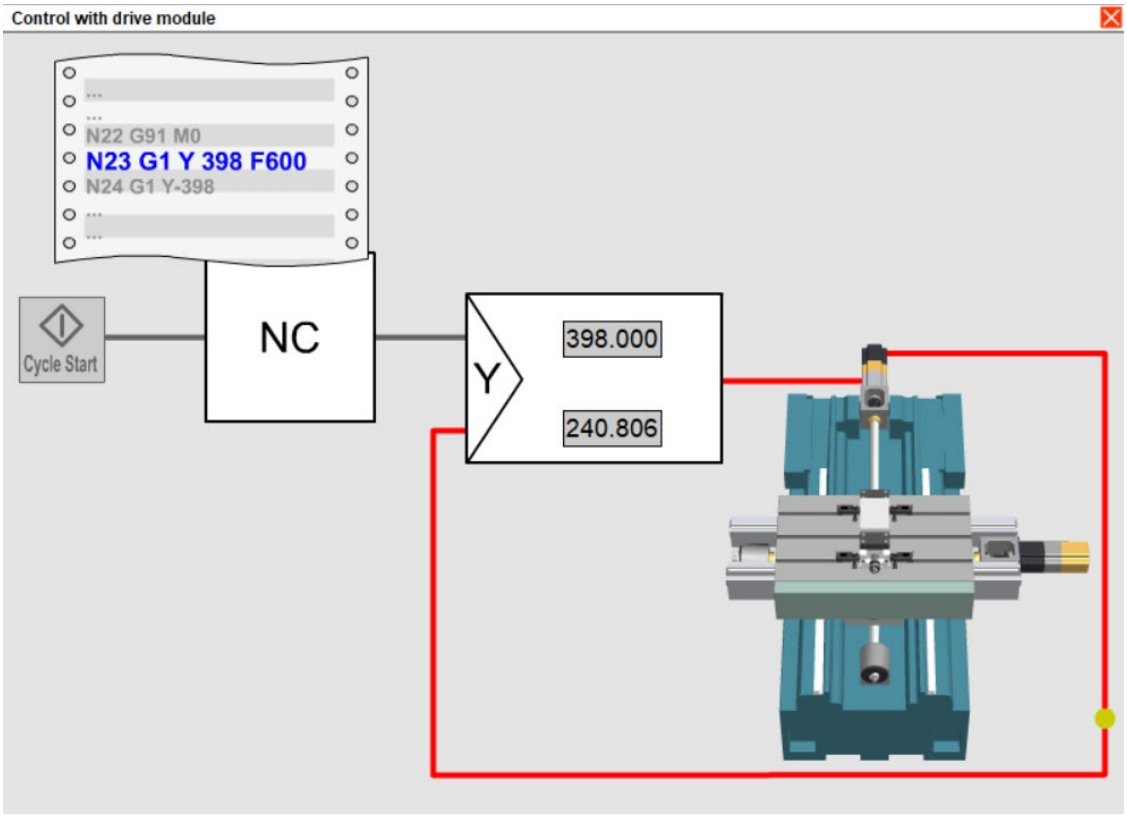
3. CNC glodalice

CNC (*engl. Computer Numerical Control*) glodalice su strojevi koji se koriste za obradu materijala, poput metala, plastike, drva ili kompozitnih materijala. Glodalice koriste rotirajući alat s više reznih oštrica kako bi uklonile materijal i oblikovale radni komad prema željenom dizajnu.

3.1. Opis CNC glodalica

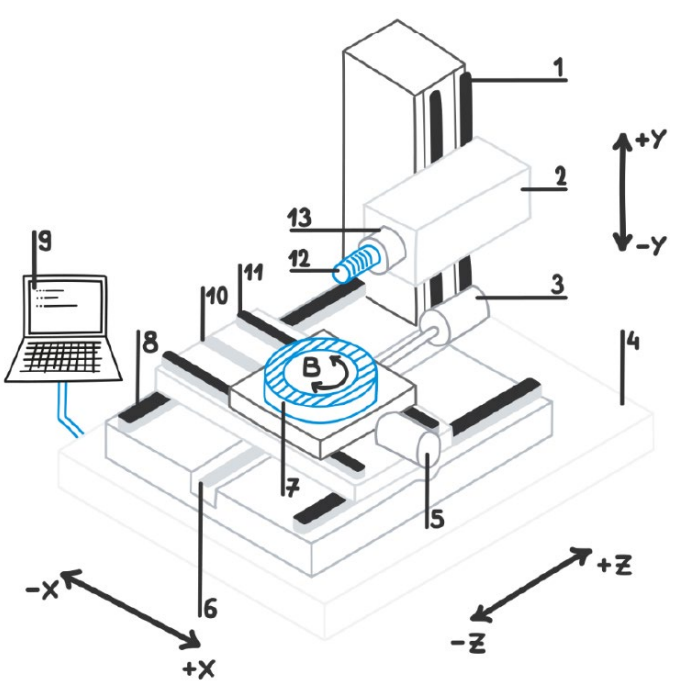
Neke od ključnih značajki CNC glodalica su nabrojane u nastavku.

- **Numeričko upravljanje**: CNC glodalice koriste numeričko upravljanje kako bi precizno kontrolirale kretanje alata i stola. Upravljačka jedinica čita G-kod program i pretvara ga u odgovarajuće pokrete i operacije [14].



Slika 3.1 Kontrolna petlja NC stroja s prikazom stroja i naredbi [3]

- **Više osi:** većina CNC glodalica je višeosna, što znači da mogu pokretati alat i/ili obradak u smjeru tri ili više osi. Najčešće se koriste troosne glodalice (X, Y, Z), ali postoje i višeosne glodalice koje omogućuju kretanje alata u dodatnim rotacijskim osima, kao što su osi A, B i C [14].



1	Vodilica Y-osi
2	Y-klizač
3	Pogon posmaka osi Z
4	Okvir stroja
5	Pogon posmaka osi X
6	Vijak za pomicanje osi Z
7	Strojni stol (rotirajući; B-os
8	Vodilica Z-osi
9	Kontrolor
10	Vijak za pomicanje osi X
11	Vodilica X-osi
12	Alat za glodanje
13	Glavno vreteno

Slika 3.2 Prikaz osi i pripadnih dijelova CNC stroja [14]

- **Različite operacije glodanja**: CNC glodalice mogu izvesti različite operacije glodanja, uključujući ravno glodanje, spiralno glodanje, kružno glodanje, konturno glodanje i graviranje. Ovisno o alatu i parametrima obrade, mogu se postići različiti oblici, reljefi i detalji na radnom komadu [3].

Slika 3.3 Čeono glodalo promjera 63 mm [2]

Slika 3.4 Alat za glodanje utora promjera 20 mm [2]

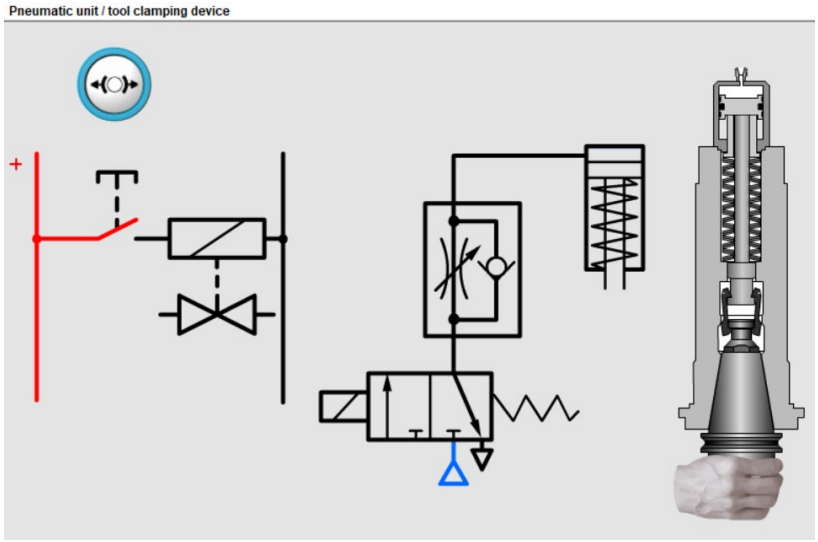
Slika 3.5 Alat za glodanje utora promjera 10 mm [2]

Slika 3.6 Alat za bušenje promjera 8.5 mm [2]

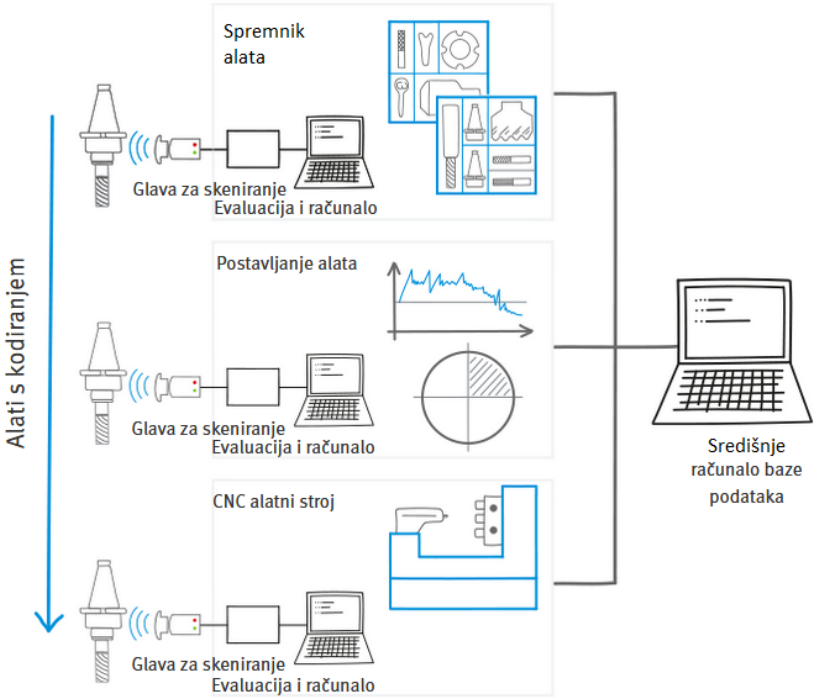
Slika 3.7 Alat za urezivanje navoja M10 [2]

Slika 3.8 Mjerna sonda [2]

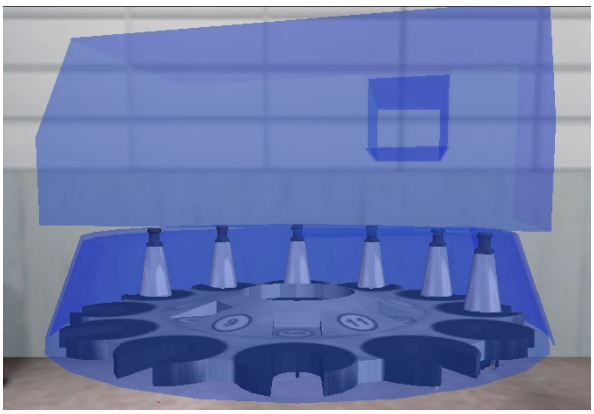
- **Automatska promjene alata**: CNC glodalice često imaju mogućnost automatske izmjene alata. Najčešće vrste alata i uređaja koji se mogu nalaziti u izmjenivaču alata prikazane su od slike 3.3 do slike 3.8. To omogućuje stroju da izvodi više operacija bez potrebe za ručnom intervencijom. Upravljačka jedinica može upravljati promjenom alata prema potrebama programa [3].



Slika 3.9 Pneumatska shema mehanizma za izmjenu alata [14]



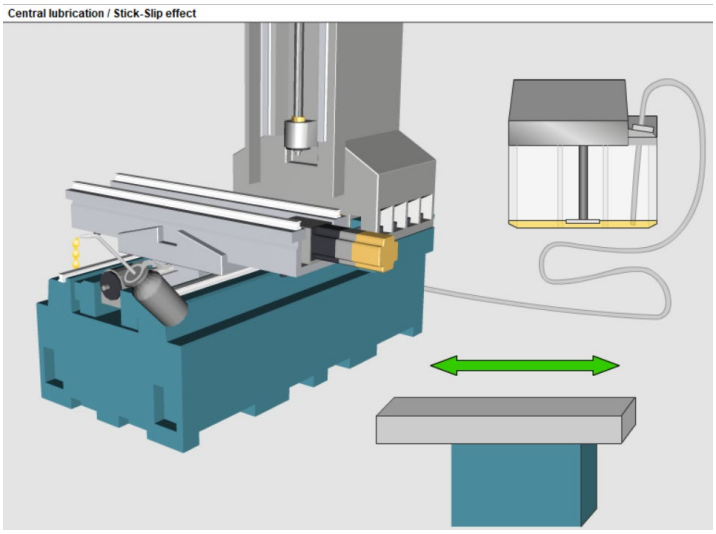
Slika 3.10 Blokovna shema potpunog mehanizma za izmjenu alata [14]



Slika 3.11 Prikaz sjene kučišta nosača alata – revolvera [2]

Slika 3.12 CAD model nosača alata – revolvera [3]

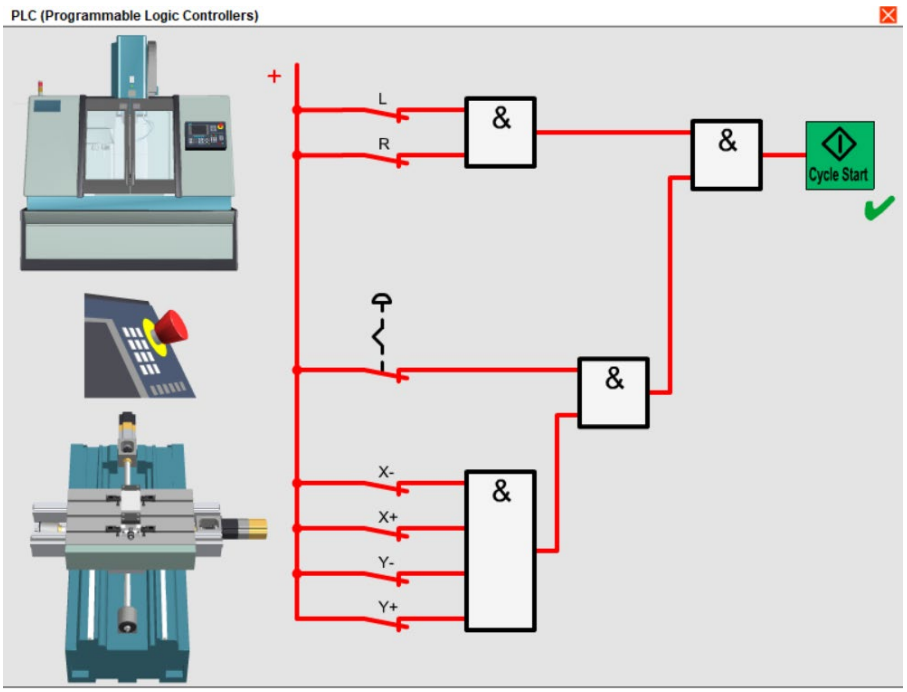
- **Integrirano hlađenje i podmazivanje**: Mnoge CNC glodalice imaju integrirane sustave hlađenja i podmazivanja kako bi olakšale obradu. SHIP ili sredstva za hlađenje i podmazivanje koriste se kako bi se smanjila toplina i poboljšalo uklanjanje strugotine. Podmazivanje osigurava optimalne uvjete rada alata i produžuje njegov vijek trajanja [3].



Slika 3.13 Prikaz sustava za podmazivanje CNC stroja [3]

- **Napredno sučelje za komunikaciju s operaterom**: CNC glodalice obično imaju napredno sučelje za komunikaciju s operaterom, koje omogućuje unos i izmjenu G-kod programa, praćenje statusa operacija, nadzor brzine, pozicioniranja i drugih parametara. Sučelje može biti u obliku tipkovnice, zaslona osjetljivog na dodir ili računalnog sučelja [3].

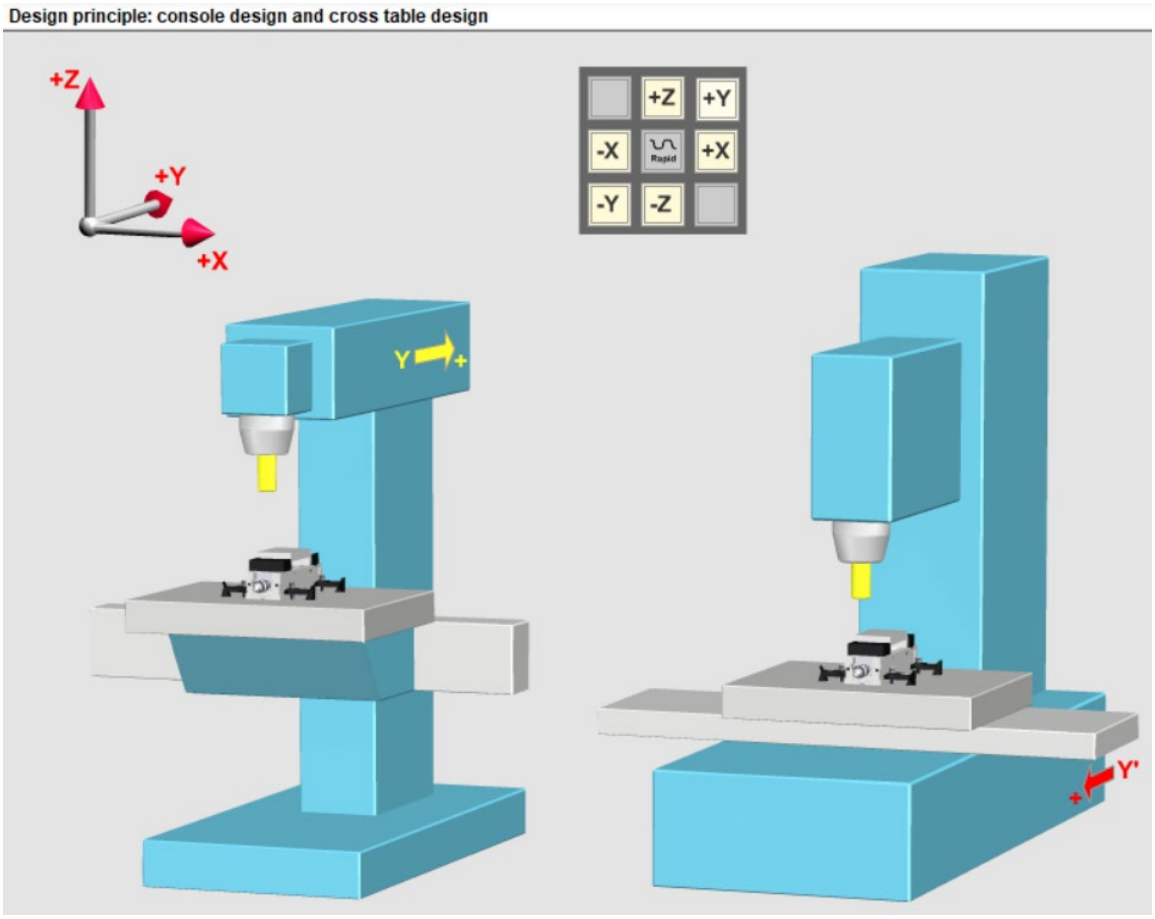
Slika 3.14 Prikaz grafičnog sučelja Siemens SINUMERIK [3]



Slika 3.15 PLC upravljačka shema CNC stroja [3]

3.1.1. Opis 3-osne CNC glodalice

3-osna CNC glodalica je tip CNC stroja koji omogućuje kretanje alata u tri osi: X, Y i Z.



Slika 3.16 Prikaz osi primjenjivih na radni stol CNC stroja [3]

Kombinacija kretanja u svim tri osi omogućuje CNC glodalici izvođenje različitih operacija glodanja, poput ravnog glodanja, konturnog glodanja, spiralnog glodanja i graviranja. Preciznost i ponovljivost kretanja u svakoj osi omogućuju postizanje željenih oblika i detalja na radnom komadu [3].

3.2. Usporedba mogućnosti CNC glodaćih obradnih centara

CNC glodaći obradni centri su sofisticirani strojevi koji nude različite mogućnosti obrade materijala. Usporedba njihovih mogućnosti obično se temelji na faktorima poput veličine, snage, broja osi, brzine rezanja, točnosti i fleksibilnosti. Ključni aspekti koje treba uzeti u obzir prilikom usporedbe mogućnosti obradnih centara su nabrojani u nastavku. Veličina i kapacitet: CNC glodaći obradni centri dolaze u različitim veličinama i kapacitetima. Veći obradni centri obično

imaju veće radne stolove i mogu obraditi veće radne komade. Ovisno o potrebama aplikacije, odabir stroja ovisi o veličini i kapacitetu koji su potrebni za obradu. Broj osi: obradni centri mogu biti 3-osni, 4-osni ili čak višeosni. Broj osi određuje slobodu kretanja alata i omogućuje izvođenje složenijih operacija. Višeosni strojevi omogućuju rotacijska kretanja alata i mogu oblikovati složene konture i površine. Brzina rezanja se odnosi na brzinu kojom se alat giba prilikom rezanja ili slikovito „prolazi kroz materijal“. CNC obradni centri mogu imati različite brzine rezanja, koje ovise o snazi vretena i stabilnosti stroja. Veće brzine rezanja omogućuju bržu obradu materijala, ali treba paziti na kvalitetu obrade i točnost. Točnost je važan faktor u CNC glodačkim obradnim centrima, posebno kod izrade preciznih dijelova. Odnosi se na sposobnost stroja da ponovljivo postigne željene mjere i položaje obrade. Bolja točnost zahtijeva kruću konstrukciju, naprednije upravljačke jedinice i precizne sustave za kretanje. Fleksibilnost i automatizacija se odnose na sposobnost obradnog centra da se prilagodi različitim vrstama posla i izvede različite operacije. Napredni CNC obradni centri mogu imati mogućnosti automatske promjene alata, automatsku kalibraciju, sustave za nadzor i ostale značajke koje povećavaju fleksibilnost i učinkovitost rada. Softverska podrška igra važnu ulogu u CNC obradnim centrima. Kvalitetan softver omogućuje generiranje i optimizaciju G-kod programa, simulaciju operacija, vizualizaciju i programiranje. Napredni softver pruža korisne alate za modeliranje, dizajniranje i planiranje obrade. Pri usporedbi mogućnosti CNC glodaćih obradnih centara, važno je uzeti u obzir specifične potrebe aplikacije i pronaći stroj koji najbolje odgovara tim potrebama [1].

3.2.1. Pregled mogućnosti s obzirom na broj upravljanih osi gibanja alata i/ili obratka

CNC glodalice mogu se klasificirati prema broju upravljanih osi gibanja alata i/ili obratka. 3-osne CNC glodalice omogućuju kretanje alata u tri osi - X, Y i Z. To omogućuje izvođenje osnovnih operacija glodanja kao što su ravno glodanje, konturiranje, kružno glodanje itd. 3-osne glodalice pogodne su za širok raspon aplikacija i često se koriste u industriji [slika i ref]. 4-osne CNC glodalice omogućuju kretanje alata u četiri osi. Osim X, Y i Z osi, imaju i dodatnu rotacijsku os, obično označenu kao A-os. Ova rotacijska os omogućuje rotiranje obrtka u odnosu na alat, što omogućuje izvođenje operacija poput indeksiranog glodanja, graviranja pod kutom i slično. 5-osne CNC glodalice omogućuju kretanje alata u pet osi. Pored X, Y i Z osi, imaju dodatne rotacijske osi, obično označene kao A i B osi. Ove dodatne rotacijske osi omogućuju složenije operacije obrade, uključujući obradu kompleksnih geometrija i trodimenzionalnih površina. 5-osne glodalice često se koriste u industriji automobila, zrakoplovstva, medicinske industrije i sl. Višeosne CNC glodalice se odnose na glodalice s više od pet osi. To mogu biti 6-osne, 7-osne, 8-

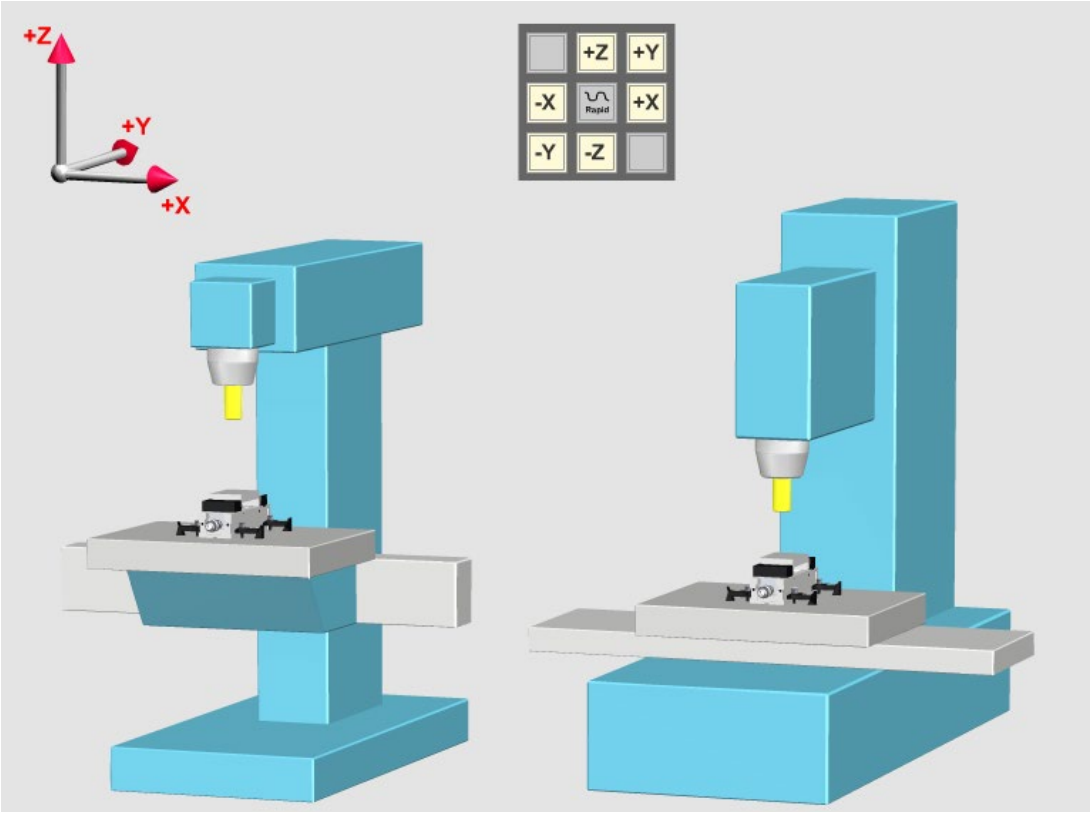
osne glodalice i više. Svaka dodatna os omogućuje dodatnu slobodu kretanja i mogućnost izvođenja složenih operacija obrade. Višeosne glodalice su vrlo napredne i koriste se u zahtjevnim aplikacijama gdje je potrebna visoka preciznost i kompleksna obrada. Važno je napomenuti da se broj osi odnosi na kretanje alata, dok se obradak može rotirati u skladu s tim kretanjem. Različite kombinacije osi mogu pružiti različite mogućnosti obrade i oblikovanja. Izbor CNC glodalice ovisi o specifičnim potrebama aplikacije i složenosti operacija koje će se izvoditi.

Također je potrebno razlikovati koliko osi je upravljivo simultano, što je definirano upravljačkom jedinicom. Na primjer kod 4-osnog glodanja, rotacijska os zauzima položaj, tek nakon zauzimanja određenog kuta kreće simultan rad preostale tri osi. Današnje upravljačke jedinice imaju mogućnost simultanog upravljanja sa desecima osi istodobno.

4. Dijelovi CNC stroja s obzirom na kinematiku i upravljanje

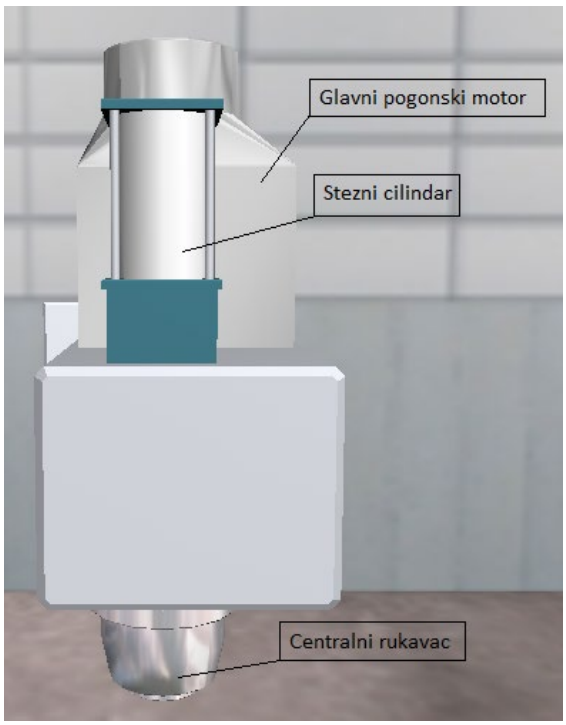
4.1. Mehanički dio CNC glodalice (konstrukcija, gibanje itd.)

Mehanički dio CNC glodalice obuhvaća konstrukciju stroja, gibanje osi i ostale komponente koje omogućuju pokretanje alata i obradu materijala. Radni stol je ravna površina na kojoj se postavlja radni komad. Može biti stacionarni vezan za kućište stroja ili pokretni vezan za najčešće Z-os, ovisno o dizajnu glodalice. Radni stol pruža potporu radnom komadu tijekom obrade i omogućuje precizno pozicioniranje i vođenje [1].



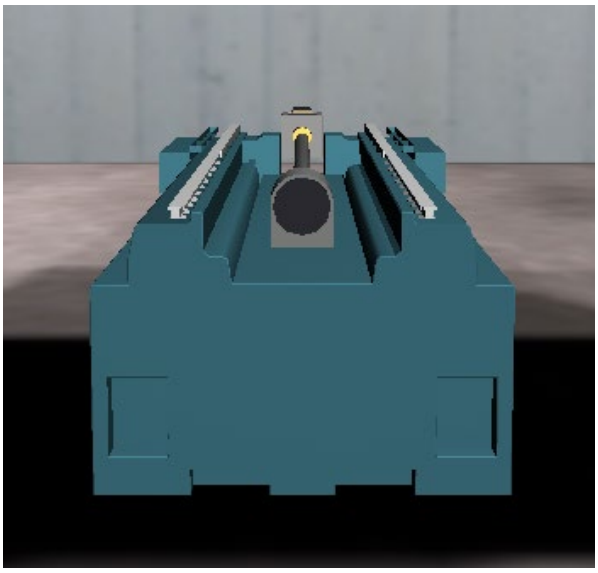
Slika 4.1 Prikaz konzole i križa radnog stola [3]

Vreteno je rotirajući element na glodalici koji drži alat za obradu. Vreteno se obično pokreće elektromotorom koji može generirati visoke brzine vrtnje. Vreteno ima odgovarajuće prihvate za montažu različitih alata za glodanje, kao što su reznice, svrdla ili glodalice.



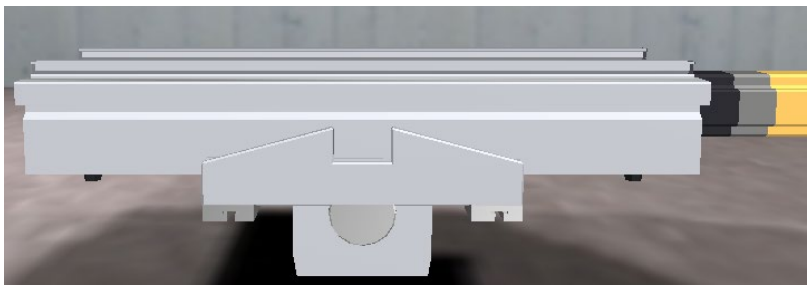
Slika 4.2 Vreteno glave CNC stroja [3]

Linearne vodilice omogućuju gibanje alata i stola u različitim osima. To su precizne pravokutne ili profilne šine na kojima klize nosači i pogoni. Linearne vodilice pružaju stabilnost, preciznost i glatko kretanje stroja. Ne dopuštaju pomak osi po poprečnim osima, dok osiguravaju gladak pomak po uzdužnoj osi. Preciznost i kvaliteta izrade linearne vodilice ne utječu direktno na uzdužni pomak.



Slika 4.3 Y os [3]

Slika 4.4 Z os [3]



Slika 4.5 X os [3]

Pogonski mehanizmi omogućuju gibanje alata i stola duž osi. Mogu uključivati razne prijenosnike, zupčanike, remene, lančanike ili druge mehanizme za prijenos snage. Pogonski mehanizmi pretvaraju rotacijski pokret motora u linearno kretanje alata i stola, navojno vreteno pretvara rotacijsko gibnje preko vezane navojne matice u linearno gibanje osi. CNC glodalice koriste različite vrste motora za pokretanje osi i vretena. To mogu biti servo-motori ili koračni-motori koji se kontroliraju pomoću upravljačke jedinice. Motori pružaju precizno i pouzdano pokretanje osi, omogućujući točno pozicioniranje i kretanje alata. Konstrukcija CNC glodalice može varirati od otvorenog okvira do zatvorenih kućišta s zaštitnim panelima. Konstrukcija stroja mora biti čvrsta, stabilna i dovoljno kruta kako bi podnijela sile obrade i održala preciznost.

Pravilno podešavanje mehaničkih komponenti i redovito održavanje ključni su za optimalno funkcioniranje CNC glodalice.

4.2. Upravljačka jedinica

Upravljačka jedinica je mozak cijelog CNC sustava. Kao to je već naglašeno upravlja osima te ostalim perifernim uređajima. Osnovna arhitektura su ulazne/izlazne jedinice, procesorska jedinica, memorija sustava, korisničko sučelje. Iako je jedan smjer razvoja upravljačkih jedinica da su sve naprednije sa stajališta upravljanja, drugi smjer je pojednostavljenje zbog ušteda. Te prebacivanje upravljačkog i logičkog djela na servo upravljačke. Jako puno operacija servo upravljači mogu sami izvršiti bez uopće korištenja upravljačke jedinice, u takvim slučajevima ona služi samo za interakciju ili povezivanje i zadavanje pozicija. Čak i interpolaciju gibanja danas mogu izvršiti servo upravljači. Klasična upravljačka jedinica za 3-osnu CNC glodalice pruža preciznu kontrolu i izvođenje operacija prema G-kod programu. Funkcionalnosti i specifičnosti upravljačke jedinice mogu varirati ovisno o proizvođaču i modelu CNC glodalice [1, 11, 15]. To i jedan od problema upravljačkih jedinica, ponekad ne kompatibilnost sa ostalim proizvođačima servo opreme. Gdje upravljačke jedinice zahtijevaju svu servo opremu istog proizvođača.

4.3. GUI

4.3.1. Opće karakteristike GUI-a

GUI (*eng. Graphical User Interface*) je skraćenica koja se odnosi na grafičko korisničko sučelje. To je način prikaza informacija i interakcije s računalnim programom ili sustavom putem grafičkih elemenata poput ikona, prozora, gumba, polja za unos teksta, izbornika i drugih vizualnih elemenata. GUI omogućuje korisniku da komunicira s računalom na intuitivan način. Umjesto da se koristi samo tekstualni unos naredbi, GUI korisnicima pruža vizualno sučelje koje olakšava navigaciju, unos podataka i izvršavanje radnji. Korisnici mogu jednostavno koristiti miš ili dodirni zaslon da bi interaktivno manipulirali elementima na zaslonu. Kroz GUI, korisnici mogu vidjeti prikaz informacija na zaslonu, kao i koristiti grafiku i simbole za predstavljanje funkcionalnosti i opcija programa. Osim toga, GUI pruža korisnicima povratne informacije o njihovim akcijama, poput promjene boje gumba nakon klika ili prikaza poruka o greškama. GUI je široko korišten u različitim aplikacijama, operativnim sustavima, programskim alatima i uređajima. On poboljšava korisničko iskustvo, olakšava korištenje programa, smanjuje potrebu za učenjem kompleksnih naredbi i omogućuje brzu interakciju s računalnim sustavom.

4.3.2. GUI upravljačke jedinice 3-osne CNC glodalice

GUI upravljačke jedinice 3-osne CNC glodalice pruža korisniku vizualno sučelje koje olakšava unos podataka, nadzor operacija i kontrolu stroja. Evo nekih karakteristika koje se često nalaze u GUI-ju upravljačke jedinice 3-osne CNC glodalice. Zaslon s informacijama: GUI prikazuje relevantne informacije o statusu stroja, trenutnoj poziciji alata, brzini gibanja, napredovanju programa i drugim relevantnim podacima. To može biti u obliku tekstualnih prikaza, brojčanih vrijednosti, grafikona ili drugih vizualnih indikatora.Gumbi i izbornici mogu biti organizirani u logične skupine kako bi olakšali navigaciju i pristup određenim funkcijama. GUI sadrži polja za unos teksta ili numeričkih vrijednosti. Korisnik može koristiti ta polja za unos specifičnih vrijednosti, poput brzine gibanja ili dimenzija obratka. Polja za unos mogu biti popraćena opisnim oznakama ili jedinicama mjere radi jasnoće.

Grafički prikaz predstavlja radni komad, alat ili putanju gibanja. To može biti u obliku 2D ili 3D prikaza koji pomaže korisniku vizualizirati oblik obradka ili provjeriti ispravnost putanje alata.

Može prikazati alarme ili upozorenja kada se pojave greške ili nepravilnosti tijekom rada stroja. To može uključivati poruke o preopterećenju, pogrešnoj putanji, nedostatku alata ili drugim problemima. Alarmi pružaju korisniku informacije o odstupanjima i pomažu u dijagnosticiranju problema. GUI može imati mogućnost simulacije operacija glodanja. To omogućuje korisniku da vizualno vidi kako će se izvesti odabrana operacija, provjeri putanju alata, detektira moguće kolizije ili ocijeni rezultat prije nego što se operacija stvarno izvrši.

Postavke i prilagodljivost korisničkog sučelja obično sadrži postavke i opcije koje korisnik može prilagoditi svojim potrebama. To može uključivati postavke brzine gibanja, jedinica mjere, jezik sučelja ili izgled prikaza. Prilagodljivost omogućuje korisniku da prilagodi sučelje prema svojim preferencijama i poboljša svoje radno iskustvo. Mogućnost izrade vlastitih načina interakcije, i programiranje raznih ciklusa te unos parametara ciklusa kao na primjer za umjeravanje alata.

Napomena da su opisane karakteristike tipične za GUI upravljačkih jedinica 3-osnih CNC glodalica, no specifičnosti mogu varirati ovisno o proizvođaču i modelu stroja.

4.4. Servo sustav (motor, driver)

4.4.1. Opće karakteristike

Servo sustav je ključni dio CNC glodalice i koristi se za precizno kontroliranje pokreta osi i postizanje točnog pozicioniranja alata. Ovdje su opće karakteristike servo sustava. Servo motor je

elektromotor koji se koristi za pokretanje osi. Servo motori imaju visoku preciznost, brzi odziv i visoku snagu. Oni pretvaraju električnu energiju u rotacijski mehanički pokret. Koriste se različite vrste servo motora, uključujući DC servo motore, AC servo motore i korak-servo motore. Servo motori uključuju povratnu petlju koja pruža informacije o stvarnoj poziciji i brzini osi. To se obično postiže korištenjem enkodera, rezolucija enkodera određuje preciznost servo sustava. Povratna petlja omogućuje servo upravljačkom sustavu da prati i održava željenu poziciju i brzinu pokreta. Servo upravljački sustav, poznat i kao servo upravljač, kontrolira rad servo motora. Upravljački sustav prima signale iz CNC upravljačke jedinice i generira odgovarajuće signale napajanja i kontrolne signale koji pokreću servo motor i kontroliraju njegovu brzinu, poziciju i moment. Servo sustav ima mogućnost podešavanja parametara kako bi se prilagodio specifičnim potrebama. To uključuje parametre kao što su brzina pokreta, ubrzanje, dekeleracija, osjetljivost povratne petlje, filtriranje signala i druge postavke koje utječu na performanse servo sustava. Zatvorena petlja se može odvijati unutar servo upravljača ili on može samo slijediti signale sa upravljačke jedinice. Komunikacija s upravljačkom jedinicom je ostvarena kako bi servo upravljač primao naredbe i odaslao povratne informacije. To se obično postiže preko komunikacijskog protokola poput Modbusa, EtherCAT-a ili drugih industrijski standardnih protokola. Servo sustavi obično uključuju sigurnosne značajke poput zaštite od preopterećenja, zaštite od kratkog spoja, zaštite od pregrijavanja i zaštite od gubitka koraka. Ove značajke osiguravaju siguran i pouzdan rad servo sustava i sprječavaju oštećenje sustava ili ozljede. Važno je napomenuti da se specifičnosti servo sustava mogu razlikovati ovisno o proizvođaču i modelu. Prilikom odabira servo sustava, važno je uzeti u obzir specifične zahtjeve CNC glodalice i prilagoditi servo sustav tim zahtjevima kako bi se postigla željena preciznost i performanse.

4.4.2. Karakteristike servo sustava kod 3-osne CNC glodalice

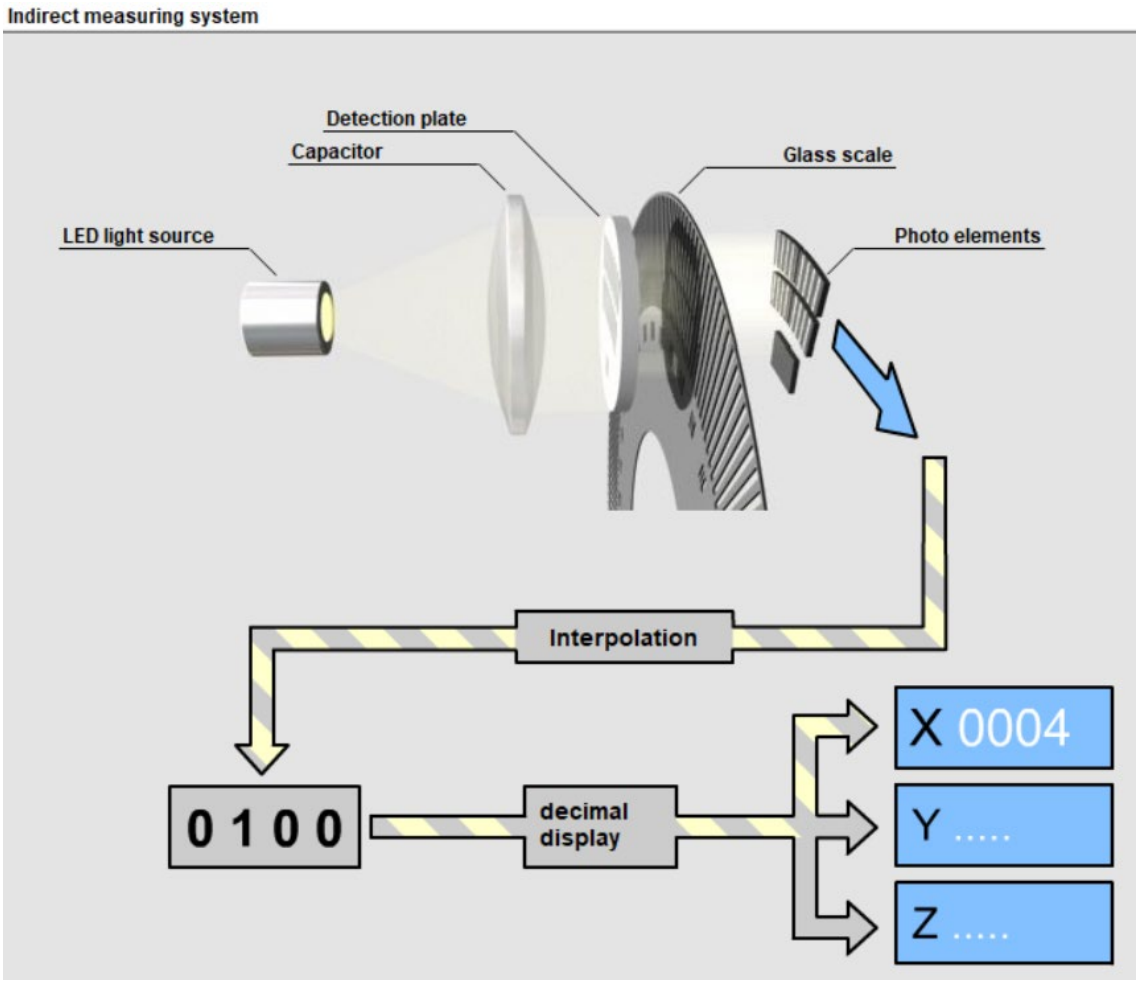
Karakteristike servo sustava kod 3-osne CNC glodalice mogu uključivati više stvari. Servo sustav u 3-osnoj CNC glodalici obično ima visoku preciznost pozicioniranja. To omogućuje točno pozicioniranje alata i radnog komada te postizanje željenih mjera i geometrije obrade. Servo sustav u 3-osnoj CNC glodalici karakterizira brzi odziv na naredbe upravljačke jedinice. To znači da servo motori brzo reagiraju na promjene brzine i smjera gibanja, što omogućuje visoku dinamiku gibanja i efikasno izvođenje operacija glodanja. Servo motori u 3-osnoj CNC glodalici imaju visoku snagu koja je potrebna za obradu različitih materijala. Visoka snaga servo sustava omogućuje obradu tvrdih materijala, kao i izvođenje zahtjevnih operacija glodanja s većim opterećenjem. Povratna petlja servo sustava u 3-osnoj CNC glodalici koristi visokorezolucijske enkodere ili slične senzore kako bi precizno pratila stvarnu poziciju osi. To osigurava točnost i

ponovljivost gibanja te minimizira pogreške pozicioniranja. Stabilnost i glatkoća gibanja: Servo sustav u 3-osnoj CNC glodalici pruža stabilno i glatko gibanje osi. To se postiže upravljanjem brzinom, ubrzanjem i deakceleracijom motora, kako bi se osiguralo ravnomjerno kretanje alata i postizanje visoke kvalitete obrade. Zaštita i sigurnost servo sustav-a u 3-osnoj CNC glodalici obično uključuje sigurnosne značajke poput zaštite od preopterećenja, zaštite od pregrijavanja, zaštite od kratkog spoja i drugih. Ove značajke osiguravaju siguran rad stroja i sprječavaju moguće oštećenje sustava ili ozljede. Važno je napomenuti da su specifičnosti servo sustava u 3-osnoj CNC glodalici ovisne o proizvođaču i modelu stroja. Prilikom odabira stroja, važno je uzeti u obzir karakteristike servo sustava kako bi se osigurala optimalna preciznost, performanse i pouzdanost CNC glodalice.

4.5. Povratna veza

Povratna veza ima ključnu ulogu u CNC strojevima, uključujući i CNC glodalice. Ona omogućuje sustavu da prati i kontrolira stvarno stanje stroja, osigurava preciznost gibanja, kompenzira pogreške i osigurava točno pozicioniranje. Praćenje pozicija omogućuje CNC stroju da prati stvarnu poziciju alata i obratka u odnosu na željenu poziciju definiranu u G-kod programom. To se postiže korištenjem senzora poput enkodera koji bilježe stvarno gibanje osi. Na temelju tih informacija, sustav može prilagoditi pokrete kako bi postigao točno pozicioniranje. Usporedba stvarne i referentne vrijednosti omogućuje sustavu da detektira i korigira pogreške koje mogu nastati tijekom gibanja. Na primjer, ako dođe do odstupanja u pozicioniranju, sustav može primijeniti korektivne signale kako bi ispravio pogrešku i vratio se na željenu poziciju. To osigurava visoku preciznost obrade i ponovljivost. Uz praćenje pozicije sustav vrši praćenje brzine pojedine osi. Brzina se utvrđuje iz uređaja enkodera, kao i za praćenje pozicije ili posebnog uređaja taho-generatora koji služi samo za praćenje brzine. To je važno za postizanje željene brzine i ubrzanja, kao i održavanje stabilnosti gibanja. Ako dođe do odstupanja u brzini, sustav može prilagoditi napajanje motora kako bi se postigla željena brzina i održala ravnoteža. Povratna veza igra ključnu ulogu u sigurnosti CNC strojeva. Ona omogućuje detekciju i zaštitu od neželjenih situacija poput kolizija ili preopterećenja. Ako senzor bilježi nepoželjne promjene, sustav može automatski zaustaviti ili prilagoditi rad stroja kako bi se spriječila šteta na stroju ili ozljede. Povratna veza omogućuje kalibraciju i kompenzaciju parametara stroja. Na temelju povratnih informacija, sustav može prilagoditi parametre upravljanja motora, pomaka vretena ili kompenzacije geometrijskih pogrešaka kako bi se osigurala točnost i preciznost obrade. Ukratko, povratna veza ima ključnu ulogu u praćenju i kontroliranju stvarnog stanja CNC stroja, osiguravajući točnost i preciznost gibanja. To omogućuje CNC stroju da postigne željeno

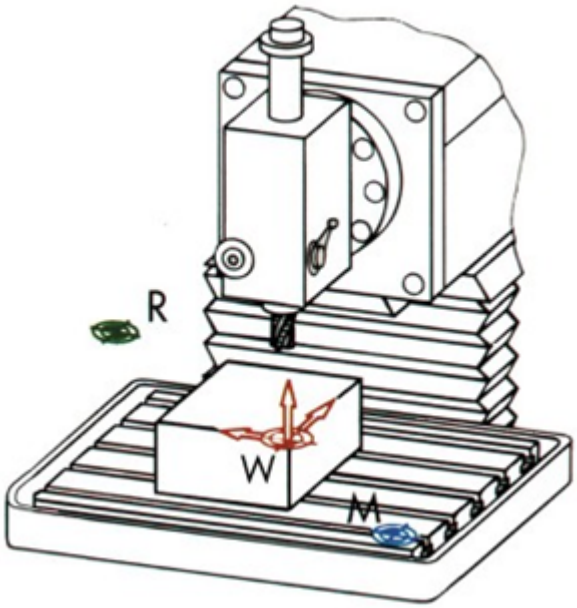
pozicioniranje, brzinu i kvalitetu obrade, dok istovremeno pruža sigurnost i zaštitu od neželjenih situacija.



Slika 4.6 Povratna mjerna veza sa enkoderima [3]

4.6. Nul-točke

Nul-točka, također poznata kao referentna točka ili početna točka, ima veliki značaj kod 3-osne CNC glodalice. U radnom prostoru stroja postoji nekoliko važnih referentnih točaka koje su definirane u upravljačkom sustavu



Slika 4.7 Nul točke stroja [2]

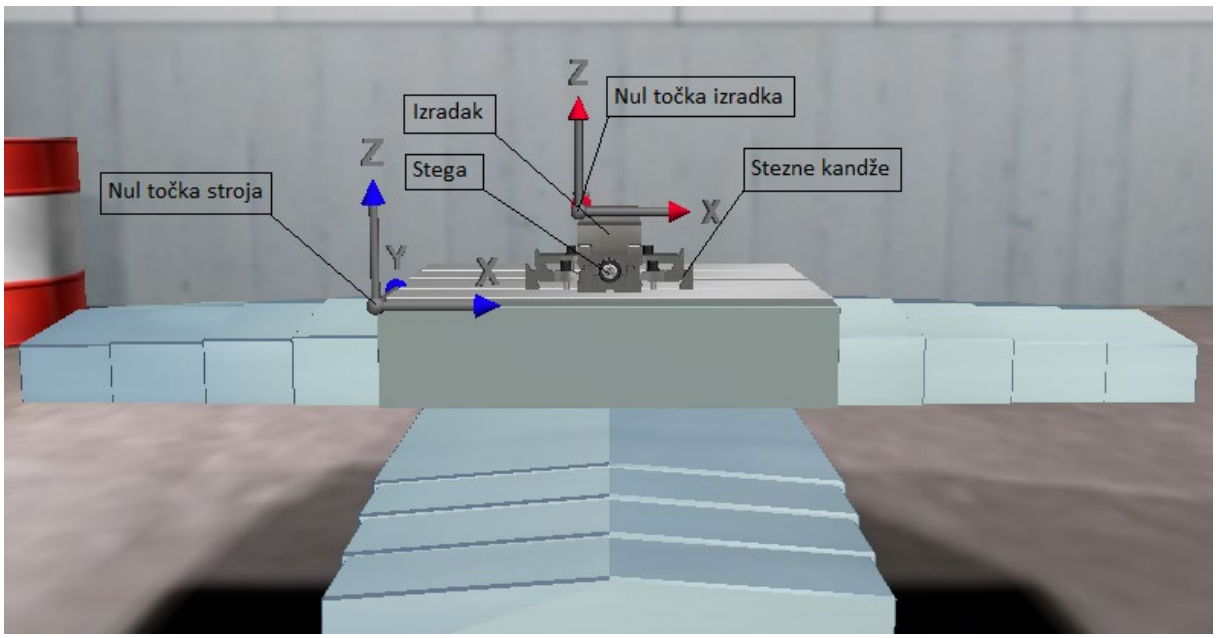
Tablica 4-1Popis nul točki stroja

	*Nul točka stroja M* Nulta točka stroja je definirana od strane proizvođača i ne može biti promijenjena.
	*Nul točka obratka W* Nulta točka obratka (W), koja se često naziva i programska nulta točka, od koje je definiran programski kod. Može biti postavljena u bilo koju poziciju u radnom prostoru uz preporuku da to bude točka za koju je vezan najveći broj dimenzija na crtežu obratka.
	*Referentna točka R* Referentna točka (R) se koristi za podešavanje nulte točke mjernog sustava, jer u većini slučajeva nulta točka stroja nije dostupna. Od ove točke mjerni sustav počinje računanje pozicije pri pozicioniranju alata i obratka.
	*Nul točka alata N* Nul točka alata (N) definira poziciju pomaka alata. Početna pozicija nul točke alata nalazi se unutar prihvata radnog vretena

Ovo je pozicija na kojoj se određuju početne vrijednosti za koordinatni sustav stroja. Nul-točka služi kao referentna točka za određivanje apsolutnih pozicija osi stroja. Nakon što se stroj referencira na nul točku, on zna da su sve koordinate izražene u odnosu na tu točku. To omogućuje precizno pozicioniranje alata i obratka na željene pozicije tijekom obrade. Referentna Nul-točka može služiti i kao sigurnosna točka. Nakon što se stroj referencira na nul-točku, operater može biti siguran da je stroj u poznatom i sigurnom stanju. To može uključivati zaustavljanje svih pokreta i osiguranje da nema prepreka na putanji gibanja. Nul-točka se koristi i za kalibraciju i kompenzaciju parametara stroja. Na temelju nul-točke, moguće je prilagoditi parametre poput geometrije stroja, pomaka osi ili koraka motora kako bi se osigurala točnost i preciznost obrade. Postavljanje nul-točke omogućuje reproduktivnost obrade, ponovljivost izrade u o odnosu na prošlo pokretanje stroja i referenciranje. Kada se stroj referencira na nul-točku, operater može precizno reproducirati pozicije alata i obratka za ponovno izvođenje istih operacija. To je posebno važno kada je potrebno izvesti više operacija na istom radnom komadu ili kada je potrebno ponoviti postupak obrade na drugim radnim komadima. Nul-točka omogućuje precizno podešavanje radnog komada. Nakon referenciranja na nul-točku, operater može pravilno postaviti radni komad u odnosu na alat i osigurati ispravnu poziciju za obradu. To omogućuje postizanje željene geometrije, dimenzija i kvalitete obrade. Ukratko, nul-točka je važna za precizno pozicioniranje, sigurnost, kalibraciju i kompenzaciju kod 3-osne CNC glodalice. Ona osigurava referentnu točku za koordinatni sustav stroja i omogućuje točno i reproduktivno pozicioniranje alata i obratka [3].

Točka, do koje se dolazi pomakom, u odnosu na referentnu nul točku, je nul točka obratka koja se postavlja na radni komad. Sve pozicije koje su definirane unutar program su definirane u odnosu na tu točku. Upravljačka jedinica dopušta postavljanje nekoliko ovakvih nul točaka na više radnih komada koji se nalaze na radnom stolu. Nije potrebna posebno definirati obradu za svaki radni komad. Obrada se vrši samo pozivanjem slijedeće radne točke sa istom programom.

Pri gibanju stroja ili izvođenju programa, koordinate pomaka stroja su u biti pomaci nul točke alata u odnosu na nul točku obratka. Vrijednosti koordinata u programu su pomaci vrha alata u odnosu na radni komad.



Slika 4.8 Stezni stol CNC stroja [3]

5. Interpreter

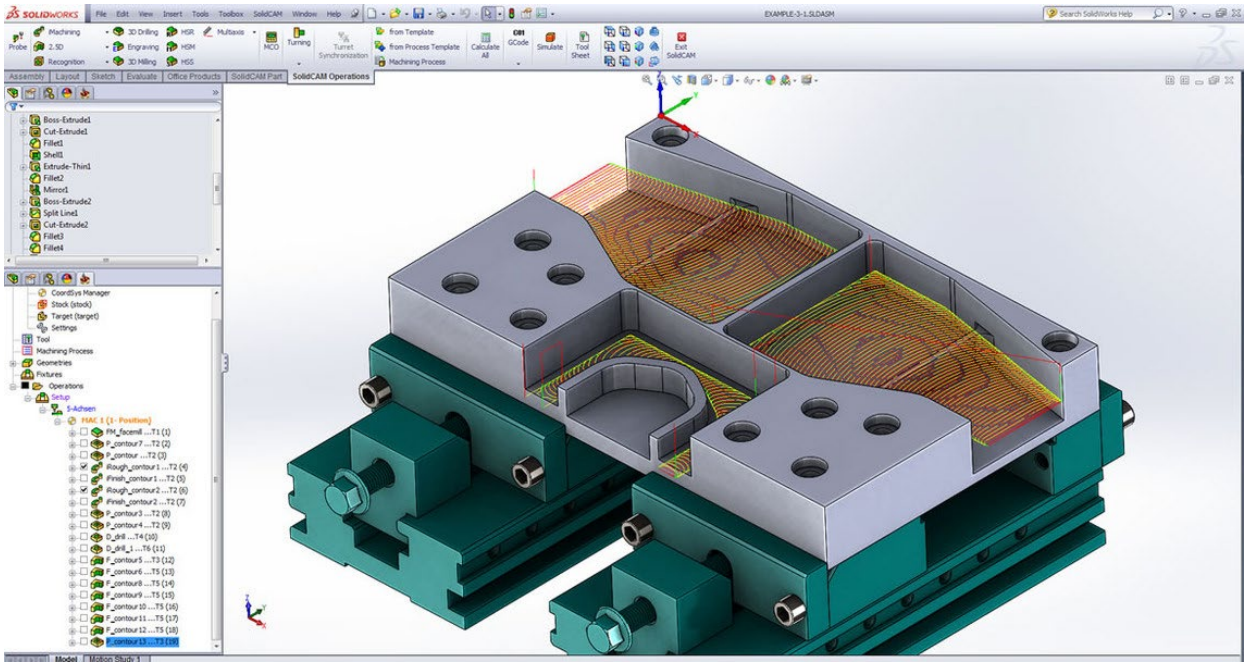
5.1. G-kod

G-kod je skraćenica za geometrijski kod *(*engl. *G-code*), koji je standardni jezik programiranja koji se koristi za upravljanje radnji i operacija na CNC (engl. *Computer Numerical Control*) strojevima. CNC strojevi se koriste u industriji za automatizirano izvođenje različitih zadanih operacija obrade materijala, kao što su rezanje, bušenje, glodanje ili graviranje. G-kod je niz naredbi koje definiraju različite „upute“ koje CNC stroj treba izvesti.

Načini unosa G-koda se mogu razlikovati. Postoji način ručnog upisivanja na samoj upravljačkoj jedinici (tzv. programiranje na mjestu, engl. *online*). Metoda se sastoji od ručnog upisivanja naredbi G-koda, i dovoljno je dobra za jednostavnija vođenja stroja. Zahtijeva od operatera određeno iskustvo zbog same apstrakcije programiranja, gdje si operater mora sam vizualizirati gibanje stroja prema naredbama koje unosi. Jedan veliki nedostatak ovog načina programiranja, kod nekih upravljačkih jedinica, jest nemogućnost simultanog rada stroja i programiranja. Sljedeći način unosa programa je prijenos programa na upravljačku jedinicu, a program je, najčešće, generiran nekim programskim alatom.

Programski alati služe kao podrška proizvodnji, a nazvani su CAM (engl. *Computer Aided Manufacturing*) programi. Ovaj način programiranja se svodi na grafičko programiranje gdje su obradak, alati i ostali dijelovi stroja, koji sudjeluju u obradi, prikazani trodimenzionalnim modelima. Programiranje se svodi na generiranje putanji alata obratka iz trodimenzionalnog CAD modela.

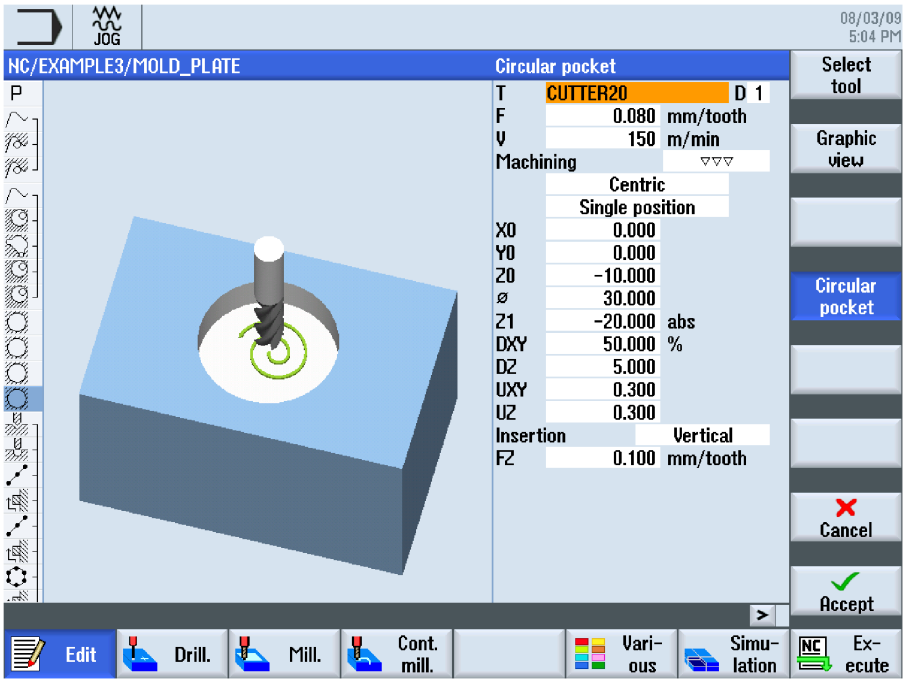
Bitan segment u lancu generiranja programa, te samog pokretanja programa na upravljačkoj jedinici, jest postprocesor koji „prevodi“ putanje alata u G-kod instrukcije za specifičnu upravljačku jedinicu, i, konfiguraciju stroja. Najčešće je to označeno kao CAD/CAM, odnosno, integrirani sustav modeliranja željenog konačnog obratka, i sirovca, te pripreme proizvodnje (slika u nastavku).



Slika 5.1. CAM programski alat SolidCAM (izvor: SolidCAM)

Ovo programiranje se koristi za izradu najkompleksnijih izradaka te najčešće nudi korisniku simulaciju programa koji će biti generiran. Pruža, uz generiranje putanje alata, fleksibilnost u definiranju alata, parametara kretanja alata, kompenzacije alata, izmjene alata, pomoćnih funkcija i upravljanje ostalim funkcionalnostima stroja i popratne opreme. Generirani program se prenosi sa računala na upravljačku jedinicu najčešće putem prijenosne jedinice (USB), ili putem računalne mreže (Ethernet). Detaljnije je prikazano u [9]. Velika prednost ovog načina programiranja je mogućnost distanciranog programiranja (engl. *offline*) na računalu, dok, u isto vrijeme stroj može izvoditi proizvodni proces.

Programiranje koje ne koristi specifičnosti oba prethodno navedena načina je programiranje putem ciklusa. Operater se ne bavi direktno G-kodom, već unosi specifične informacije o obratku, materijalu, korištenim alatima i putanji alata. Tako stvoren program nije uobičajeni G-kod već se program sastoji od niza ciklusa gdje svaki ciklus tvori jedan manji dio obrade, kao naprimjer bušenje rupa, glodanje utora itd. Ovaj način programiranja određen je od strane proizvođača upravljačkih jedinica te u određenoj mjeri podržava grafički prikaz putanja alata, grafički prikaz unošenja parametara pojedinog ciklusa itd.



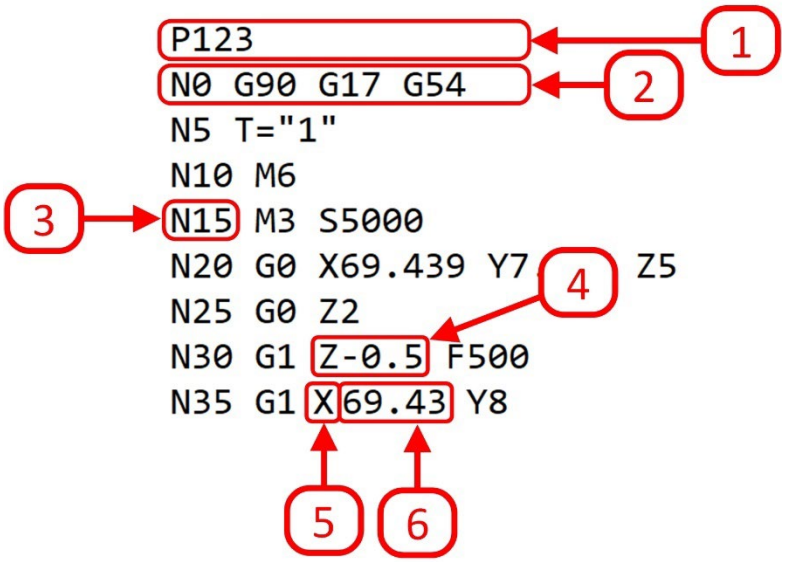
Slika 5.2 Programiranje ciklusa Siemens ShopMill

5.1.1. Dijelovi G-koda

Kako bi bolje shvatili način rada interpretera, potrebno je upoznati se najprije sa osnovnim dijelovima G-kod programa, što pridonosi razumijevanju načina rada interpretera. Odnosno, potrebno je pojasniti što različite kombinacije slova i brojeva znače za daljnje procesuiranje intrepretiranog programa. U ovom potpoglavlju biti će prikazane najčešće korištene instrukcije u programiranju CNC stroja. Također, daljnja detaljna razrada, zbog količine pozadinske logike koja slijedi nakon interpretacije, uključivati će samo instrukcije koje su vezane za gibanje osi stroja. G-kod je sastavljen od instrukcija koje upravljačka jedinica čita, prevodi i procesuira u najčešća gibanja CNC stroja. Programi različitih proizvođača mogu se razlikovati u sintaksi, značenju pojedinih instrukcija, naredbi za programiranje CNC stroja, tj. koje radnje će izvršiti upravljačka jedinica, ovisno o različitim kombinacijama i rasporedu slova i brojeva.

Sintaksa programa za upravljanje CNC strojeva (ili G-kod) je standardiziran standardima ISO 6983 i ANSI/EIA RS-274. Time se definira struktura programa i način djelovanja na CNC stroj i njegovu perifernu opremu.

a)



b)

```
%_N_PRIMJERPROGRAMA_MPF
;$PATH=/_N_WKS_DIR/_N_SHOPMILL_WPD
```

Slika 5.3 Struktura programa – G-kod naredbe

Na prethodnoj slici je prikazana podjela G-koda, tj. značenje sljedećih oznaka koje označavaju:

1. Broj programa - koristi se kao identifikacija programa, tj. kao identifikacija programa unutar upravljačke jedinice. Označava se znakom „P“ i numeričkom vrijednosti broja programa. Na slici 5.3 b) prikazan je jedan primjer načina obilježavanja programa. Broj programa može se označavati i znakom „O“ ili „#“. Također, označavanje može biti i znakom „%“, koji označava naziv programa. Blok ispod opisuje i mjesto spremanja programa unutar memorije upravljačke jedinice.
2. Programski blok - svaki program se sastoji od linija ili blokova. Najčešće započinje znakom „N“ koji definira broj linije. Izvršenje programa se izvodi odozgo prema dolje, logičkim redom.
3. Programski blok se sastoji od niza adresa, koje su odvojene, najčešće, razmakom. Oznaka „N“ označava broj programa sa korakom povećanja od 5 ili 10. Adresa u primjeru nema utjecaj na radnje stroja.
4. Minimalna jedinica unutar jednog bloka je riječ koja opisuje vrstu instrukcije, u ovom primjeru - gibanje stroja. Riječ se sastoji od adrese i numeričke vrijednosti.
5. Adresa je predstavljena znakom ili kombinacijom znakova abecede, od A do Z. Opisuje parametar instrukcije kojim se želi upravljati. Definira vrstu instrukcije koju se želi izvesti na stroju. Kao znakovi, koriste se 26 znakova engleske abecede. Kao posebne

vrste znakova mogu se koristiti simboli, simboli koji su vezani za numeričku vrijednost (+, -, .), te, ostali specijalni simboli koji su pokazani 1. točkom definiranja naziva programa i mjesta spremanja unutra memorije upravljačke jedinice.

6. Drugi dio, od kojeg se sastoji adresa, naziva se numerička vrijednost. Ona brojčano zadaje vrijednost parametra adrese. Općenito, numeričke vrijednosti mogu biti zapisane pozitivnim ili negativnim brojem, te, kao cjelobrojna ili decimalna vrijednosti. To ovisi o upravljačkoj jedinici i vrsti instrukcije na koju se odnosi.

5.1.2. Pripremne funkcije

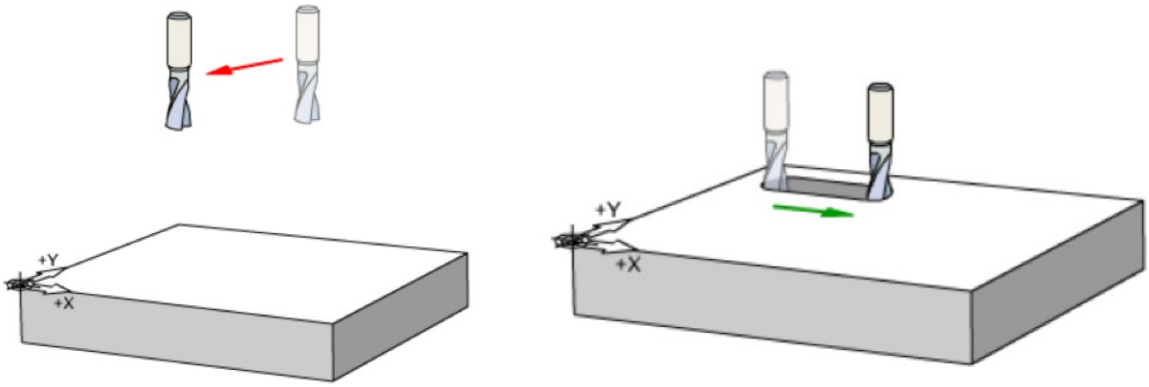
Pripremne funkcije izvršavaju pripremne naredbe za izvršenje gibanja CNC stroja, koje su predstavljene znakom „G“. Adresa pripremne funkcije ima svrhu pripremiti upravljačku jedinicu, najčešće na određeno gibanje, ili, kod nekih upravljačkih jedinica na pozivanje određenog ciklusa. Bitno je naglasiti da se u oba slučaja ne koriste iste numeričke vrijednosti adrese G. Numerička vrijednost može podijeliti pripremne funkcije na modalne čija pozivanja, u jednom bloku, imaju utjecaj na više blokova tijekom izvršavanja. S druge strane, one ne-modalne vrijede samo za blok u kojem se pozivaju. Primjeri podjele biti će dani kod daljnje podjele i objašnjenja pripremne funkcije. Ako se u jednom bloku koristi više pripremnih funkcija (gdje jedna isključuje drugu, više vrsta gibanja…) upravljačka jedinica koristi onu vrstu gibanja koja je zadnja definirana. Pripremne funkcije podijeljene su na različite grupe koje su prikazane daljnjim tekstom. Navedene grupe nisu kronološki prikazane kako bi se trebale pojavljivati u programu G-koda, nego, prema učestalosti korištenja i važnosti u radu CNC.

Naredbe gibanja (engl. Motion commands)

Naredbe gibanja (engl. *Motion commands*) su naredbe koje određuju način gibanja alata ili stola CNC stroja, ovisno o izvedbi stroja i osi. Gibanje se odnosi na gibanje alata u odnosu na radni komad, ili se giba radni komad, u odnosu na alat.

G00 označava brzo gibanje alata koje ne zahtjeva parametar brzine gibanja, već je brzina zadana brzinom koja se postavlja prilikom puštanja u rad. Brzina gibanja je najčešće blizu maksimalne brzine gibanja koja je određena konstrukcijskom izvedbom i korištenim aktuatorima stroja. Brzo gibanje ili brzi hod alata je ne-modalna naredba koja se mora koristiti u svakom bloku programa gdje se želi ostvariti takvo gibanje, koje služi za pozicioniranje alata iznad mjesta početka obrade, odnosno, radnog komada. Ne koristi se za kretanja kada je alat u radnom zahvatu i izvršenju obrade. Putanja alata kod ovog gibanja nije od velike važnosti, tako da najčešće, ovisno

o upravljačkoj jedinici, gibanje nije interpolirano. Interpolirano gibanje predstavlja način upravljanja pojedinim osima kako bi se ostvarilo određeno kretanje između dvije točke. Interpolirano gibnje u ovom slučaju bi značilo - ravna linija između dvije točke; brzina gibanja svake osi je takva da sve osi stroja stignu u odredišnu poziciju, u istom trenutku. Dok, s druge strane, kod ne-interpoliranog gibanja, sve osi ne dolaze u odredišnu točku u isto vrijeme; brzina svih osi je jednaka, pa tako, ona os koja mora prevaliti manju udaljenost prije zauzme poziciju, tako da se putanja alata sastoji iz više spojenih ravnih linija. Takav način ne-interpoliranog gibanja ne zahtjeva veliku procesorsku moć upravljačke jedinice. Odredišna točka gdje se želi pozicionirati alat opisana je koordinatama osi. Iza znaka G00 definirane su koordinate sa numeričkim vrijednostima. G01 X___ Y___ Z___ za pozicioniranje alata u stroju koji posjeduje tri translacijske osi.

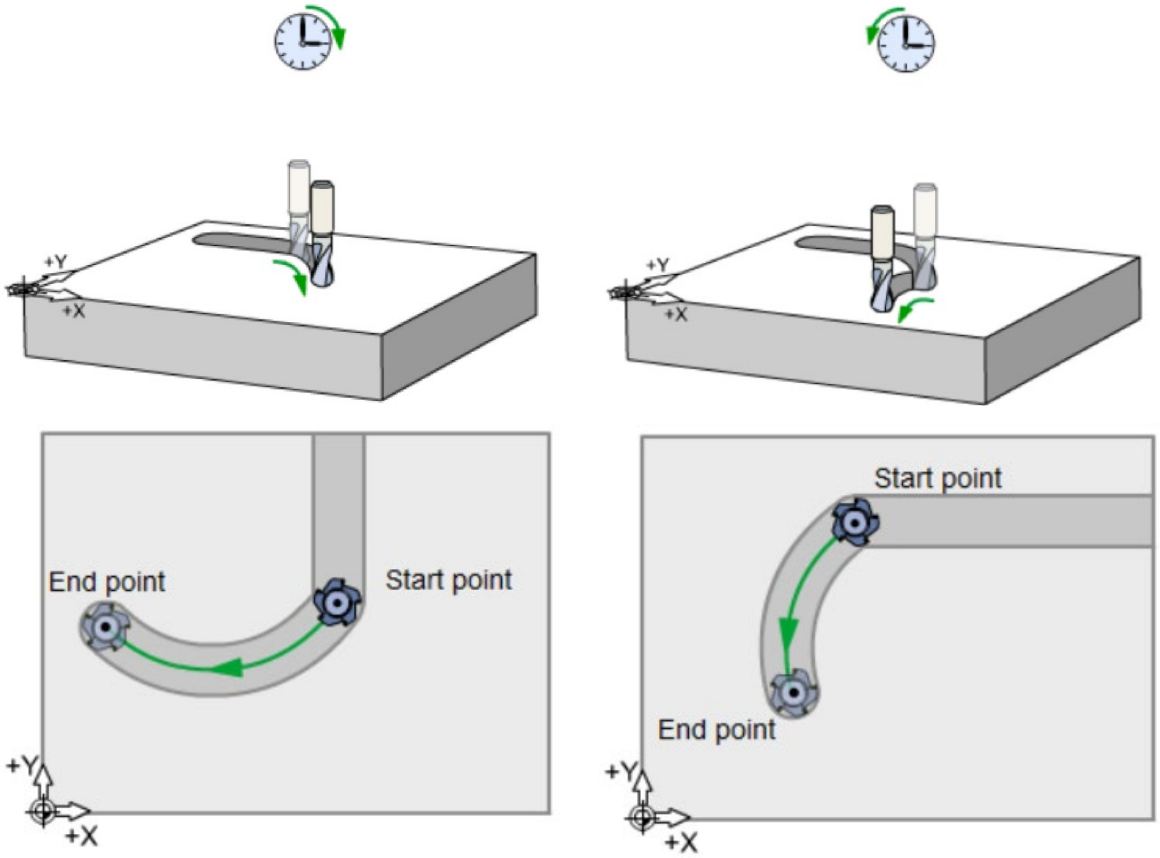


Slika 5.4 G00 naredba (lijevo), G01 naredba (desno)[3]

G01 označava linearno interpolirano gibanje sa zadanom brzinom gibanja; putnja alata u zadanu željenu točku je ravna linija, te je naredba modalna. To znači da pozivanje naredbe u jednom bloku programa vrijedi za G-kod daljnjeg izvođenja programa dok se ne pojavi neka druga vrsta naredbe. Linearno gibanje se koristi za obradu, gdje je alat u zahvatu odnošenja materijala sa predmeta obrade. U ovom slučaju, interpolacija pomaka, odnosno, upravljanje pojedinim osima koje svojim pomacima doprinose ukupnom gibanju je jako bitno kako bi se održala kontinuirana brzina i smjer kretanja alata. To utječe na kvalitetu obrađene površine. Potrebno je definirati brzinu gibanja. Odredišna pozicija se definira isto kao i kod G00.

G02 i **G03** označavaju kružno interpolirano kretanje. G02 izvodi gibanje u smjeru kazaljke na satu, a G03 izvodi gibanje suprotno od kazaljke na satu [9]. Kružna gibanja također se koriste za odnošenje materijala sa predmeta obrade, tako da je potrebna preciznost interpolacije, odnosno, precizno vođenje osi kako bi se ostvarila kružna (lučna) putanja alata. Osim definiranja smjera gibanja potrebno je definirati odredišnu točku, te poziciju središta radijusa željene putanje. Dio bloka za kružna gibanja izgleda ovako: G01 X___ Y___ Z ___I ___J ___ K___. Kao i kod

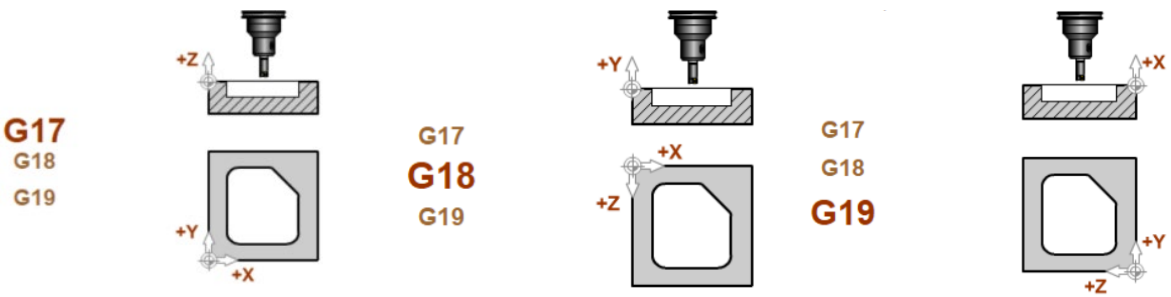
prijašnjih gibanja X, Y i Z os definiraju odredišnu točku dok I, J i K definiraju inkrementalnu udaljenost središta radijusa od početne točke u smjeru osi X (I), Y(J) i Z(K). Ova vrsta gibanja ne mogu se simultano gibati u sve tri osi, nego je gibanje po kružnoj putanji ograničeno samo u jednoj ravnini, koje mogu biti x-y, x-z, y-z. Primjer gibanja u ravnini x-y prikazan je slijedećom slikom.



Slika 5.5 G02 i G03 naredbe [3]

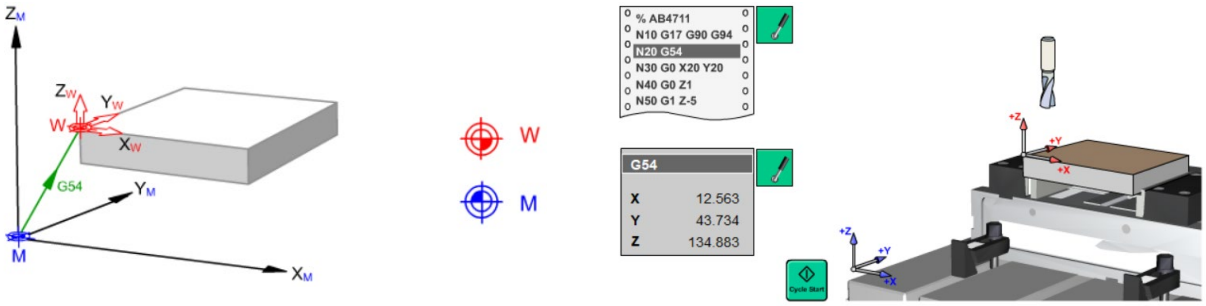
Naredbe aktivne ravnine

Kako bi se kružne interpolacije mogle provesti potrebno je definirati u kojoj ravnini će kružnica biti opisana. **G17** – x-y ravnina, **G18** – x-z ravnina, **G19** – y-z ravnina.

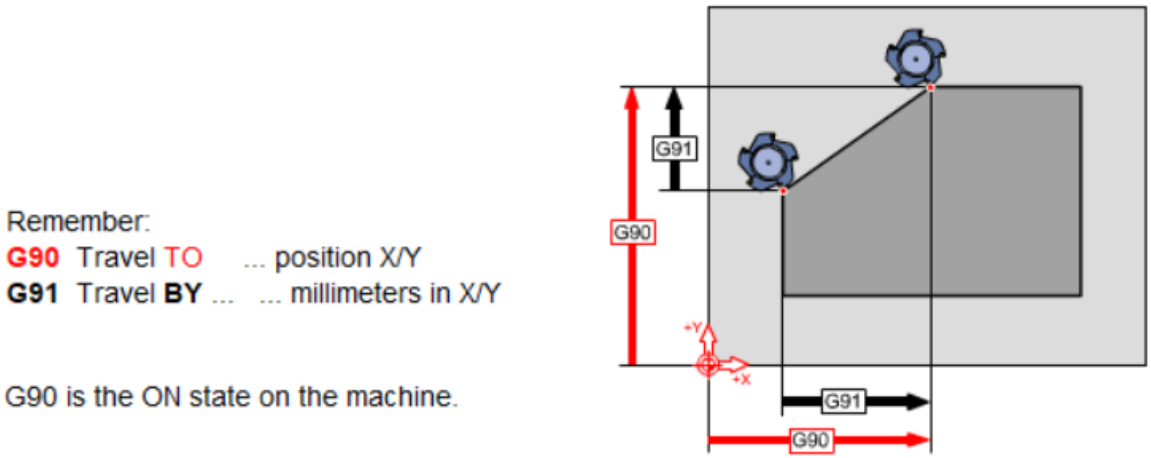


Slika 5.6 Odabir ravnina [3]

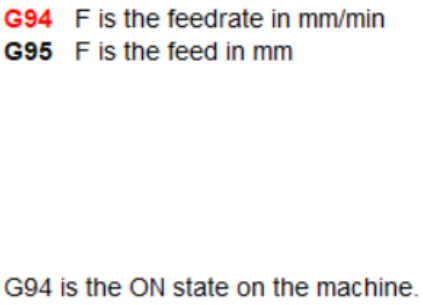
Odabir aktivne ravnine najviše utječe na kružna gibanja. Postizanje kružnih gibanja (kao na slici 5.5) ostvarivo je korištenjem G17 ravnine, kružna gibanja su u ravnini x-y. Ovo je najčešće korištena ravnina, te se kod mnogih upravljačkih jedinica ne mora posebno definirati jer je unaprijed zadana. Kružna gibanja kod promjene aktivne ravnine G18 ili G19 izvodile bi putanju gibanja radijusa vertikalno u odnosu na predmet obrade (slika 5.5), a ne horizontalno kao kod G17.



Slika 5.7 Radni pomaci CNC stroja [3]



Slika 5.8 G90 i G91 naredbe [3]



Slika 5.9 G94 i G95 naredbe [3]

Naredbe pozicioniranja

G90 i **G91** naredbe pozicioniranja (engl. *Positioning commands*) određuju pozicioniranje alata u odredišnu točku. G90 se koristi za apsolutno pozicioniranje, gdje su definirane koordinate udaljenosti od stacionarnog aktivnog radnog koordinatnog sustava. Naredba G91 se koristi za inkrementalno pozicioniranje, gdje su definirane koordinate udaljenosti od trenutne pozicije. Odredišna točka ovisi o trenutnoj poziciji alata.

G54-G59 definiraju naredbe za postavljanje aktivnog koordinatnog sustava radnog komada; sa G54 se označava prvi koordinatni sustav radnog komada, do najčešće, G59. G54 je ishodište koordinatnog sustav, pozicija i orijentacija u odnosu na globalni koordinatni sustav, ili, koordinatni sustav stroja. Koordinatni sustav radnog komada određuje sam operater, te ga postavlja negdje na radni komad kako bi olakšao unošenje ili izmjenu programa G-koda. Nakon pozivanja aktivnog koordinatnog sustava, sve točke unutar programa su definirane u odnosu na aktivni lokalni koordinatni sustav. Dodani pomak, premještanje koordinatnog sustava radnog komada može se uključiti pozivanjem lokalnog koordinatnog sustava koji je vezan za aktivni koordinatni sustav radnog komada.

G64 i **G61** naredbe izvođenja programa, tj. brzine u kutovima radnog komada. Ovim naredbama se specificira način gibanja, brzina gibanja prilikom dolaska radnog alata u kut, i, promjena smjera gibanja. Obje naredbe imaju veliki utjecaj na kasniji proračun interpolacija i generiranje putanje. Oznakom G64 alat se ne zaustavlja u točki promjene smjera rada, nego, nastavlja gibanje. Kako bi alat nastavio gibanje tijekom promjene smjera, a zadržao brzinu, promjena smjera se izvodi malim radijusom što uzrokuje i promjenu u putanji. Rezultat je brza obrada, posebice kod programa sa puno promjena smjera, ali, uz određenu nepreciznost koja je ponekad i zanemariva. Naredbom G61 radni alat se zaustavlja u svakoj točki promjene smjera gibanja, te, nastavlja gibanje sa ubrzanjem iz brzine jednake nuli. Ovaj način gibanja povećava preciznost, ali se vrijeme izvođenja programa povećava.

5.1.3. Pomoćne funkcije

Pomoćne funkcije služe za upravljanje dijelova stroja, tj. najčešće za jednostavno upravljanje perifernim uređajima CNC stroja. Označavaju se sa „M“.

M8 naredbe za hlađenje i podmazivanje: Ove naredbe kontroliraju sustave hlađenja i podmazivanja CNC stroja. To može uključivati uključivanje i isključivanje rashladnih tekućina, postavljanje brzine protoka rashladne tekućine, podmazivanje alata i slično [9].

M3 i **M4** upravljanje smjerom vrtnje glavnog vretena: M3 ili M4, naredbe koje definiraju vrtnju glavnog vretena u smjeru gibanja kazaljke na satu - CW, ili, smjer vrtnje suprotan od smjera

gibanja kazaljke na satu - CCW. Dodatno, može se izvršiti naredbe ograničenja brzine, promjene prijenosnog omjera glavnog vretena itd.

M5 - naredba zaustavljanja vretena. Koristi se u kombinaciji sa definiranjem brzine vrtnje koji će biti spomenut kasnije.

M6 naredba poziva promjenu alata koji mora biti naveden uz ovu naredbu. Naredba za pozivanje promjene alata je: M6 T2. Uz pozivanje alata može se pozvati i kompenzacija alata. Potrošenost alat uzrokuje netočnu mjeru izrađenog komada, te potrebo je izvršiti pomicanje putanje alata kako bi se ostvarile točne dimenzije [9].

5.1.4. Ostale funkcije

T - naredbe alata: Ove naredbe se koriste za upravljanje alatima koji se koriste u CNC operacijama. To može uključivati naredbe za odabir alata, postavljanje brzine rezanja, promjenu alata tijekom operacije, mjerenje alata i slično [9].

F - naredbe brzine: Ove naredbe određuju brzinu kretanja alata ili stola, definiranju relativnu brzinu između alata i obratka. Primjerice, F naredba se koristi za postavljanje brzine rezanja gdje brzina rezanja ovisi o mjernoj jedinici kojom je izražena, tj. da li se koristi metrički ili inčni mjerni sustava. Najčešće korištena mjerna jedinica, odnosno numerička vrijednost koja se nalazi uz adresu F, je mm/min.

S - naredba brzine vrtnje radnog vretena: Oznaka „ S“ izražava se u okretajima u minuti.

5.2. Prevođenje i priprema G-koda

Generiranje G-koda: Prvi korak je generiranje G-koda koji odgovara željenim operacijama obrade. To se obično postiže korištenjem CAM softvera. CAM softver prima ulazne nacrte ili modele, i, pretvara ih u niz G-kod naredbi. U ovom koraku se također mogu postaviti parametri obrade, kao što su brzina rezanja, dubina rezanja, alati koji se koriste itd.

Simulacija i verifikacija nakon i tokom generiranja G-koda: Preporučljivo je provesti simulaciju i verifikaciju kako bismo provjerili ispravnost i sigurnost programa. Simulacija omogućuje virtualno izvođenje CNC operacija kako bi se provjerilo da li se alat kreće onako kako je planirano, te, da li postoje mogući sudari, ili druge pogreške. Verifikacija uključuje provjeru postupka obrade, sigurnosti i korektnosti izvedbe. Simulacija i verifikacija je najčešće integrirana unutar CAM programskog alata. Osnovna simulacija se provodi tokom programiranja koja ne uključuje cijeli CNC stroj, već samo simulacija i verifikacija na temelju alata, radnog komada te ostalih elementa koji su ključni u provođenju obrade, kao na primjer, stezne naprave. Nakon

potvrdne početne simulacije pokreće se detaljnija simulacija sa cijelim CNC strojem i ostalom opremom. Razlog zašto se cijeli model stroja ne koristi za simulaciju tijekom programiranja je otežano praćenje putanje alata zbog zaklanjanja od ostalih elemenata CNC stroja. Postoji mogućnost posebnog programa koji provodi samo simulaciju i verifikaciju programa. Izgradnja simulacije jednog takvog simulacijskog okruženja biti će prikazan u ovom radu. Cijeli CNC sustav se može podijeliti na nekoliko elemenata. Elementi su dio arhitekture upravljačkog sustava CNC stroja. U ovom radu najviše je stavljen naglasak na softverske elemente. Na kaju će biti spomenuti i hardverski elementi arhitekture upravljačke jedinice. Prvi element arhitekture je korisničko sučelje ili GUI (engl. *Graphic User Interface*) koje je poveznica između operatera stroja i pozadinske programske upravljačke logike. Svaki proizvođač ima drugačiji izgled korisničkog sučelja, ali osnovne postavke i način rada posjeduju sva korisnička sučelja. Prije samog rada sa G-kodom i korisničkim sučeljem, vrši se unošenje parametara i postavki određenih dijelova stroja, kao, na primjer, postavke servo sustava, glavnog obradnog vretena, granice radnog prostora stroja, parametri izvođenja ciklusa itd. Korisničko sučelje omogućuje upravljanje sa osima stroja, radnim vretenom, paljenje rashladnog sredstva itd. Najbitniji dio korisničkog sučelja je prijenos G-koda, i, promjene G-koda. Iako se cijeli G-kod može pisati unutar korisničkog sučelja, važno je reći da se ovdje podrazumijeva prijenos G-koda iz nekog CAM programa. Vrši se samo eventualna izmjena nekog dijela programa. Najčešće se radi o brzini gibanja ili posmaku. Kod naprednijih upravljačkih jedinica moguće je izvršiti programiranje korištenjem ciklusa, gdje je unošenje parametara riješeno grafičkim putem. Tako stvoren program nije G-kod program. Dio o korisničkom sučelju neće se posebno analizirati, kao niti razvoj algoritama, jer korisničko sučelje nema donošenja logičkih odluka - koncentracija je na logici upravljanja. Interpreter je samo jedan od blokova koji sudjeluje u protoku podataka i upravljanju strojem. Iterpreter je samo jedan element jezgre numeričkog upravljanja numeričke kontrolne jezgre (engl. *Numerical Control Kernel*). Može se reći da je jezgra numeričkog upravljanja „mozak“ upravljačke jedinice i, zadužena je za sinkroniziranje svih radnji. Informacije dobivene intrepretacijom dolaze do interpolatora koji proračunava pozicije i brzine u svakoj vremenskoj iteraciji. Raspodjeljuje brzine po osima, ovisno o inkrementalnom pomaku između trenutne točke, i, odredišne točke. S obzirom da korištene brzine nije moguće trenutno postići, mehanički dijelovi stroja se ne mogu trenutno početi gibati određenom brzinom. Potrebna je odrediti ubrzanja i usporenja, provesti interpolaciju ubrzanja i usporenja nad brzinom. Postoje dva osnovna načina interpolacije ubrzanja i usporenja. Jedan od načina interpolacije ubrzanja i usporenja je nakon generirane brzine, dok je drugi prije generirane brzine, odnosno uslijed interpolacije brzine vrši se interpolacija ubrzanja i usporenja. Posljednji element jezgre numeričkog upravljanja je kontroler pozicije i/ili brzine koji na temelju generiranog profila (koji je opisan vremenskim intervalima, brzinom, pozicijskim intervalima,

ubrzanjima i usporenjima) pokušava izjednačiti takav profil sa stvarnim veličinama. Kontroler, najčešće izveden kao PID kontroler, upravlja servo sustavima osi kako bi gibanja pojedine osi odgovarale gibanjima opisanim profilom brzine/ubrzanja.

5.2.1. Funkcija interpretera

Svrha interpretera u upravljačkoj jedinici je prevođenje G-koda u naredbe koje razumije upravljačka jedinica. Svrha G-koda je lakša interpretacija čovjeku, odnosno, prilagođen je, kako bi čovjek lakše interpretirao instrukcije gibanja stroja. Interpreter prevodi naredbe sustavu upravljanja kako bi sustav izvršio gibanje. Sustav koristi podatke koje interpreter generira. Sam rad i svrha interpretera je jednostavna, ali, implementacija i razvoj algoritma - nisu. Svrha nije samo prevođenje program nego i provjera sintakse G-koda, kao i briga o modalnim i ne-modalnim naredbama, gdje, kod modalnih naredbi interpreter mora pamtiti numeričke vrijednosti za slijedeće blokove, a resetirati numeričke vrijednosti, kod ne-modalnih naredbi. G-kod je predočen ASCII znakovima kako bi bio razumljiv korisniku, kao operateru upravljačke jedinice. Interpolator ima zadatak čitanja ASCII znakova te njihovog prevođenja i spremanja u internu memoriju u obliku *buffer-a* koji dijeli sa interpolatorom. *Buffer* konkretno, u ovom slučaju, znači set varijabli koje služe za internu pohranu trenutnog i unaprijednog čitanja informacija iz G-koda. Sam naziv i svrha *buffer-a* je određena vrstom programske riječi, odnosno varijabla određuje pročitanu adresu a numerička vrijednost koja se nalazi uz adresu je vrijednost koja je pohranjena unutar predviđene varijable. Interpreter prevodi i priprema naredbe G-koda sinkronizirano sa interpreterom. Ovdje ključnu ulogu ima brzina prevođenja i pripreme naredbi, jer je, kod malih pomaka i velike brzine gibanja, potrebna i velika brzina čitanja. Ovisno o načina rada stroja, podsjetnik na naredbe G91 i G94 koje određuju da li će alat zaustaviti svoje gibanje u odredišnoj točki, i, time omogućiti određeni vremenski interval da interpreter pročita/prevede/pripremi podatke iz slijedećeg bloka. Takav način biti će detaljni opisan kod razvoja algoritma interpretera i interpolatora, jer je i algoritam puno jednostavnije objasniti i pratiti njegov ciklus rada. Kod naredbe kontinuiranog gibanja, interpreter mora analizirati i pripremiti nekoliko blokova programa. Tako da, kod velikih brzina i malih pomaka osi, može se dogoditi zaustavljanje gibanja, kako bi interpreter, a najviše interpolator, mogao proračunati parametre gibanja za slijedeći blok. Kao što već spomenuto, rad interpretera je poprilično jednostavan u odnosu na ostale elemente upravljačke jedinice, ali su dizajn i implementacija otežani zbog provjere sintakse tijekom čitanja. Odnosno, interpreter ima veliku ulogu u detekciji grešaka programa. Način uočavanja grešaka i „osjetljivost“ na pogreške ovisi o proizvođaču upravljačke jedinice.

Interpretacija G-koda sastoji se od nekoliko modula:

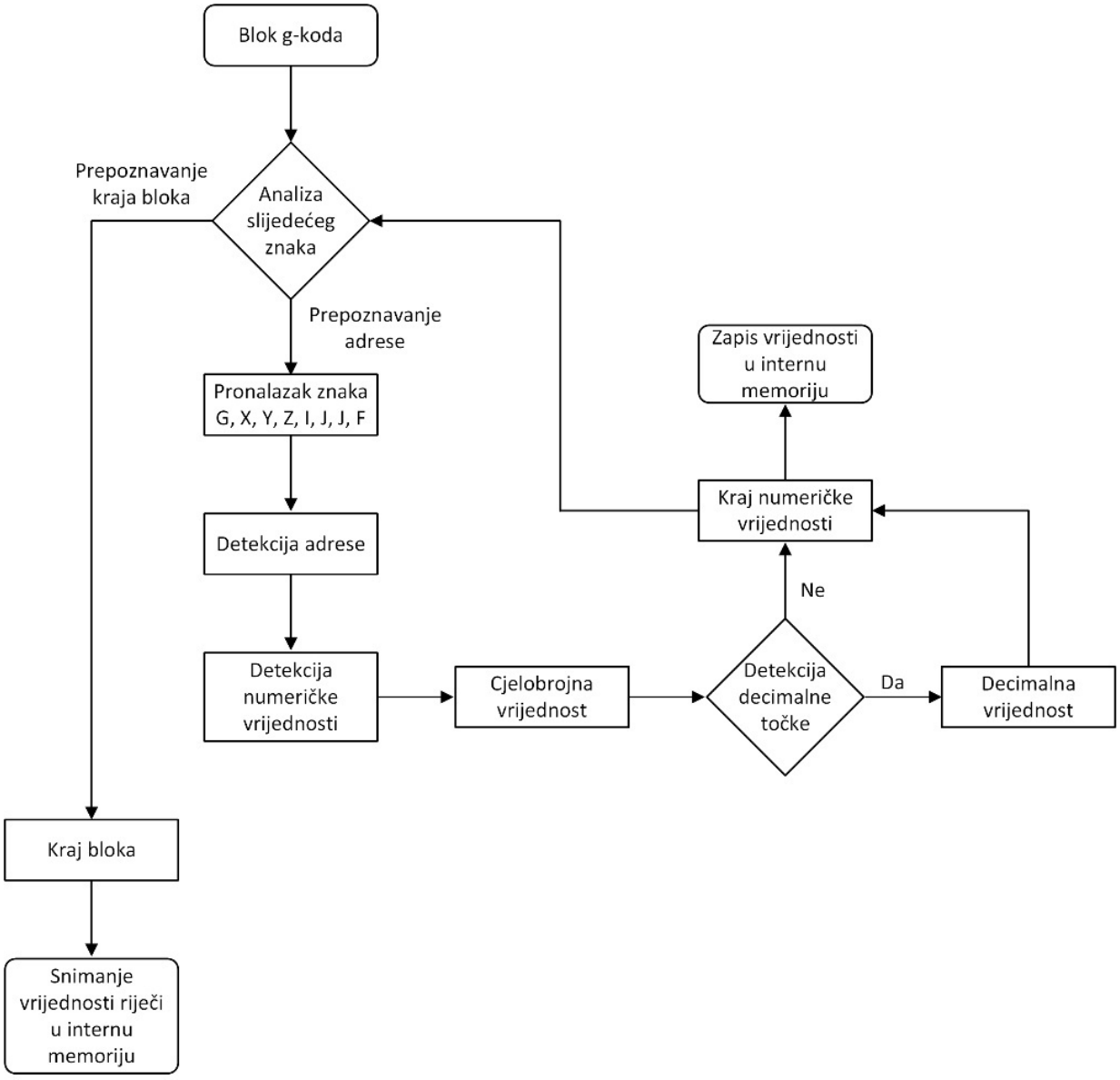
- **Parser** – modul koji čita kod, liniju po liniju. Prolazeći kroz svaki znak u bloku, raspoznaje adrese bloka. Unutra bloka prepoznaje riječi bloka, te, riječ razlaže na adresu i numeričku vrijednost. Vrši provjeru sintakse, da li je blok pravilno formatiran. Provjera logičkih petlji i grananja ako se koriste u programu.
- **Izvršitelj** – Izvršava naredbe koje su dobivene Parser-om. Naredbe su sada razložene prema načinu gibanja, broju okretaja vretena, brzini gibanja, te, ostale sa pripadajućim numeričkim vrijednostima. Pohranjuje ih u memoriju upravljačke jedinice za daljnju obradu, nakon završetka čitanja koje je uvjetovano pročitanim blokom.
- **Generator putanje** - modul koji računa pozicije svake koordinatne s obzirom na „pročitani“ blok. Računa koordinate u odnosu na koordinatni sustav stroja ili globalni koordinatni sustav uzimajući u obzir vrstu gibanja, vrstu kompenzacije, ograničenja stroja itd. Ovisno od korištenog apsolutnog, ili inkrementalnog sustava, proračunava poziciju alata u odnosu na već sve spomenute utjecaje gdje se mora nalaziti pozicija alata u nekom globalnom koordinatnom sustava, te, pohranjuje takvu vrijednost. Kod čitanja numeričke vrijednosti uzima u obzir koja vrsta koordinatnog sustava je korištena, te dodjeljuje mjerne jedinice.
- **Macro izvršitelj** – intrepretira macro-naredbe, rutine i ostale specijalne funkcije koje nisu proizašle od strane proizvođača CNC stroja. Odnosno, ovo je dio programa pisan od strane operatera u programskom jeziku koji je niže strukture od G-koda ili nekog od ostalih numeričkih upravljačkih programa. Koristi se za veće mogućnosti upravljanja strojem i njegove periferije, što nije moguće izvesti G-kod programom. Takvi jezici najčešće sliče, ili jesu programski jezici BASIC i PASCAL.
- **Upravljanje greškama** – ako se pojavi odstupanje od strukture G-koda (greška sintakse, nepravilno zadavanje instrukcija, nepodržane instrukcije) nakon detekcije se poduzimaju mjere generiranja greške kako bi operater bio obavješten. Po potrebi, može se zaustaviti izvršenje programa i zaustavljanje stroja, ili nastavak rada, uz ispravak greške.

Način rada interpretera i nekih od modula biti će prikazan na algoritmu koji je pisan u programskom jeziku C++ u sljedećem potpoglavlju te će biti dodano objašnjeni. Bitno je naglasiti da je G-kod, koji je moguće pročitati, a samim time i izvršiti, ograničen sa vrstama instrukcija. Ovdje će biti prikazane samo naredbe koje su neophodne za gibanje CNC stroja. Razlog je što je ova upravljačka jedinica zamišljena za upravljanje CNC strojem u operacijama obrade kompleksnih površina gdje generiranje programa ovisi o CAM programskim alatima. Te se razne

naredbe, kao kompenzacije alata, mogu izvesti programskim alatom, bez potrebe naredbe za kompenzaciju.

5.3. Algoritam interpretera

Svrha interpretacije blokova G-koda je ekstrakcija pojedinih elemenata bloka. Interpretator mora razlučiti pojedine riječi u bloku, te prema adresi dodijeliti brojčanu vrijednost. Svaka adresa u algoritmu predstavljena je varijablom, odnosno, listom varijabli gdje je svaki element liste jedna numerička vrijednost pojedine riječi u bloku. Algoritam prolazi svaki znak koji se nalazi unutar tekstualnog uređivača GUI sučelja, gdje je potrebno kopirati G-kod. Pokretanje rada izvršenja G-koda i gibanja stroja, započinje čitanjem G-koda.



Slika 5.10 Dijagram toka funkcioniranja interpretera

Čitanje se ne izvodi doslovno blok po blok, nego algoritam intrepretera čita G-kod redom dok ne naiđe na kraj reda, što označava kraj bloka. Algoritam interpretera čeka naredbu za čitanje slijedećeg bloka.

```
for(;i<gcode_text.length(); i++)
```

Početak petlje koji ispituje svaki znak, odnosno, svaki znak predstavlja adresu G-koda unutar riječi bloka. Unutar bloka, osim slova i brojeva, razmaci, početak i kraj reda, te ostali posebni znakovi koji nisu vidljivi, korisniku se broje kao znakovi. Svi ti znakovi u ACSII formatu imaju određenu numeričku vrijednost - time se mogu registrirati i razmaci i kraj svakog bloka. Algoritam prolazi kroz petlju sve dok petalja ne dosegne broj znakova koliko ih se nalazi u tekstu. Prije nego dosegne posljednji znak, inerpreter mora završiti prepoznavanaje adresa dosegom adrese M i numeričke vrijednosti 30. M30 označava kraj programa. U internu memoriju se ne kopira jedan blok, već se čitanje provodi nad cijelim tekstom koji je kopiran iz tekstualnog urednika GUI sučelja. Početak programa započinje traženjem adrese riječi. Prvi i jednostavniji dio programa, koji intrepretira vrijednost vrste gibanja „G“, jednostavniji je„ jer ova adresa, kao brojčanu vrijednost nema predznake niti decimalne vrijednosti, odnosno, nema specijalne znakove. U ovom slučaju intrepretacije G-koda može imati samo 4 numeričke vrijednosti. 0 i 1 za linearna gibanja i 2 ili 3 za kružna gibanja. Adresu „N“, rednog broja bloka, koji se nalazi prije adrese vrste gibanja „G“, nije potrebno bilježiti jer algoritam bilježi redni broj bloka, ali se vrlo lako, opcionalno, može ubaciti. Adresa „N“, broja reda, služi više operateru za snalaženje u programu prije samog pokretanja G-kod programa na upravljačkoj jedinici.

```
if((int)gcode_text[i].unicode() == 71) // G
{
for(i++; (int)gcode_text[i].unicode() < 58 &&
(int)gcode_text[i].unicode() > 47; i++)
{
temp_read = temp_read * 10 + ((int)gcode_text[i].unicode() - 48);
}
read_gLine.G = temp_read;
temp_read = 0;
}
```

Programska petlja koja provjerava sve znakove G-koda, kada pronađe ASCII znak koji odgovara vrijednosti slova „G“, pridijeli vrijednost koja se nalazi iza znaka. U ovom slučaju, nije potrebno provjeravati daljnje znakove, jer, iza adrese „G“ mora se nalaziti samo jedan broj. Ali, zbog univerzalnosti programskog algoritma koristi se petlja koja provjerava sve numeričke znakove iza znaka „G“, sve numeričke znakove između 0 i 9 koji odgovaraju vrijednostima ASCII koda od vrijednosti 48 do 57 - ovo je ujedno i detekcija pogreške. Varijabla u kojoj je spremljena

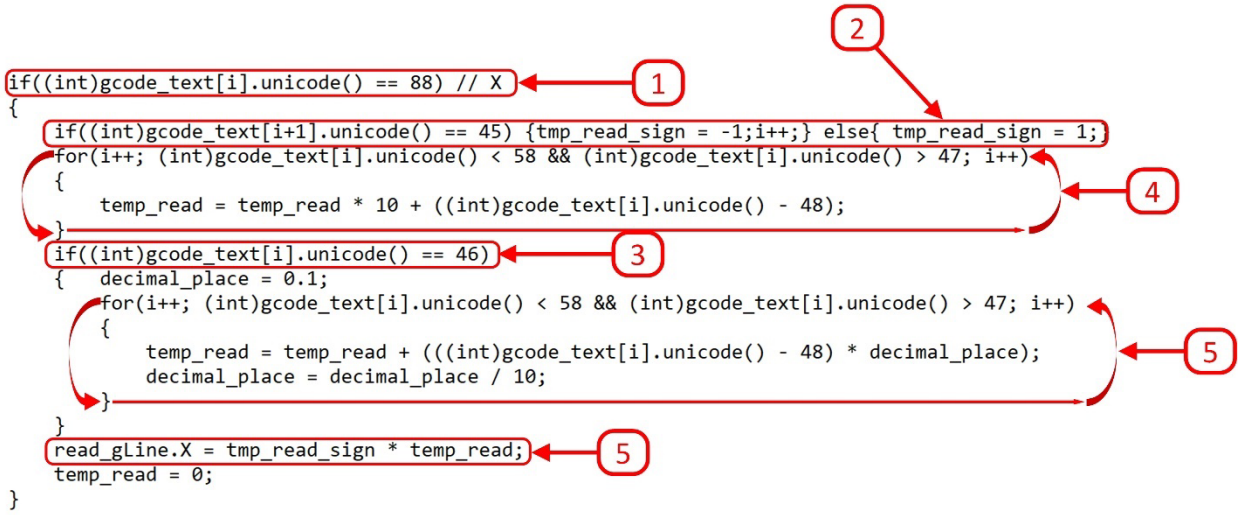
numerička vrijednost vrste gibanja naziva se *read_gLine.G* čija se vrijednost dijeli sa slijedećim elementom - interpolatorom. Odnosno, dijeli se cijeli objekt nazvan *read_gLine* u kojem se nalaze pojedine pročitane adrese kako bi se povećala modularnost C++ programa, tako da, varijable koje nemaju prefiks *read_gLine.,* koriste se lokalno i ne dijele se sa interpolatorom. Sve varijable su opisno nazvane na engleskom jeziku. Takav način detekcije riječi, detekcija adresa i numeričkih vrijednosti ujedno je i provjera sintakse.

Dio algoritma za prepoznavanje X osi sa dodjelom numeričke vrijednosti: Prije nego se započne dodjela pozicija koordinatnih osi potrebno je razlučiti kojoj osi koordinatnog sustava pripadaju. Ishodište pozicija pojedinih točaka vezane su za koordinatni sustav radnog komada. Ishodište takvog koordinatnog sustava je trenutna pozicija vrha alata prilikom pokretanja programa.

```
if((int)gcode_text[i].unicode() == 88) // X
{
     if((int)gcode_text[i+1].unicode() == 45) {tmp_read_sign = -
1;i++;}
     else{ tmp_read_sign = 1;}
     for(i++; (int)gcode_text[i].unicode() < 58 &&
(int)gcode_text[i].unicode() > 47; i++)
     {
         temp_read = temp_read * 10 + ((int)gcode_text[i].unicode() -
48);
     }
     if((int)gcode_text[i].unicode() == 46)
     {   decimal_place = 0.1;
         for(i++; (int)gcode_text[i].unicode() < 58 &&
(int)gcode_text[i].unicode() > 47; i++)
     {
     temp_read = temp_read + (((int)gcode_text[i].unicode() - 48) *
decimal_place);
     decimal_place = decimal_place / 10;
     }
     }
     read_gLine.X = tmp_read_sign * temp_read;
     temp_read = 0;
}
```

Kod ove adrese numerička vrijednost može biti i negativna vrijednosti i imati decimalnu točku. Algoritam nakon detektiranog znaka „X“, ispituje postojanje znaka „-„ kao predznak minus kasnije cijelu vrijednost množi sa -1. Nakon ispitanog predznaka varijablom tmp_read_sign, kreće detekcija brojčane vrijednosti, prvo cjelobrojnog dijela unutar varijable *temp_read*. Svaki slijedeći znak u cjelobrojnoj vrijednosti mora se pomnožiti sa 10, kako bi se vrijednost uvećavala. Nakon dosega decimalne točke ili znak „.“, sve vrijednosti dijele se sa 10 nakon svakog znaka - ovo se odvija unutra varijable *decimal_place,* tako da se decimalna vrijednost numeričke vrijednosti odvoji od cjelobrojne vrijednosti. Unutar petlje svake adrese ispituje se sve dok ima znakova koji

imaju ASCII vrijednost između 48 i 57, ili, ASCII vrijednosti za decimalnu točku. Svaku znak koji je različit od navedenih znakova smatra se krajem adrese, detektira se razmak ili početak novog reda.



Slika 5.11 Algoritam čitanja numeričke vrijednosti X osi

Oznake na slici su sljedeće:

1. Detekcija adrese X osi
2. Provjera predznaka X osi
3. Detekcija decimalne točke
4. Petlja za čitanje cjelobrojne numeričke vrijednosti
5. Petlja za čitanje decimalne numeričke vrijednosti
6. Spremanje numeričke vrijednosti X osi u internu memoriju varijablom *read_gLine.X*

Vrijednost pozicije po X osi spremljena je u varijablu *read_gLine.X*. Daljnjeg postupka manipulacije sa vrijednostima pozicije nema, jer algoritam ne uključuje kompenzacije alata koje bi slijedile, kao ni pomaci ili rotacije koordinatnih sustava. Algoritam čita „čisti“ G-kod, izvedba interpretera je nešto jednostavnija jer nema potrebe niti za daljnjim kalkulatorom stvarne pozicije. Algoritam na ovaj način provjerava za svaku adresu numeričku vrijednost. Znak za početak novog reda je istovremeno i detekcija kraja bloka te početak interpolacije putanje alata na temelju vrijednosti dobivanih interpreterom. Vrijednosti slijedećeg bloka u adresama „X“, „Y“, „Z“, „F“ ostaju iste, ako drugačije nije navedeno u bloku. Stroj kada dosegne vrijednost u osi X ostaje u toj poziciji bez obzira što u slijedećem bloku nije navedena os X. Ali, adrese I i J za centar kružnih gibanja, moraju se resetirati ako u tom bloku nije specificirano kružno gibanje - takve naredbe su ne-modalne te zahtijevaju definiranje u svakom bloku programa. Interpreter očekuje čitanje

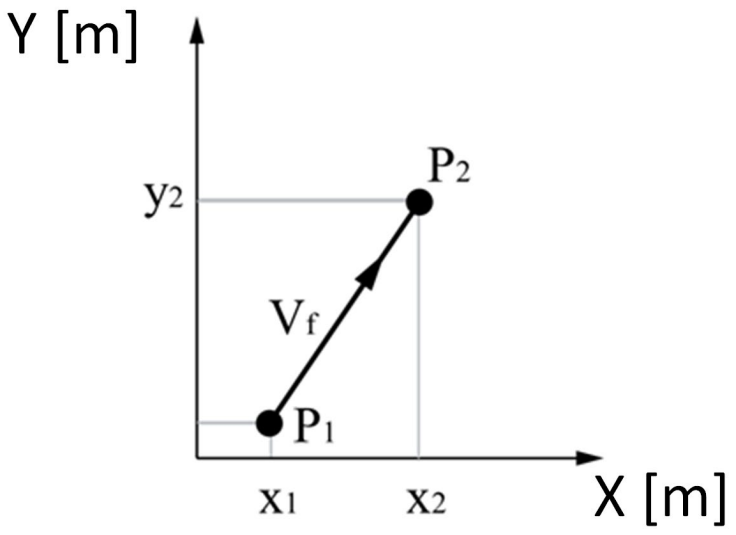
znakova redom početak bloka „G“, „X“, „Y“, „Z“, „F“ , „I“, „J“ kraj bloka. Svako odudaranje od ovog redoslijeda generira grešku, svakako ako su svi znakovi potrebni za definiranu vrstu gibanja.

6. Interpolator

Interpolator služi za raspodjelu gibanja za svaku os, kako bi sve osi tvorile interpolirano gibanje alatom, koji je povezan sa osima. Potrebno je razlikovati raspodjelu gibanja, odnosno raspodjelu brzine po osima. Osi ne mogu trenutno postići trenutnu brzinu i ostvariti prestanak gibanja, postići brzinu nula. Određivanje ubrzanja i usporenja se provodi nakon proračuna brzina, čime se svakako produžava vrijeme gibanja.

6.1. Osnovne radnje interpolacije

Interpolator u kontekstu CNC strojeva odnosi se na dio upravljačke jedinice koji interpretira naredbe iz G-kod programa i generira signale za kretanje osi stroja. Interpolator ima zadatak izračunati i upravljati putanjom gibanja alata ili obratka na temelju zadanih parametara. S obzirom da svaki CNC stoj upravlja sa minimalno dvije osi konkretno, interpolator izračunava između kojih točaka i s kojom brzinom se treba kretati alat ili obradak, kako bi se postigla željena putanja. Sinkronizira sve osi zajedno kako bi pomak svake osi ostvario željenu putanju. To uključuje generiranje putanje između točaka koordinata, brzinskih profila i interpolacija između tih točaka kako bi se postigao glatki prijelaz. Interpolator ukupnu putanju gibanja mora raspodijeliti na osi koji se koriste - svaka os mora opisati određeni segment puta. Svaki segment pojedine osi određuje ukupni prijeđeni put.



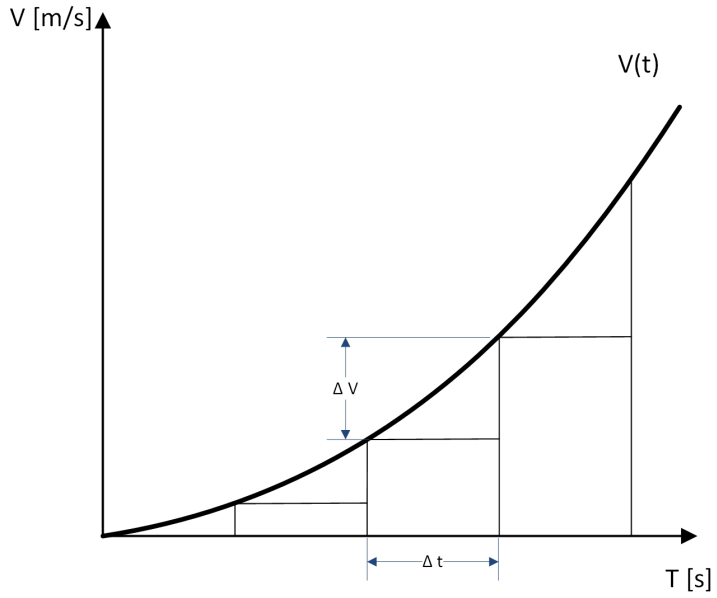
Slika 6.1 Gibanje dvije osi

Uz put, koji mora ostvariti svaka os, interpolator mora odrediti kojim brzinom će se svaka os gibati, a da pritom njihova gibanja ostvaruju brzinu koja je zadana posmakom. Interpolator se

također brine o drugim aspektima gibanja, kao što su ubrzanje i deceleracija, zaustavljanje i startanje, promjena smjera gibanja i kompenzacija geometrijskih pogrešaka stroja. Njegova uloga je osigurati precizno i kontrolirano gibanje alata ili obratka prema G-kod programu. Interpolacija može biti linearna, kružna (za kružne putanje), spiralna (za spiralne putanje) i interpolacije NURBS linijama. Opći cilj interpolatora je postići glatko, precizno i efikasno gibanje alata ili obratka na CNC stroju. To omogućuje izvođenje složenih operacija glodanja s visokom točnošću i kvalitetom obrade, postizanje dimenzija koje su identične onima propisanim dokumentacijom ili izvornim trodimenzionalnim modelom. Interpolirane putanje alata moraju biti što sličnije idealnim putanjama alata. Raspodjela pozicija i brzina gibanja svake od osi, moraju biti u granicama sposobnosti stroja. Odnosno, brzina ne smije prekoračiti maksimalne brzine koje pojedini servo motor može ostvariti. Svaka interpolacija između dvije točke uzrokuje određenu grešku krajnje pozicije. Interpolator mora osigurati da se greška ne akumulira, jer G-kod može imati na tisuće blokova, a u svakog bloku se izvodi promjena pozicije. Također, veliki broj točka uzrokovao bi sve veću razliku između zadanih točaka i točaka ostvarenih interpolacijom.

6.2. Hardverska interpolacija

Glavna podjela interpolacije može se izvesti prema načinu provođenja interpolacije. Podjela se vrši hardverski i softverski. Hardverska implementacija interpolatora će biti samo ukratko opisana, dok će softverska interpolacija biti detaljnije opisana kroz primjere i razvoj algoritma upravljanja. Hardverska interpolacija koristi elektroničke komponente za generiranje impulsa pomoću kojih je moguće izvesti velike brzine interpolacije, ali je modifikacija logike, odnosno algoritma, otežana. Najznačajnija metoda hardverske interpolacije je DDA (engl. *Digital Differential Analyzer*) ili digitalni diferencijalni analizator. Za pojašnjenje rada DDA hardverske metode interpolacije uzimaju se samo dvije osi u ravnini prilikom linearnog gibanja. Prevaljeni put je površina ispod funkcije brzine, što znači: u diskretnom sustavu je moguće put S dobiti sumiranjem svih brzina Vi sa vremenskom intervalom Δt.



Slika 6.2 Funkcija brzine u diskretnom sustavu

$$S(t) = \int_0^t V \cdot \Delta t = \sum_{i=1}^{k} V_i \cdot \Delta t \qquad (1)$$

Funkcija putanje S(t) jednaka je zbroju površine ispod krivulje ili zbroju svih pojedinačnih brzina pomnoženih sa vremenskim intervalom. To znači da je svaki slijedeći pomak puta, u određenom vremenskom intervalu, jednak prošlom putu plus promjeni putanje u trenutnom vremenskom intervalu. Sk je promjena puta u vremenu.

$$S_k = S_{k-1} + \Delta S_k = S_{k-1} + V_k \cdot \Delta t \qquad (2)$$

Na isti način može se izračunati trenutna brzina, brzina u prošlom vremenskom intervalu plus trenutna promjena brzine. Interpolator se sastoji od dva registra. Jedan registar, zadužen za brzinu, može uvećavati ili smanjivati svoje stanje V prema ulaznoj veličini promjene brzine. Registar brzine povezan je sa registrom Q. Registar je određene veličine i svaki preljev registra se detektira kao promjena puta ΔS. Svaka promjena puta mora se sumirati posebnim registrom. Princip rad se svodi na propusnost registra Q koji kao ulaznu vrijednost prima konstantnu frekvenciju, koja je obrnuto proporcionalna vremenskoj konstanti. Generiranje impulsa iz DDA registra stvara se preljevom registra, kako bi frekvencija generiranja impulsa iz DDA registra bila manja od osnovne frekvencije. DDA se ponaša kao djelitelj frekvencije za svaku od osi. Osnovna dužinska jedinica definira najmanju promjenu pozicije generiranjem jednog impulsa.

6.2.1. Softverska interpolacija

Razvojem računalnih znanosti algoritmi interpolacije se mogu izvesti programski, što danas i jest slučaj. Dalje će biti prikazane metode programske interpolacije za linearna gibanja. Postoje dvije osnovne grupe softverske interpolacije:

-referentna impulsna metoda, i,
-metoda uzorkovanja podataka.

Referenta impulsna metoda je slična, štoviše, jedna njena metoda se naziva softverski DDA interpolator - programska izvedba DDA interpolatora koja s procesorom upravljana generira impulse za pojedine osi prema servo sustavu. Svaki impuls odgovara osnovnoj dužinskoj jedinci (BLU - engl. *Basic Lenght Unit*). Ako je osnovna dužinska jedinica 0.001 mm, 1000 generiranih izazvat će pomak od 1 mm.

Slijedeća metoda impulsa metoda je **interpolator stepenaste aproksimacije** – gdje je putanja gibanja predstavljana matematičkom funkcijom. U svakoj iteraciji uspoređuje trenutne vrijednosti po X i Y osi, ubacuju ih u funkciju putanje, čije je rješenje odluka gdje se nalazi pozicija točke u odnosu na funkciju. Ako je rješenje funkcije pozitivan broj potreban je pomak po jednoj osi na primjer X osi, ako je rješenje negativno potreban je pomak po drugoj osi na primjer Y osi. Ako je rješenje 0 - nema pomaka. Problem sa ovom metodom je što zahtjeva jako puno iteracija za preciznu aproksimaciju funkcije, te, u svakoj iteraciji pomak je samo po jednoj osi, nema simultanog pomaka.

Metoda direktnog traženja radi slično kao prijašnja metoda, ali ova metoda traži najkraći put do slijedeće točke, ispitivanjem svih mogućnosti. To ima za posljedicu simultano gibanje obje osi. Svaka trenutna točka može imati 4 različita pomaka u odnosu na prijašnje stanje. Pomak osi X za jednu osnovnu dužinsku jedinicu u pozitivnom, ili negativnom, smjeru i pomak osi Y za jednu osnovnu dužinsku jedinicu u pozitivnom ili negativnom smjeru. Interpolacijske metode uzorkovanja podataka, kojom se koriste moderni CNC strojevi. Metoda se svodi na uzorkovanje ukupne udaljenosti koju je potrebno prevaliti iz jedne točke u drugu. Potrebno je izračunati prevaljeni put svake od osi unutar jednog segmenta iteracije.

6.3. Osnovni profili gibanja

Kod CNC glodalica, osnovni profili gibanja odnosi se na vrstu putanje koje opisuje alat. **Linearni profil gibanja** najčešće se koristi za gibanje kod CNC stroja. Gibanje alata se izvodi između dvije točka ravnom linijom. Koristi se kod gibanja, kada je alat u radnom zahvatu, i, za brza gibanja za pozicioniranje u početnu točku obrade. Najjednostavniji je način upravljanja jer zahtjeva najmanje pozadinske logike za izvođenje gibanja. Sve ostale vrste putanja gibanja, ili željene putanje koje su opisane krivuljama višeg reda, mogu se segmentirati na beskonačno puno linearnih gibanja, te se uz gubitak točnosti mogu izvesti svi ostali profili. **Kružna gibanja** koristi se za izvođenje kružnih ili lučnih putanje alata ili obratka. Definiranje i smjer kružnih gibanja je već pokazan. Kružna gibanja je moguće ostvariti samo simultanim gibanjem dvije osi, što znači da je putanju moguće ostvariti samo dvodimenzionalno. Odabir ravnine u kojoj se odvijaju kružna i lučna gibanja ostvaruje se odabirom aktivne ravnine. U kombinaciji sa linearnim gibanjem također je moguće provesti (uz određenu toleranciju) aproksimirati putanje koje su opisane krivuljama višeg reda. **Spiralna ili cilindrična** putanja koristi se za izvođenje spiralne putanje alata ili obratka. Gibanje zahtjeva gibanje sve tri osi simultano - dvije osi se gibaju kružno u osi C (rotacija oko osi Z) dok se treća os giba linearno po Z osi. Gibanje se opisuje radijusom i korakom uspona. Spiralno gibanje se koristi kod narezivanje navoja ili prodiranje alata u predmet obrade radnim zahvatom. Najkompleksnija gibanja koja se mogu izvesti su putanje koje su opisane **slobodnim krivuljama** ili krivuljama više reda. Upravljačka jedinica ne može interpolirati putanju gibanja iz parametara koji opisuju takvu vrstu putanje. Zbog same kompleksnosti, za matematički opis takve krivulje, upravljačke jedinice najčešće zbog kompleksnosti ne interpoliraju ovu vrstu putanje. Putanja se segmentira na puno manjih dijelova koji se aproksimiraju linearnim i kružnim putanjama koje se lakše interpoliraju, i, potrebna je manja procesorska moć procesorske jedinice koja provodi interpolaciju. Za obradu slobodnih površina gdje je potrebno zadovoljiti dva uvjeta. Jedan uvjet je velika brzina obrade i točnost obrađene površine koju je teško postići segmentacijom površine. U tom slučaju se koristi NURBS interpolacija, odnosno NURBS linije, slobodno oblikovane krivulje koje su definirane kontrolnim točkama, vektorima i težinskim koeficijentima. To su parametri gibanja, kao što je točka po X, Y i Z os parametra za linearno gibanje. Upravljačka jedinica direktno čita takve parametre i izvodi interpolirano gibanje. NURBS je samo jedan način predstavljanja takvih slobodno oblikovanih površina, ostale su kubične krivulje, Bezier krivulje, B-krivulje. Kao što je već nekoliko puta naglašeno, koristi se kod obrade kompleksnih slobodno oblikovanih površina kod izrade kalupa, lopatice turbina, implantati, dijelovi motora itd. Ovo su osnovne vrste interpoliranog gibanja koje se koriste kod CNC glodanja. Bitno je razlikovati vrste

gibanja, parametre koji opisuju određeno gibanje, i, na kraju, sposobnost same upravljačke jedinice da izvodi određena gibanja.

6.4. Generiranje profila brzine

Generiranje profila brzine kretanja kod CNC glodalica važan je aspekt kontrole gibanja stroja. Profili brzine omogućuju kontrolu brzine gibanja alata ili obratka tijekom različitih faza obrade. Postoje različiti načini generiranja profila brzine kretanja, a neki od njih uključuju stvari objašnjene u nastavku. Kako bi se ostvarilo glatko gibanje stroja, bez naglih trzaja, nakon poznavanja brzina svake od osi potrebno je upravljanje ubrzanjima i usporenjima. Kada je riječ o upravljanju ubrzanja i usporenja, koriste se dvije metode. Jedna je **upravljanje ubrzanjima i usporenjima prije interpolacije** (ADCBI - engl. *Add/Dec Control Before Inerpolation*) i **upravljanje ubrzanjima i usporenjima nakon interpolacije** (ADCAI - engl. *Add/Dec Control After Interpolation*). ADCBI je procesorski zahtjevnija metoda i danas se koristi u modernim CNC strojevima. Potrebna je velika količina memorije zbog pohrane velike količine podataka, ali, upravljanje je puno točnije bez pojave greške. ADCAI metoda je slična interpolaciji, svaka os se upravlja zasebno. Moguća je pojava greške, koja kod metode ADCBI nije moguća jer upravljana putanja je identična onoj zadanoj. Upravljanje ubrzanjima i usporenjima poslije interpolacije – ADCAI, također može biti izvedeno hardverski i programski. Ulazna vrijednost u algoritam upravljanja ubrzanjima i usporenjima su impulsi brzine iz interpolatora, što znači, već određene brzine za svaku os. Algoritam upravljanja zaglađuje promjene u brzini impulsa na početku i kraju.

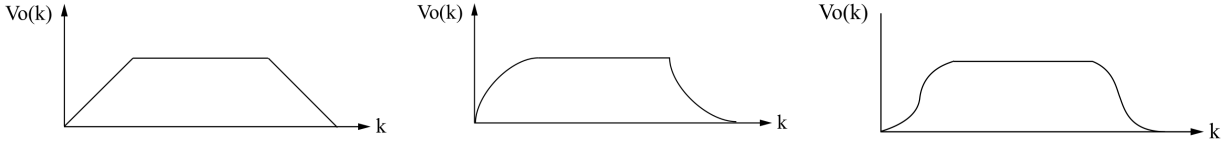
Postoje tri načina upravljanja:

-linearno,

-eksponencijalno upravljanje,

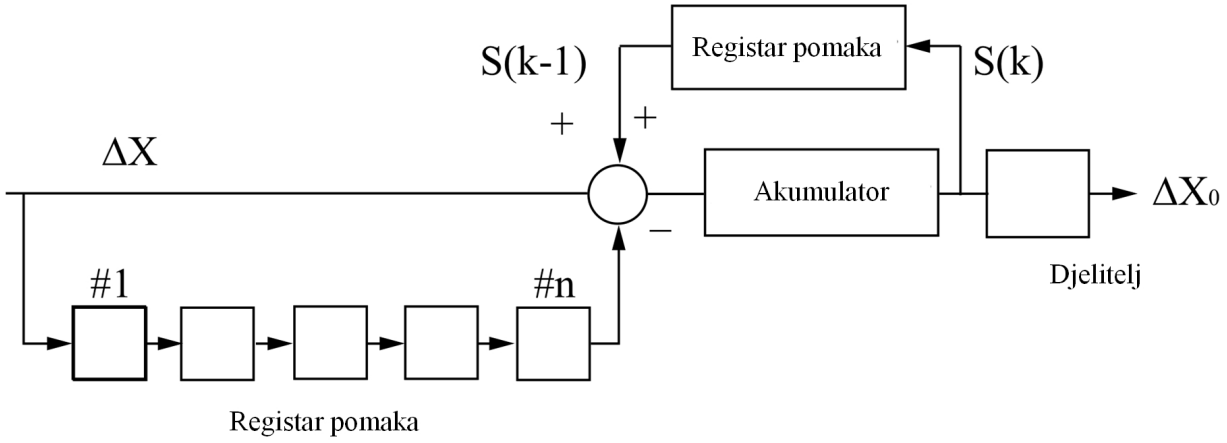
-S-oblikovano upravljanje.

Linearno povećanje i smanjenje brzine je najjednostavnije za proračun i implementaciju - takav puni profil se naziva i trapezni profil, dok su ostali profili puno kompliciraniji za implementaciju, ali uzrokuju manje trzaje pri pokretanju i zaustavljanju osi. Takav trzaj se očituje dvostrukom derivacijom početka i kraja profila, gdje kod linearnog profila derivacija, odnosno ubrzanje i usporenje je konstantno, dok kod S-oblikovanog segmenta ubrzanja i usporenja potrebna je kompleksnija matematička funkcija za opis takvog oblika.



Slika 6.3 Različiti profili brzine: (lijevo) linearni, (sredina) eksponencijalni, (desno) S-oblikovani profil

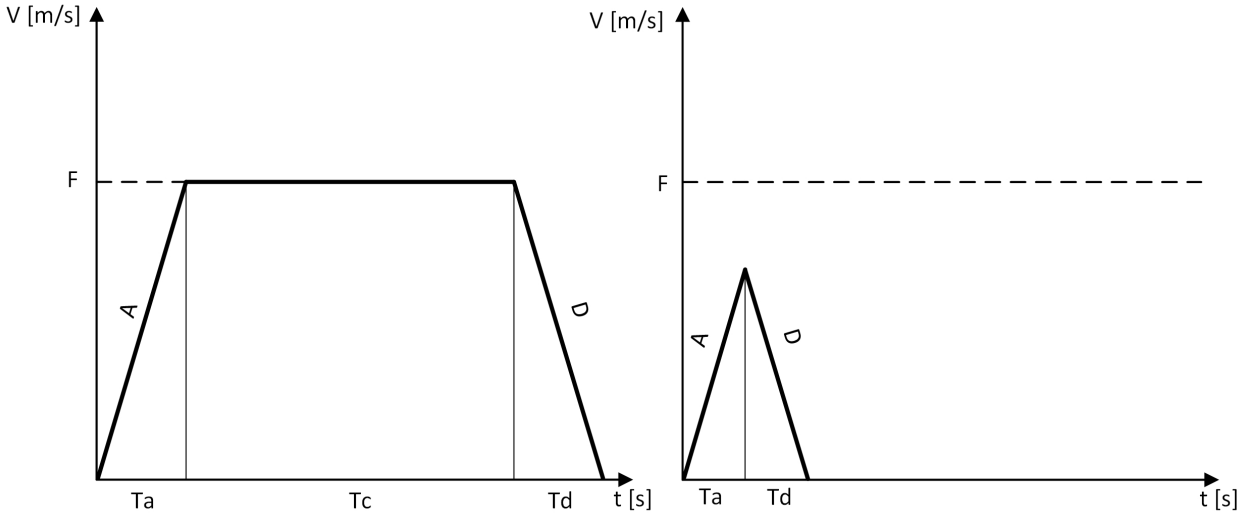
Linearni način upravljanja određuje linearno ubrzanje i usporenje gdje se upravljački dio sastoji od nekoliko registara. Broj impulsa koji ulaze u upravljački dio unutar vremenskog uzorkovanja. Broj izlaznih impulsa je jedak broju izlaznih impulsa. Jedan dio impulsa prolazni kroz nekoliko serijski spojenih registara. Potreban je određeni broj vremenskih uzorkovanja kako bi registri poprimili vrijednost ulaznih impulsa. U istim vremenskim intervalima ulazni impulsi dolaze direktno na sumator, gdje se oduzimaju sa vrijednostima iz serijski spojenih registara. Izlaz iz sumatora spojen je na akumulator koji pamti broj impulsa, i, broj impulsa šalje ponovno na sumator. Kada počnu stizati impulsi određene vrijednosti, na sumatoru će se oduzimati ulazne vrijednosti sa vrijednostima iz serijskih registara kojima je u početnom trenutku vrijednost jednaka nula. To znači da će se samo zbrajati vrijednost direktno spojenih impulsa sa ulaza. Kada se registri popune sa vrijednostima, njihova negativna vrijednost počne stizati na sumator, i oduzimati se sa trenutnim vrijednostima. Razlika impulsa će biti 0 i brzina postaje konstantna. U periodu kada vrijednosti sa serijskih registara ne počinju stizati, izlazni broj impulsa linearno raste, u periodu stizanja serijskih registra izlazna vrijednost se ne mijenja. Kada ulazni impulsi poprime vrijednost nula, izlazna vrijednost počinje linearno opadat dok ne poprimi vrijednost nula.



Slika 6.4 Hardversko upravljanje ubrzanja i usporenja

Kod upravljanja S-krivuljom ubrzanja i usporenja, svaki serijski registar ima određeni koeficijent sa kojim je direktno spojen na sumator. Svaka od vrijednosti serijskog registra se množi sa koeficijentom prije nego se sumira u sumatoru. Upravljanje ubrzanjem i usporenjem prije interpolacije ADCBI – prilikom generiranja profila uzima se u obzir ubrzanje i usporenje. Prilikom generiranja profila brzine moraju biti poznati svi parametri, kao što su dužina puta L, dopušteno

maksimalno ubrzanje i usporenje A (ubrzanje i usporenje je jednako po iznosu pa se može gledati samo ubrzanje), vrijeme iteracije τ(s) te zadani posmak F. Objašnjena je izgradnja profila brzine za linearni trapezni profil brzine. Prije generiranja, potrebna je provjera vrste generiranog profila, da li je određenim ubrzanjem moguće postići zadanu brzinu. Ako je zadana brzina ostvariva, profil brzine imat će jedna dio konstante brzine. Takav profil naziva se **normalni blok profila**. Ako nije ostvariva brzina, zadanu brzinu nije moguće ostvariti - takav blok se naziva **kratki profil brzine**.



Slika 6.5 Profili brzina

Za ostvarenje normalnog bloka profila brzine mora se zadovoljiti jednadžba:

$$\frac{F^2}{2A} + \frac{F^2}{2D} \geq L \tag{3}$$

Ili, ako su ubrzanja i usporenja jednaka A = D, jednadžba se može zapisati i ovako:

$$\frac{F^2}{A} \geq L \tag{4}$$

Površine ispod krivulja ubrzanja i usporenja moraju biti veće od ukupnog prijeđenog puta kako bi se ispunio uvjet normalnog bloka. Ako taj uvjet nije zadovoljen, uz korištenje maksimalnog ubrzanja, nije moguće postići zadanu brzinu koristi se kratki blok profil brzine. Nakon utvrđivanja vrste profila potrebno je izračunati vremena potrebna za svaki od segmenata profila. Kod normalnog profila segment ubrzanja i usporenja može se izjednačiti ako se uzme pretpostavka da je ubrzanje i usporenje jednako. Vrijeme je tada moguće izračunati slijedećom jednadžbom:

$$T_A = T_D = \frac{F}{A} = \frac{F}{D} \tag{5}$$

Vrijeme konstantne brzine moguće je izračunati oduzimanjem ukupnog puta i puta koji os prolazi u segmentima ubrzanja i usporenja.

$$T_C = \frac{L - \frac{F^2}{2A} - \frac{F^2}{2D}}{F} \tag{6}$$

Na sličan način se određuju vremena kod kratkog bloka, samo se koristi maksimalna postiziva brzina, dok konstantnog vremena nema, u tom slučaju. Parametri za određivanje brzine u svakom vremenskom intervalu su pripremljeni. Generiranje profila brzine izvodi se jednadžbom:

$$V_{i+1} = V_i + \tau \cdot A \ (i = 0, 1, 2 \ , \dots . , N_A) \tag{7}$$

Prevaljeni put u svakoj iteraciji jednake je:

$$L_i = \frac{V_{i+1}^2 - V_i^2}{2A} \tag{8}$$

Gdje je:

V_i – brzine u i-toj iteraciji intervala

L_i – prijeđeni put ili pomak u i-toj iteraciji

Broj iteraciji za segment ubrzanja:

$$N_A = \frac{T_A}{\tau} \tag{9}$$

U slučaju usporenja brzina se oduzima od prošlog stanja. Ako upravljačka jedinica radi u načinu rada preciznog stopa završetak svakog profila rezultira brzinom kretanja nula. Svaki slijedeći profil ima početnu brzinu nuka. Analiza, praćenje i testiranje algoritama koji generiraju profil brzine i provode gibanje osi je puno lakše kada svaki blok profila počinje i završava sa brzinom nula. Ako su u pitanju kružna gibanja, potrebno je simultano gibanje obje osi. Brzine se tada izračunavaju preko kutnih brzina ω

$$V_x = F \cdot \cos(\omega \cdot t) \quad (10)$$

$$V_y = F \cdot sin\,(\omega \cdot t) \quad (11)$$

Kutna brzina jednaka je:

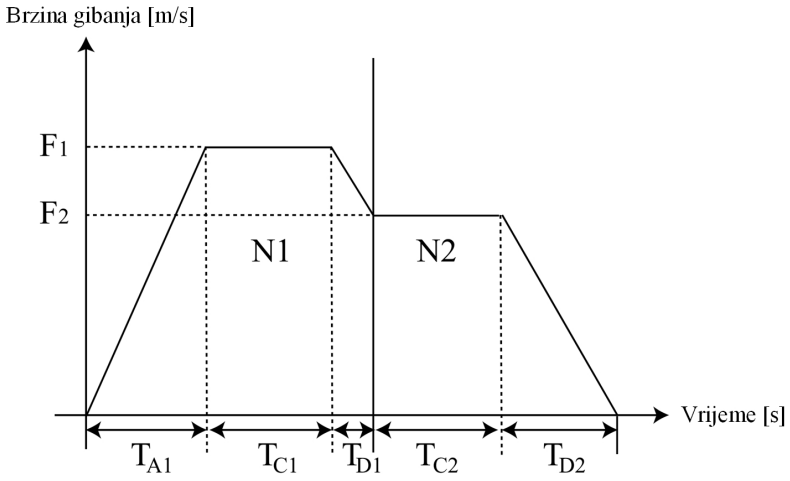
$$\omega = \frac{F}{R} \quad (12)$$

Ubrzanja se tada računaju prema:

$$A_x = -F \cdot \omega \cdot sin(\omega \cdot t) \quad (13)$$

$$A_y = F \cdot \omega \cdot cos(\omega \cdot t) \quad (14)$$

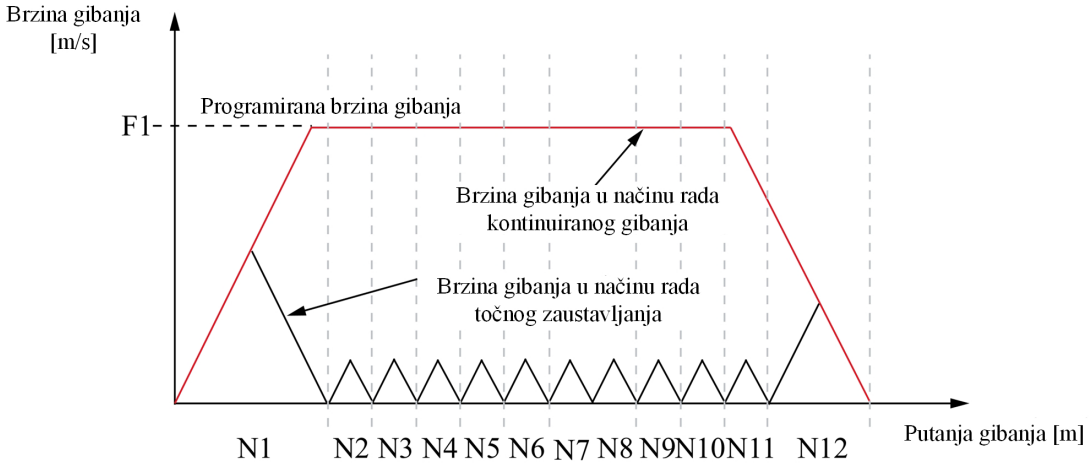
Ako upravljačka jedinica radi u načinu rada kontinuirane brzine, tada početne i krajnje brzine nisu jednake nuli. U tom slučaju potrebno je uzeti u obzir način spajanja, predefinirati sve moguće kombinacije sa dva različita bloka.



Slika 6.6 Profili brzine kontinuiranog gibanja

Profili brzine kontinuiranog gibanja se sastoje od normalnih (trapeznih) i kratkih (trokutastih) profila brzine. U slučaju da je brzina gibanja u sljedećem bloku manja, već u trenutnom bloku je potrebno usporenje brzine, kao što je prikazano na prethodnoj slici. U slučaju veće brzine gibanja sljedećeg bloka, ubrzanje na veću brzinu gibanja se odvija u toku sljedećeg bloka, ne za vrijeme trenutnog bloka. Ovaj proces se također može predočiti prethodno prikazanom slikom, samo je

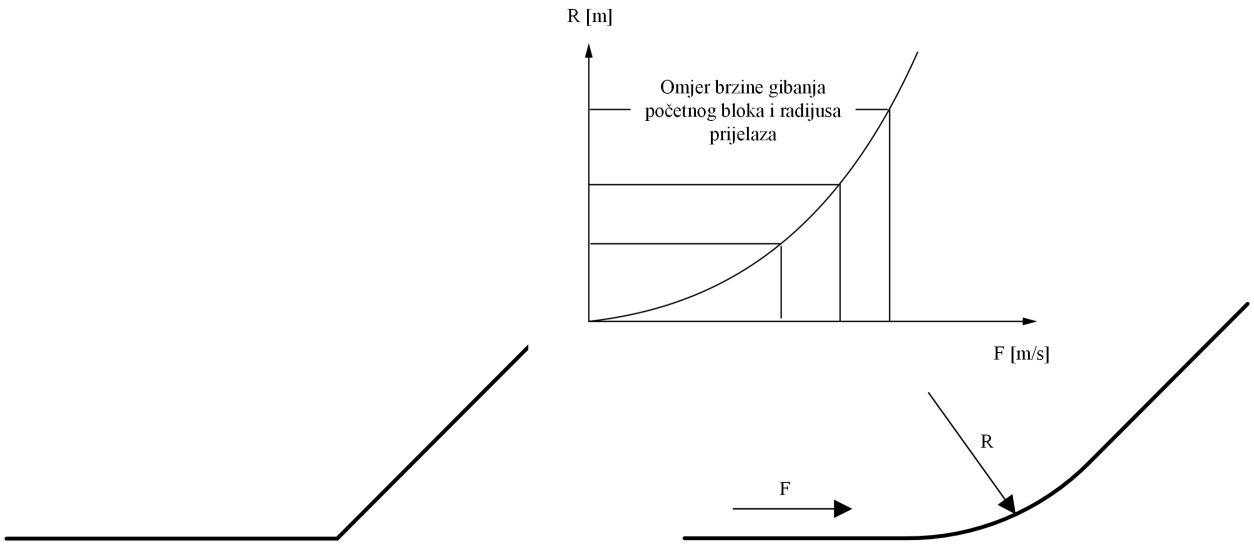
sliku potrebno promatrati sa desna na lijevo. Ovo je primjer samo sa normalnim blokovima, ista situacija može biti i sa kratkim blokovima ili u kombinaciji normalnih i kratkih blokova. Velike brzine obrade kod malih pomaka, kao što je već spomenuto, precizne obrade slobodno oblikovanih površina, kako bi se poboljšala kvaliteta obrađene površine, potrebna je veća segmentacija slobodno oblikovanih krivulja koje upravljačka jedinica mora aproksimirati kao putanje. Veća segmentacija znači podjelu na više manjih linearnih i kružnih gibanja, što zahtjeva iznimno puno blokova programskog G-koda, kratkih udaljenosti. Svaki blok G-koda se može poistovjetiti sa definiranim profilom brzine, što u navedenom slučaju znači veliki broj takvih blokova, gdje se već podrazumijeva način rada u kontinuiranom gibanju, bez prekida. Kako bi upravljačka jedinica optimizirala brzinu gibanja između blokova, brzina obrade i aproksimacije segmentiranih linija je ključna u ovom slučaju. Potrebno je izvesti gledanje unaprijed, odnosno, upravljačka jedinica ne samo da analizira slijedeći blok G-koda, i proračunava parametre profila, nego proračunava stotine, ili tisuće blokova, odnosno profila gibanja unaprijed, te parametre pohranjuje u internu memoriju. Cilj funkcije gledanja unaprijed je optimizacija brzine gibanja, s obzirom da se zadana brzina teško postiže zbog stalnih promjena smjera gibanja, pa se tako optimizacijom želi postići što je moguće veća dopustiva brzina gibanja. Ova funkcija nema smisla u načinu rada zaustavljanja nakon svakog bloka. Jer, nakon što se gibanje završi, vremenski period potreban za proračun sljedećeg koraka je „neograničen“. Dovoljno je vremena za proračun parametara gibanja za sljedeći profil brzine gibanja.



Slika 6.7 Usporedba putanje i brzine gibanja između točnog zaustavljanja i kontinuiranog načina gibanja

Slikom je prikazana usporedba gibanja točnog zaustavljanja i kontinuiranog načina gibanja, gdje je vidljivo da je korištenjem kontinuiranog načina gibanja, koje je ostvarivo primjenom funkcije gledanja unaprijed, postiziva puno veća brzina. Kod profila brzine koji je generiran u načinu rada

točnog zaustavljanja, profil izgleda „pilasto“. Sastoji se od više kratkih profila, gdje zbog velike brzine i malih pomaka niti u jednom bloku alat ne može postići zadanu brzinu gibanja. Ovakav „pilasti“ profil izaziva vibracije i mehaničke udare tijekom gibanja što može utjecati na kvalitetu obrade. Jedan nedostatak kontinuiranog gibanja, funkcije gledanja unaprijed, postizanje što veće brzine gibanja, je gubitak preciznosti. Iako u slučajevima obrade slobodno oblikovanih površina to nije nedostatak, nego više kompromis ili prednost. Prednost, iz razloga što niti nije cilj dobivena površina koja se sastoji od puno manjih linearnih pomaka, nego glatka površina bez prijelaza između promjene smjera gibanja linearnim vođenjem alata. Preciznost je obrnuto proporcionalna brzini gibanja i promjenama te brzine, da bi se postigli glatki prijelazi putanje gibanja, a time i promjene, mora se odrediti brzina u kutu promjene smjera gibanja i brzine. Takav prijelaz između dvije linearne putanje gibanja izveden je zaobljenjem. Jedna metoda sastoji se od analize kuta između dva linearna gibanja, te aproksimacija prijelaza kružnim gibanjem, gdje je za kružno gibanje u kutu potrebno proračunati ubrzanje i brzinu gibanja. Maksimalna brzinu u kutu ne smije prelaziti dopušteno ubrzanje. Odnosno, utjecaj na brzinu u kutu ima brzina trenutnog i sljedećeg definiranog posmaka, kut između dva linearna gibanja i ubrzanje koje ovisi o promjenu brzine gibanja trenutnog i slijedećeg bloka. Drugi načina određivanja brzine gibanja u kutu je pomoću razlike u brzini gibanja svake od osi. Ovdje se misli na osi koordinatnog sustava, ne na osi samog stroja. Brzina gibanja u kutu određuje se koeficijentom koji ovisi o razlici brzina gibanja svake od osi i maksimalne dopuštene brzine gibanja te osi. Koeficijent mijenja krajnju brzinu u trenutnom bloku i početnu brzinu u slijedećem bloku. Na fizičku putanja koju ostavlja alat, korištenjem raznih aproksimacija, odnosno zaglađenja, nema konačni utjecaj sam interpolator i definiranje ubrzanja i usporenja. Generiranje profila, kao slijedeći korak ima regulator pozicije i brzine, tako da utjecaj na konačnu putanju i zaglađenost putanje ima i regulator.



Slika 6.8 Radijus prijelaza između dva linearna gibanja

6.5. Algoritam interpolatora

Interpolacija putanje gibanja se sastoji od proračuna svih profila brzina svake komponente vektora brzine. Prije generiranja profila brzine, odnosno, parametara kojima se proračunava profil brzine, potrebno je izračunati parametre koje koristi mikrokontroler ili simulacijski program za proračun trenutne pozicije, brzine i ubrzanja u svakom trenutku.

6.5.1. ADCBI metoda interpolacije

Vrijednosti adresa dobivenih interpreterom G-koda potrebno je pripremiti za interpolaciju putanje. Dalje će biti objašnjen dio algoritma koji računa pripremne parametre za interpolaciju. Iako se metoda naziva određivanje ubrzanja i usporenja prije interpolacije brzine, najprije je potrebno odrediti komponente brzine, te na temelju toga dodijeliti ubrzanja i usporenja. Prije proračuna parametara za interpolaciju putanje, potrebno je odrediti smjerove gibanja i brzine sve 3 komponente vektora brzine. Ovaj dio se radi samo za linearna gibanja. Komponente brzine ovise od aktivnosti svake od osi, te promijeni pozicije svake od osi. Prvo je potrebno odrediti trenutnu promjenu pozicije svake od osi, koji put će svaka os proći u trenutnom bloku. Promjena pozicije je trenutna pozicija bloka i prošla vrijednost pozicije. Prvo se analiziraju linearna gibanja, na kraju kružna.

```
diff_posX = first_gLine.X - incPos.X;
diff_posY = first_gLine.Y - incPos.Y;
diff_posZ = first_gLine.Z - incPos.Z;
```

Vektor brzine je najjednostavniji ako je promjena pozicije samo po jednoj osi. Onda je brzina posmaka iz bloka jednaka vektoru brzine samo te osi. Tako da u ovom dijelu algoritam prvo razvrsta sve moguće slučajeve promjene pozicije za sve 3 osi, zatim na temelju toga računa komponente vektora brzine.

```
else if(diff_posX != 0.0f && diff_posY != 0.0f && diff_posZ == 0.0f)
{ feedZ = 0;
feedX = cos(atan( abs(diff_posY/diff_posX) )) * first_gLine.F;
feedY = sin(atan( abs(diff_posY/diff_posX) )) * first_gLine.F; }
```

Slučaj kada su vrijednosti po X i Y osi različite od 0, a os Z je jednaka 0. Gibanje se izvodi samo u ravnini X-Y. Posmak iz trenutnog bloka potrebno je rastaviti na komponente po X i Y osi. Matematički prikaz samo jedne komponente:

$$v_x = co\,s\left(arcco\,s\left(\frac{x_1}{\sqrt{x_1^2 + y_1^2}}\right)\right) \cdot f \tag{15}$$

$$v_y = \sin\left(arcsin\left(\frac{y_1}{\sqrt{x_1^2 + y_1^2}}\right)\right) \cdot f \tag{16}$$

Gdje je :

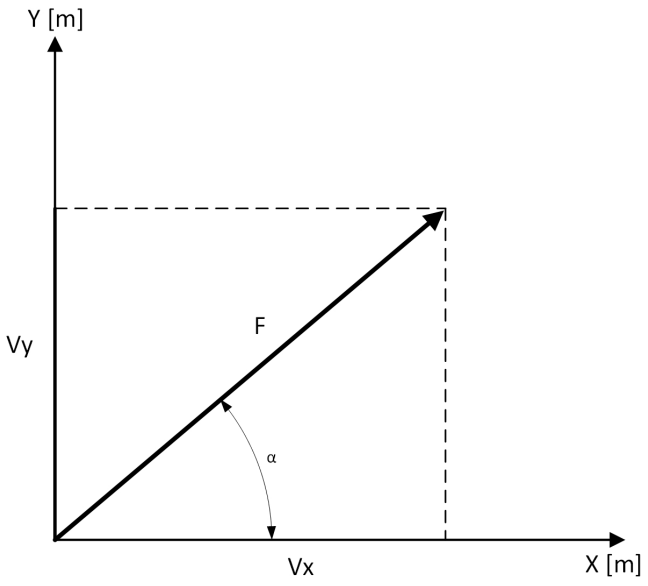
vx = komponenta po X osi

vy = komponenta po Y osi

f je zadani posmak

x1 i y1 su promjene pozicija po svakoj od osi

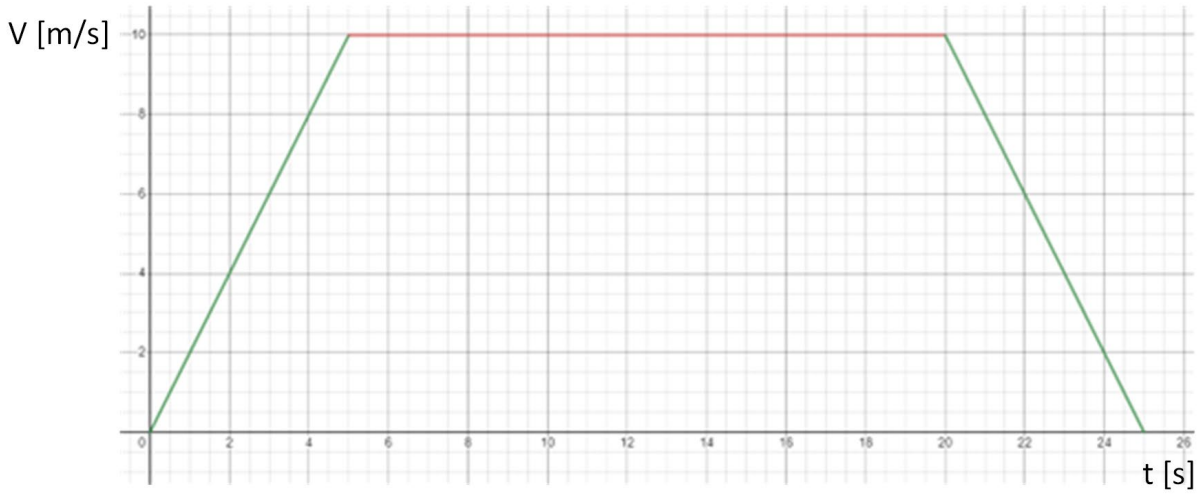
Brzine su dobivene iz omjera prijeđenog puta svake od osi. Komponente puta po X i Y osi, moraju razmjerno odgovarati komponentama puta.



Slika 6.9 Vektor brzine sa komponentama za X i Y os

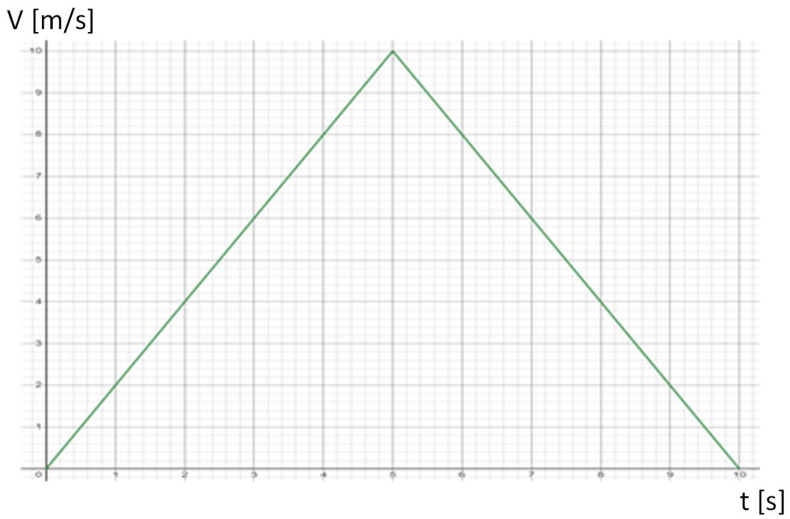
Nakon utvrđenih komponenti brzina i pozicija, algoritam utvrđuje vrstu gibanja prema kojoj izvodi interpolaciju u vremenu. Odnosno, provjerava koja vrsta profila će se pojaviti za pojedine osi - može se pojaviti samo dvije vrste profila **trapezni** i **trokutni**, normalni profil i kratki profil.

Slučaj kada X os mora proći najveću udaljenost, te vrsta profila koja se može pojaviti: Odluka o vrsti profila (trapezni ili trokutni) donosi se na temelju površine ispod krivulje trapeznog profila.



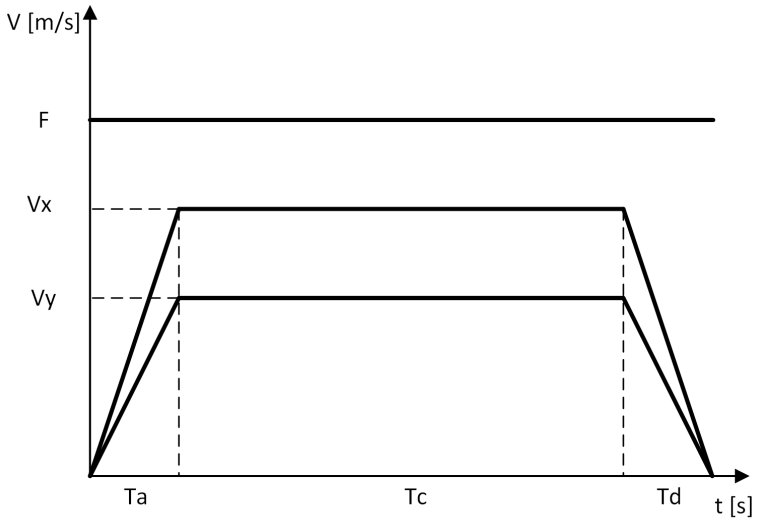
Slika 6.10. Normalni trapezni profil brzine

Algoritam provjerava površinu ispod krivulje i uspoređuje sa promjenom pozicije po dominantnoj osi. Površina ispod krivulje je jednaka prijeđenom putu. Ako su površine puta ispod dijela trapeza ubrzanja i usporenja manje od puta kojeg os mora proći za interpolaciju se odabire trapezni profil. Ako je površina manja - slučaj kad je, ili mala udaljenost, ili prevelika brzina kretanja. U tom slučaju interpolacijom se neće moći dostići zadana brzina, te se u tom slučaju odabire trokutni profil kod kojeg je potrebno reducirati ubrzanja i usporenja. Prema reduciram ubrzanjima i usporenjima će se računati maksimalna brzina koja je ostvariva.



Slika 6.11 Kratki trokutasti profil brzine

Za dvije mogućnosti profila mora se ispitati koja os je najdominantnija u izvođenju kretanja. Koja os mora proći najveću udaljenost - ako mora proći najveću udaljenost njena komponenta brzine mora biti najveća. Ako dominantna os uspije izvesti gibanje, odnosno, ona os koja mora prevaliti najveću udaljenost, mora se kretati najvećom brzinom. Ako takva os uspije izvesti gibanje trapeznim profilom sve ostale osi koje se moraju kretati manjom brzinom moraju biti u stanju kretati se istim profilom. Tako da se u ovom dijelu analizira samo jedna od tri osi, dok ostale osi moraju pratiti takvo gibanje. Gibanja sve tri osi moraju biti sinkronizirana, tako da segmenti profila ubrzanja i usporenja moraju imati isto vremensko trajanje.



Slika 6.12 Dva normalna profila kod simultanog gibanja

Ova dva parametra ispituju se zajedno te se određuje vrsta gibanja koja se pohranjuje u varijabli *mode_move*. Tri načina gibanja, ako se radi o trapeznom profilu, i tri načina gibanja ako se radi o trokutnom kratkom profilu. Za svaki profil po tri načina gibanja ovisno o osi.

```
if ( (abs(diff_posX) >= abs(diff_posY) && abs(diff_posX) >=
abs(diff_posZ) ) &&
((pow(feedX, 2)/(2*acc) + pow(feedX, 2)/(2*acc)) < abs(diff_posX)) )
{ mode_move = 1; }
```

Kod kružnih gibanja, sličnu stvar je potrebno provesti u ovom djelu programa, samo ima više parametra koji opisuju kružna gibanja. Proračun je vezan na parametre kojima se opisuje geometrija kružnice. Prvo je potrebno izračunati parametre svojstvene za kružna gibanja radijus rotacije, kutnu brzinu i kutno ubrzanje.

```
radius = sqrt(pow(first_gLine.I, 2) + pow(first_gLine.J, 2));
W = first_gLine.F / radius;
Aw = acc / radius;
```

Zatim je potrebno iz tih parametara izračunati početni i krajnji kut na kružnom dijelu koji se želi opisati. Kao i prevaljeni kut. Sa tim parametrima može se pristupiti analizi na način da li će se koristiti trapezni ili trokutni profil, samo, u slučaju proračuna kružnih gibanja, u analizi se ne koriste prevaljeni dužinski put i brzina, nego prevaljeni kut i kutna brzina. Prikazani proračun analizira samo kružna gibanja u X-Y ravnini.

```
if( (pow(W, 2)/(2*Aw) + pow(W, 2)/(2*Aw)) < angle && first_gLine.G ==
2)
{ mode_move = 7; }
```

Odluka o vrsti profila se također temelji na površini ispod trapeza i, da li je površina veća ili manja od puta koji je potrebno prevaliti. Nakon definiranja vrste gibanja potrebno je izračunati parametre koji se koriste u vremenskoj petlji interpolatora. Svrha pripreme parametara za vremensku petlju interpolatora je svesti računanje unutar interpolatora na što manju razinu, razlog tome je što se vremenska petlja u stvarnim sustavim poziva u periodu i do 1000 puta u sekundi. Kako bi se osiguralo dovoljno vremena da procesorska jedinica izračuna sve potrebne vrijednosti unutra vremenske petlje, proračun unutar petlje treba svesti na što je moguće jednostavniju razinu. Odnosno, više obrađenih podataka pripremiti prije nego se pozove vremenska petlja. Prema vrsti gibnja koje je zapisano u varijabli *mode_move* računaju se vremenski parametri, i, parametri

ubrzanja i usporenja za linearna gibanja. Svi parametri se računaju za dominantu os koja mora sa najvećom brzinom prevaliti najmanji put.

```
case 1: // X trapeze
           // times calculating
           Ta = abs( feedX / acc );
           Td = abs( feedX / acc );
           Tc   =   (   abs(diff_posX)   -   (pow(feedX,2)/(2*acc))   -
(pow(feedX,2)/(2*acc)) ) / feedX ;

           J1 = (int)( (Ta/Titr) + 0.5f );
           J2 = (int)( (Tc/Titr) + 0.5f );
           J3 = (int)( (Td/Titr) + 0.5f );

           a1_X = dirX * feedX / (J1*Titr);
           a3_X = dirX * -feedX / (J3*Titr);
           a1_Y = dirY * feedY / (J1*Titr);
           a3_Y = dirY * -feedY / (J3*Titr);
           a1_Z = dirZ * feedZ / (J1*Titr);
           a3_Z = dirZ * -feedZ / (J3*Titr);
       break;
```

Primjer proračuna, kada je X os dominantna u trapeznom obliku profila: Parametri koji se proračunavaju su vremenske konstante ubrzanja, usporenja i trajanja konstantne brzine. Vremenske konstante opisuju trajanje pojedinih segmenata trapeznog profila preko kojih se računa broj vremenskih iteracija za svaki segment profila. Vremena opisana varijablom *T* označavaju vrijeme u sekundama, a vrijeme izraženo varijablom *J* predstavlja broj vremenskih iteracija trajanja dotičnog segmenta. Ovu varijablu nije moguće izraziti mjernom jedinicom. Ubrzanja *a1*, odnosno usporenja a3 opisuju samo nagib krivulje trapeznog profila ili nagibe u trokutnom profilu.

6.5.2. Vremenska petlja interpolatora

Vremenska petlja poziva se u određenim vremenskim intervalima. Na temelju prethodno izračunatih parametara, generira trenutnu brzinu profila, grafički objašnjeno, iscrtava trapezni profil, ili, trokutasti profil. Uz brzinu, može se vrlo jednostavno doći do trenutne pozicije u svakom trenutku. Vremensku petlju poziva tajmer, koji je u ovom slučaju podešen na prekidnu rutinu od 1000 puta u sekundi. Svake sekunde proračunava se trenutna brzina i pozicija svake od osi. Unutar petlje prvo se provjerava koja vrsta gibanja se korisiti.

```
if(mode_move == 1 || mode_move == 2 || mode_move == 3)
```

```
    {
        if(stop == true)
        {
            curVel.X = curVel.X + (a3_X * Titr);
            curVel.Y = curVel.Y + (a3_Y * Titr);
            curVel.Z = curVel.Z + (a3_Z * Titr);

            incPos.X = incPos.X + (curVel.X * Titr);
            incPos.Y = incPos.Y + (curVel.Y * Titr);
            incPos.Z = incPos.Z + (curVel.Z * Titr);
        }
        else if(itr <= J1 && stop == false)
        {
            curVel.X = curVel.X + (a1_X * Titr);
            curVel.Y = curVel.Y + (a1_Y * Titr);
            curVel.Z = curVel.Z + (a1_Z * Titr);

            incPos.X = incPos.X + (curVel.X * Titr);
            incPos.Y = incPos.Y + (curVel.Y * Titr);
            incPos.Z = incPos.Z + (curVel.Z * Titr);
        }
        else if(itr <= J1+J2 && stop == false)
        {
            incPos.X = incPos.X + (curVel.X * Titr);
            incPos.Y = incPos.Y + (curVel.Y * Titr);
            incPos.Z = incPos.Z + (curVel.Z * Titr);
        }
        else if(itr <= J1+J2+J3 && stop == false)
        {
            curVel.X = curVel.X + (a3_X * Titr);
            curVel.Y = curVel.Y + (a3_Y * Titr);
            curVel.Z = curVel.Z + (a3_Z * Titr);

            incPos.X = incPos.X + (curVel.X * Titr);
            incPos.Y = incPos.Y + (curVel.Y * Titr);
            incPos.Z = incPos.Z + (curVel.Z * Titr);
        }

        rouVel.X = Rou(curVel.X);
        rouVel.Y = Rou(curVel.Y);
        rouVel.Z = Rou(curVel.Z);

        if (stop == true && rouVel.X == 0.0 && rouVel.Y == 0.0 &&
rouVel.Z == 0.0){ runTimer = false; mode_move = 0; }
        else if (itr == J1+J2+J3){ runTimer = false; mode_move = 0;
pos_realize = true; }
        itr ++;
    }
```

Primjerom je prikazano linearno gibanje trapeznog profila. Svaka *if* funkcija zadužena je za jedan segment trapeznog profila. Prva *if* funkcija služi samo za zaustavljanje. Ostale funkcije izvode se ovisno o trenutnom vremenskom intervalu *int* koji se svakom pozivom povećava za jedan, te se trenutni broj iteracije uspoređuje sa potrebnim brojem iteracija za svaki od segmenata profila J1, J2 i J3.

7. Petlje povratne veze

Izlazne vrijednosti iz interpolatora ne upravljaju direktno sa servo sustavom. Servo sustav upravlja motorom, pogonima svake od osi, prema upravljačkim signalima iz interpolatora. Vrijednosti pozicije i brizne osi nisu iste kao upravljački signali. Iz tog razloga uvodi se povratna veza kao povratna informacija o poziciji i brzini osi. Petlja povratne veze mora uspoređivati ove dvije veličine kako bi se otklonila razlika između zadane pozicije i stvarne pozicije osi. Ili brzine. Usporedba pozicije i brzine izvedena je u dvije kaskadne petlje, a može biti i treća kaskadna petlja. Točnost pozicioniranja je vrlo važna kod CNC strojeva jer direktno utječe na dimenzije izrađenog komada. Ukupna greška pozicije je linearna udaljenost referentne pozicije i stvarne pozicije.

Vođenje CNC stroja može se podijeliti na 3 strategije:

1. točka po točka,
2. kontrola praćenja, i,
3. kontrola konture.

Najjednostavnija strategija upravljanja je točka po točka, gdje upravljačka jedinica zanemaruje grešku putanje alata, a najvažniji parametar je vrijeme i dolazak u odredišnu točku. Kod kontrole praćenja potrebno je smanjiti grešku putanje alata od referentne putanje. Kod kontrole konture potrebno je smanjenje greške konture, smanjenje greške od željenog oblika uz glatke prijelaze kako bi se ostvario kompleksni izgled površine. Površine kod kontrole konture su najčešće slobodnog oblika. Povratne veze mogu biti izvedene na nekoliko načina. Najjednostavnija vrsta petlje nema povratne informacije u kontrolnu petlju. Koristi se kod upravljanja koračnim motorima koji ne zahtijevaju upotrebu povratne veze korištenjem enkodera. Upravljanje, odnosno, u ovom slučaju se govori o slijeđenju pomoću generiranih impulsa, gdje broj impulsa određuje poziciju, a brzina generiranja impulsa brzinu kretanja osi. Polu-zatvorene petlje gdje se enkoder, kao uređaj za detekciju pozicije, postavlja na servo motor koji je povezan sa pogonskim vretenom osi CNC stroja. Preciznost pomaka osi najviše ovisi o preciznost izvedbe pogonskog vretena, kvalitete izrade navoja vretena. Servo sustav ili upravljačka jedinica ima ugrađen sustav koji kompenzira zračnost unutra pogonskog vretena. Takva zračnost se kod upravljanja naziva povratni zazor (engl. *Backlash*). Najveći utjecaj ima kod promjene smjera gibanja osi, pa tako upravljački sustav kod promjene smjera izvrši više gibanja od potrebnog, kako bi kompenzirao zračnost. Zatvorene petlje su vrlo slične polu-zatvorenim petljama, sa razlikom u tome što greška između zadane pozicije i stvarne pozicije ulazi u kontrolni sustav. Izvor mjerne pozicije je linearni enkoder koji se nalazi na samoj osi, mjerena pozicija je puno točnija u odnosu na mjerenje enkoderom. Povratni zazor

nije moguće detektirati samo sa enkoderom koji se nalazi na servo motoru, a zazor se manifestira između pomičnog dijela osi matice i vretena. Greška i kod ove vrste petlje ovisi o zračnosti pogonskog vretena. U kontrolnom sustavu se koristi pojačanje praćenja pozicije. Povećanjem pojačanja raste preciznost i odziv sustava, ali preveliko povećanje može dovesti do nestabilnosti sustava. Hibridna kontrolna petlja koristi obje vrste mjerenja. I enkoderom, koji se nalazi na motoru, odnosno na pogonskom vretenu, i, linearnom enkoderu koji se nalazi na samoj pomoćnoj osi. Koriste se dvije vrste petlji. Kao što je već navedeno povećanje pojačanja može uzrokovati nestabilan sustav, ali smanjenje može uzrokovati nepreciznost. Stoga je moguće veliko pojačanje izvesti petljom koja mjeri poziciju preko servo motora, jer u tom slučaju dio stroja, koji se upravlja, nije dio upravljačkog sustava. Pozicija iz linearnog enkodera, koji je dio stroja, se koristi samo za kompenzaciju greške, te pokazuje dobre rezultate i sa manjim pojačanjem.

7.1. Vrste petlja

Svrha petlje je eliminacija ili smanjenje pogreške pozicije i brzine između stvarnih vrijednosti i vrijednosti iz generiranog profila. Najčešća izvedba upravljačko kontrolne petlje sastoji se od tri petlje, koje se izvodi istim brzinama. **Vanjska petlja** je petlja za kontrolu pozicije, **srednja petlja** za kontrolu brzine i **unutarnja petlja** kontrolira struju motora. Struja motora je proporcionalna momentu, odnosno opterećenju motora. Petlja **kontrole momente** i/ili petlja **kontrole brzine** u nekim slučajevima je unutar servo upravljača, a ne unutar uređaja na kojem se nalazi jezgra numeričkog upravljanja. Moderni servo sustavi, koji kao jezgru numeričkog upravljanja koriste isključivo PLC, kao upravljačku jedinicu za upravljačku petlju, koriste samo kontrolu pozicije. Dok, brzinom i momentom upravlja servo upravljač, koji može izvršavati samo kontrolu momenta. Ovisi o konfiguraciji cijelog servo sustava.

Kod vrsta upravljanja postoji nekoliko upravljačkih algoritama koji se danas koriste:

- **P kontroler** – proporcionalni kontroler koji grešku sustava povećava za konstantnu vrijednost koja se kao ispravljačka vrijednost vraća u upravljačku petlju. Jako jednostavan za implementaciju i dovođenje upravljanog signala u referentnu vrijednost. Mogu se javiti problemi sa stabilnošću sustava iz tog razloga se koristi sa integralnim (I) i derivacijskim članom (D).
- **PID kontroler** – Koristi sva tri člana proporcionalno (P), integralni (I) i derivacijski (D). Biti će detaljnije objašnjen dalje u tekstu.
- Ostale vrste su npr. Fuzzy kontoler i feedforward kontoleri.

7.2. PID petlja

PID petlja ili PID kontroler ima široku primjenu u industriji zbog jednostavne arhitekture, i jednostavne implementacije, iako nema točnih matematičkih modela ponašanja sustava. Svakom iteracijom PID izračunava izlazni signal uspoređujući signale iz sustava sa referentnim signalima. Svakim utjecajem na sustav podešava izlazni signal kako bi se vrijednost sustava izjednačila sa referentnom vrijednosti. Nakon implementacije zahtjeva podešenja u odnosu na dinamiku sustava. Ako je PID kontroler podešen za jednu vrstu dinamike, ne može se koristiti ako se dinamički uvjeti unutar sustava promjene. PID kontroler može se koristi samo sa jednim ulazom i jednim izlazom, što je nepovoljno za CNC strojeve koji upravljaju sa više osi. Svaka os mora imati biti upravljana sa individualnim PID kontrolerom, te je, petlja PID regulatora za svaku os neovisna o ostalim osima.

Izlazna vrijednost iz PID regulatora:

$$u(t) = K_p e(t) + K_i \int e(t)\,dt + K_d \frac{de(t)}{dt} \qquad (17)$$

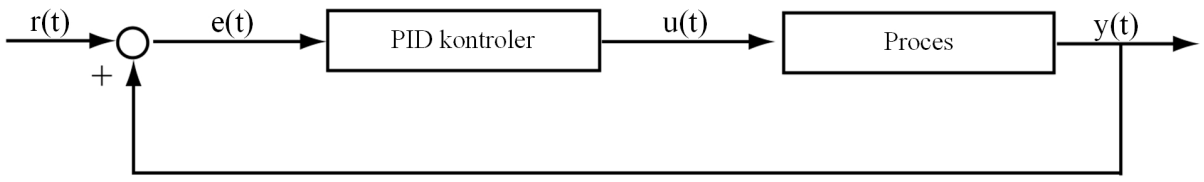
Gdje je:

u(t) – izlaz iz PID kontrolera

e(t) – pogreška sustava, razlika između referentne i stvarne izmjerene vrijednosti

r(t) – referentna vrijednost

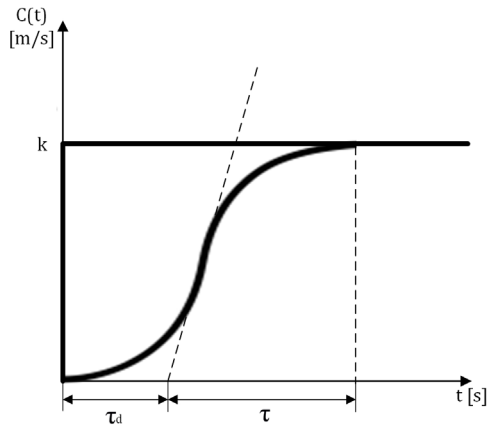
y(t) – mjerena vrijednost



Slika 7.1 Dijagram PID kontrolera

Pojačanje sustava, koje je spomenuto ranije u poglavlju vezano je za tri vrste pojačanja. Prvo pojačanje, **proporcionalno pojačanje**, koje je definirano koeficijentom proporcionalnog pojačanja K_p, koje trenutno smanjuje pogrešku sustava (razlika između referente i mjerene vrijednosti). Preveliko proporcionalno pojačanje može uzrokovati prebacivanje izlazne vrijednosti

u odnosu na referentnu, regulator preburno reagira na pogrešku, izazivajući još veću pogrešku suprotnog predznaka. U tom slučaju, sustav je podložan vibracijama i mehaničkim udarima. **Derivacijski član**, koeficijent pojačanja K_d smanjuje utjecaj prebacivanja vrijednosti i povećava vrijeme smirivanja signala. Derivacijski član eliminira pogrešku učenjem iz prošlosti. Ovaj član derivira trenutnu pogrešku, odnosno proračunava trenutnu promjenu pogreške u odnosu na prethodnu. Veliko pojačanje dovodi do brzog odziva sustava, ali može uzrokovati vibracije. I, na kraju, koeficijent pojačanja **integralnog člana** K_I eliminira pogrešku na temelju predviđanja budućnosti. Preveliki koeficijent pojačanja može također uzrokovati prebacivanje izlazne vrijednosti, ali povećava vrijeme dolaženja sustava u referentnu vrijednost. Pogreška je integrirana tijekom vremena, što znači umnožak pogreške i vremenskog intervala. Vrijednosti koeficijenata pojačanja moraju se odrediti. Određivanje se može provesti ručnim ugađanjem koeficijenata uzastopnim praćenjem ponašanja sustava. Takva metoda se sastoji od pokušaja i pogrešaka, što je sigurna metoda u simulacijskom okruženju, te, dobra metoda shvaćanja utjecaja pojedinih koeficijenata na ponašanje sustava. Postoje ostale metode ugađanja koeficijenata pojačanja, neke od njih zahtijevaju matematičke modele cijelog procesa, dok neke ne zahtijevaju. Provođenje ugađanja matematičkim modelima je zahtjevan proces gdje svaki dio kontrolne petlje mora biti definiran matematičkim modelom, što je u praksi teško provedivo. Danas sve više sustava ima mogućnost automatskog ugađanja koeficijenata pojačanja, posebice kada je PID kontroler dio servo upravljača. Jedna takva metoda, koja ne zahtjeva matematički model cijelog sustava je Ziegler-Nichols metoda, koja je dobivena eksperimentalnim putem, odnosno, koeficijenti pojačanja su dobiveni eksperimentalno. Metoda se može sastojati od dvije vrste pobude. Jedna vrsta je **odziv na impuls**, gdje izlazna vrijednost iz PID kontrolera izgleda kao S krivulja. Druga metoda, naziva **metoda osjetljivosti sustava**, funkcionira na način da se podešavanjem koeficijenata pojačanja prate oscilacije sustava.



Slika 7.2 Određivanje vremenskih parametara S-krivulje

Metoda analize S-krivulje na impulsnu stepenastu pobunu - nakon generiranog impulsa snima se S-krivulje, te se analiziraju neki parametri krivulje koji su prikazani prethodnom slikom. Na snimljenoj krivulji potrebno je ucrtati tangentu prijeloma S-krivulje, te, odrediti vrijeme kašnjenja τ_d i vrijeme konstante τ. Iz dvije vrijednosti moguće je proračunati sve koeficijente pojačanja slijedećim relacijama:

$$K_p = \frac{1.2\tau}{\tau_d}, K_i = 2\tau_d, K_d = 0.5\tau_d \tag{18}$$

7.3. Implementacija PID petlje

PID petlja je implementirana zajedno sa funkcijama koje komuniciraju sa simulacijskim programom Coppelia Robotics. PID petlja zahtjeva čitanje trenutne brzine iz simulacijskog programa. Vrijednosti koje se čitaju iz simulacijskog programa su trenutna pozicija i trenutna brzina. Algoritam uspoređuje pročitane vrijednosti sa referentnim vrijednostima iz profila brzina i pozicije generiranim interpolatorom. Razlika vrijednosti je pogreška signala koja je zapisana varijablom *error_X*, dok je u stvarnom sustavu potrebno čitati samo vrijednosti sa enkodera. Ako se upravljanje simulacijskog stroja vrši preko PID kontrolera, upravlja se brzinama osi. Dio programa za jednu od tri osi, izračunava P, I i D elemente PID kontrolera.

```
PX = P * error_X;
IX = IX + (I * error_X * dt);
DX =  (D * (error_X - last_error_X)) / dt;
```

Primjer dijela algoritma za proporcionalni, integracijski i derivacijski član PID regulatora. Zbroj pojedinih članova PID petlje je brzina kretanja pojedine osi.

```
velX = PX + IX + DX
```

Parametar *dt* vremenskog uzorkovanja mora biti jednak vremenskom intervalu *Titr*. Vremenski interval postavljen je na 0.001 sekundu, što znači da se i PID petlja izvršava 1000 puta u jednoj sekundi. Što je vremensko uzorkovanje manje pomak osi je preciznije vođen. Izlazna vrijednost iz PID kontrolera je brzina po svakoj od osi, PID kontroler upravlja sa brzinom sa svakom od osi na isti način kao kod stvarnog servo upravljača.

8. Izrada simulacije

8.1. Simulacija kod CNC obrada

Izrada G-koda je kompleksan proces. Uz nadzor preciznosti generiranja putanja, potrebno je provjeriti pogreške unutar programa. Pogreške je potrebno otkriti prije puštanja u rad na stroju, jer tada one dovode do skupih i opasnih situacija. Od zastoja u izvođenju programa zbog korištenja krive sintakse do loma dijelova stroja, radnog komada, alata i ostale opreme. Ponekad, bez korištenja verifikacije programa simulacijskim programima, jedina provjera i verifikacija programa je puštanje na stvarnom stroju. U tom slučaju obrada se izvodi na zamjenskom obradku koji je najčešće od mekših i jeftinijih materijala kao što su plastika, drvo, vosak, razni ostali jeftiniji polimeri. Nakon obrade vrši se provjera dimenzija i kvaliteta obrađene površine u omjeru koliko dopušta ovakva vrsta materijala te naknadna optimizacija programa. Svrha korištenja mekših materijala je barem djelomično osiguranje od udara alata u obradak, koja je najčešća pogreška u programiranju CNC strojeva, a može dovesti do loma alata, deformacije obradka pa čak i do mehaničkih oštećenja pogonskih aktuatora i prijenosnika gibanja CNC stroja. I u slučajevima uspješno provedene obrade, ovo je gubitak za proizvodnju, posebice kod testiranja obrade zahtjevnih površina slobodno oblikovanih, kao kod npr. kalupa. Obrada takvih dijelova je jako dugotrajna, i samim testiranjem povećava cijenu cijele proizvodnje, pa na kraju i proizvoda. Kako bi se mogućnost pogreške potpuno smanjila, ili svela na najmanju moguću mjeru, koriste se programski alati za verifikaciju G-koda. Uz cilj eliminacije pogreške, simulacije zamjenjuju klasično testiranje, što ubrzava sam proces proizvodnje uz smanjenje mogućnosti oštećenja. Alati za izradu i generiranje G-koda, CAM programski alati, imaju mogućnost izrade simulacije. Odnosno, nakon generiranog programa vrši se testiranje, simulacija i verifikacija G-koda. Uz provjeru sintakse, simulacija generira putanje alata koji vizualno prikazuju gibanje alata i mogućnost sudara sa ostalim objektima koji se nalaze na radnom stolu, kao na primjer, sa steznom napravom, uređajem za umjeravanje alata itd. Nakon izvršenog programa, simulacija pruža informacija o vremenskom trajanju izvođenja programa te vremenskom trajanju svakog od korištenih obrada koji su definirane u CAM programskom alatu.

Simulacijski programi imaju nekoliko razina simulacija koje se razlikuju prema kvaliteti simulacije i količini informacija koje se mogu generirati iz provedene simulacije. Prva razina simulacije je generiranje putanja alata gdje je radni predmet prikazan kao već obrađen, nema sirovca. Druga razina bi bila prikaz sirovca prije puštanja programa i prikaz odnošenja materijala prolaskom alata. Oba načina prikaza simulacije, veoma su korisna, pogotovo kod višeosnih obrada, recimo kod 5-osne obrade. Kod provedbe simulacije alat rotira oko radnog komada, gdje je radni

komad stacionaran, u prostoru. Posljednja razina simulacije je simulacija cijelog CNC stroja sa definiranim modelima stroja, posebno onih koji su u gibanju i mogu izazvati sudar i nesreće.

8.2. Svrha i važnost simulacije

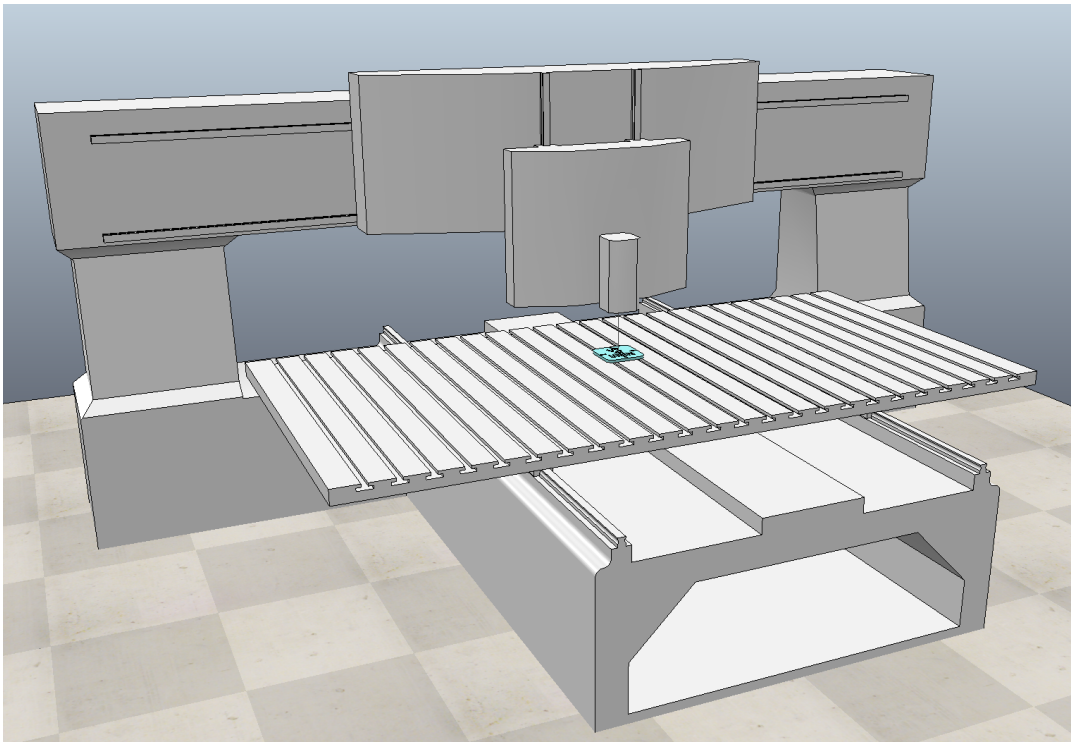
Svrha simulacije u smislu puštanja u rad nekog G-kod programa je već navedena. Ostala bitna stvar kod simulacija je edukacija, odnosno, to što nekvalificirane osobe mogu postići određenu razinu interakcija kao kod pravog stroja, bez mogućnosti oštećenja i izazivanja nesreća. Takav način učenja u sigurnom okruženju potiče na istraživanje mogućnosti nekog CNC sustava. Konačan cilj simulacije je izrada što vjernije kopije stvarne situacije industrijskog procesa proizvodnje, s obzirom da CNC stroj, unutar visokoautomatiziranog proizvodnog procesa ne djeluje sam, odnosno, CNC stroj je samo dio takve proizvodne linije, te ovisi o ostalim strojevima kao na primjer robotima, manipulatorima, dodavačima i ostalim CNC strojevima. Svi elementi proizvodnog sustava se moraju sinkronizirati, izvoditi svoje operacija kako bi se maksimalno optimizirala proizvodnja. Kod detaljne simulacije obrade, te ostalih operacija, uz obradu, kao na primjer dodavanje sirovca, te izuzimanje gotovog obratka, potrebna je detaljna izgradnja virtualnog modela proizvodnog sustava. Virtualni blizanac je izgradnja digitalne kopije stvarne industrijske okoline, kako bi se provelo programiranje, testiranje i puštanje u rad, neovisno o stvarnom stanju. Nije potrebno zaustavljanje proizvodnog toka uslijed reprogramiranja, nego se utjecaj promjene može ispitati virtualno. Virtualni blizanac nije samo trodimenzionalni prikaz pojedinih strojeva, nego ima važan učinak na kontrolu nad radom raznih upravljačkih uređaja kao što su PLC-i, upravljačke jedinice, kontroleri uz korištenje senzora itd.

U ovom radu svrha simulacije je prikaz putanja alata, radi testiranja algoritama upravljanja. Virtualno izrađeni stroj predstavlja stvarni stroj, a razina detalja u ovom slučaju, kao i kod većine simulacija, nije presudna. Ovakva tro-osna obrada nije toliko podložna greškama u vidu sudara kao što je to slučaj kod 5-osne obrade, ili obrade robotskom rukom. Ovdje je, u radu, uz provjeru rada algoritama virtualne upravljačke jedinice, napravljena provjera više načina upravljanja i njihov odraz na ponašanje rada stroja i generiranje putanje alata. Predstavljen je potencijal korištenja programskog alata za simuliranje robotskih okruženja (CNC stroj se može definirati kao vrsta robota), te izgradnja simulacije cijele proizvodnje. Napravljena je izgradnja digitalnog blizanca u programskom alatu koji je otvorene arhitekture.

8.3. Programski alat CoppeliaSim

CoppeliaSim programski alat, koji se također može pronaći i pod nazivom V-REP, programski je alat za simuliranje robotske okoline za potrebe razvoja, istraživanja, edukacije i testiranja u virtualnom okruženju. Područja robotike u kojima se koristi CoppeliaSim je industrijska robotika kao što su robotske ruke, CNC strojevi, Delta roboti itd. Također je podržana i mobilna robotika, gdje je moguće simulirati hodajuće mobilne robote, mobilne robote sa kotačima i sl. I, na kraju, zračne robote, poput dronova. Uz gotove modele robota koji se mogu koristiti kao gotovi, mogu se izgraditi i vlastite konfiguracije robota. Bitan segment kod simulacije gibanja je fizički pokretač koji simulira fizikalne zakone u simulacijom okruženju. Programsko okruženje nudi rad sa senzorima pomoću kojih jedan robotski sustav percipira svijet. Jedan bitan segment je mogućnost programiranja robotskih sustava, odnosno izrada skripti za upravljanje gibanjem. CoppeliaSim interno koristi programski jezik Lua, ali korištenjem programskog sučelja upravljanje i programskim alatom i samim objektima unutar aplikacija može se izvesti iz neke udaljene aplikacije programskim jezicima kao C, C++, Python, Matlab itd. Pomoću programskog sučelja moguće je, ne samo upravljati objektima, nego i prihvaćati informacije sa senzora, vizijskih senzora, pozicija i orijentacija objekata unutar simulacijskog okruženja.

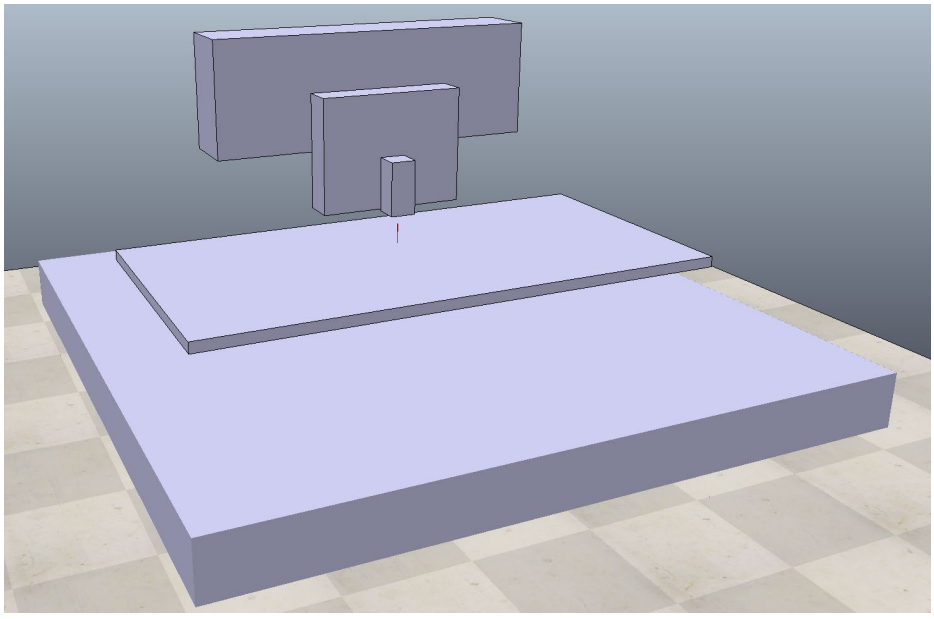
8.3.1. Projektiranje i izgradnja modela



Slika 8.1 Grafički prikaz modela CNC stroja

Kod prikaza modela CNC stroja vidljivi su 3D modeli koji samo pomažu vizualizaciji rada interpolatora CNC stroja. Prikaz modela CNC stroja služi isključivo korisniku za analizu gibanja kako bi ista mogao usporediti sa gibanjem stvarnog stroja. Dok je, s druge strane, za svrhu testiranja i istraživanja rada virtualne upravljačke jedinice, odnosno provjere algoritama bitniji dinamički prikaz modela. Programski alat Coppelia Sim može simulirati cijeli model CNC stroja, kao kinematičkog modela, gdje se analiziraju samo pomaci, brzine i ubrzanja/usporenja. U tom slučaju ovaj dinamički modela nema velikog utjecaja. Kinematički način rada simulacije poslužit će za testiranje generiranja putanje iz interpolatora, pomak samo po poziciji koja se prenosi na osi CNC stroja u simulaciji.

Dinamički model stroja dobiva puni potencijal u svrhu testiranja PID kontrolera, kada svaka os CNC stroja mora posjedovati određenu masu, inerciju, otpor gibanju i ostala dinamička svojstva krutog tijela. Iz toga razloga je dinamički model izgrađen od jednostavnih poligona kako bi fizičkom pokretaču za proračun dinamičkih vrijednosti bilo jednostavnije, odnosno, potrebna je puno manja procesorska moć.



Slika 8.2. Dinamički prikaz modela CNC stroja

U slučaju kinematike, sva tijela nemaju dinamična svojstva, pa svaki utjecaj na virtualne aktuatore uvijek će izvršiti proporcionalno gibanje. Kod načina rada u dinamičnom modelu, potrebna je kontrola, svaki pomak je posljedica neke sile i protusile. Na taj način zadavanje određene brzine gibanja određenim momentom neće uzrokovati trenutno gibanje osi.

8.4. Način komunikacije između aplikacije i simulacijskog programa

U svrhu testiranja rada upravljačke jedinice komunikacija između programa i simulacijskog programa je ostvarena samo slanjem trenutnih pozicija pojedinih osi koje su dobivene interpolacijom putanje u vremenskoj petlji. Iz tog razloga, uz računanje trenutnih brzina dodan je i proračun pozicija. Ovaj način je jednostavniji jer ne zahtjeva povratnu vezu iz Coppelia-e, simulacijski program samo prati pozicije pojedinih osi.

```
simxSetObjectPosition(clientID, X_handle, -1, &posX[0],
simx_opmode_oneshot);
simxSetObjectPosition(clientID, Y_handle, -1, &posY[0],
simx_opmode_oneshot);
simxSetObjectPosition(clientID, Z_handle, -1, &posZ[0],
simx_opmode_oneshot);
```

Prikazan je dio algoritma koji šalje pozicije pojedinih osi prema simulacijskom programu. Funkcije preko kojih se vrši komunikacija su dio jednog od više načina komunikacije. Način komuniciranja je *Legacy Remote* API koji nudi gotove funkcije za komunikaciju između simulacijskog programa i ostalih aplikacija. Ovaj način komunikacije u oba smjera je i najniže razine gdje se zahtjeva najviše poznavanje arhitekture razmjene informacija, ali je i najbrži način komuniciranja. Odnosno, samim pokretanjem CoppeliaSim simulatora pokreće se server na kojeg se aplikacija upravljačke jedinice spaja kao klijent i vrši komunikaciju. U ovom načinu rada aplikacije, upravljačka jedinica vrši slanje samo pozicija. Aplikacija preko komunikacije šalje pozicije svaki puta kada vremenska petlja interpolatora izračuna veličine koje opisuju putanju. To znači da aplikacija ne čeka odgovor od simulacijskog programa već samo šalje podatke. Svakako da ovaj način komunikacije ima mane u vidu toga što algoritam nema povratne informacije o stanju CNC stroju unutar simulacije.

Obostrana komunikacija povratnim informacijama iz simulacijskog programa koristi se u slučaju upravljanja CNC strojem PID regulacijom brzine. Aplikacija šalje brzinu kao izlaznu upravljačku vrijednost iz PID kontrolera. S obzirom da Coppelia ima mogućnost simulacije dinamičkih karakteristika kao mase, inercije, trenja, momenata gibanja modeli osi CNC stroja neće moći postići trenutnu brzinu. Sve te veličine su poremećajne veličine koje utječu na ponašanje sustava, koje PID kontroler mora kompenzirati, ovisno na pogrešku.

```
simxGetJointPosition(clientID, X_handle, &read_posX,
simx_opmode_buffer);
```

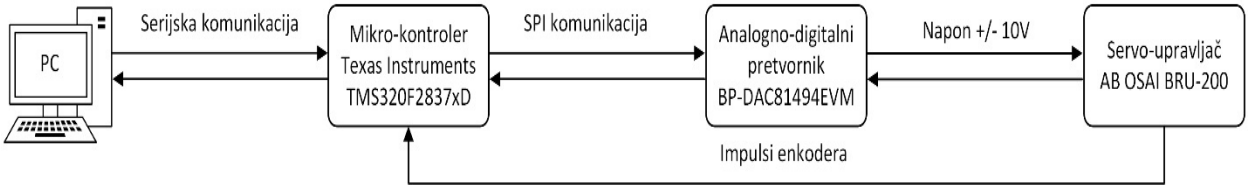
```
simxGetJointPosition(clientID, Y_handle, &read_posY,
simx_opmode_buffer);
simxGetJointPosition(clientID, Z_handle, &read_posZ,
simx_opmode_buffer);
```

Dio algoritma zaduženog za čitanje trenutnih brzina pojedine osi, uz čitanje pozicija može posebno vršiti čitanje brzina, ali se brzina može izračunati iz promjene pozicije. Obje vrijednosti ne služe samo za PID regulaciju, nego i kao podaci za provjeru rada sustava. Nakon čitanja trenutnih brzina slijedi PID regulacija koja je opisana u prethodnom poglavlju. Ishod PID regulacije je brzina pojedinih osi koja se šalje komunikacijom.

```
simxSetJointTargetVelocity(clientID, X_handle, velX,
simx_opmode_oneshot);
simxSetJointTargetVelocity(clientID, Y_handle, velY,
simx_opmode_oneshot);
simxSetJointTargetVelocity(clientID, Z_handle, velZ,
simx_opmode_oneshot);
```

9. Projektiranje procedure implementacije i testiranja razvijenog sustava na pravom CNC stroju

Prijenos algoritama iz simulacijske upravljačke jedinice na mikrokontroler koji upravlja servo pojačalima vrši se uz prijenos algoritama, uz minimalne promjene koje zadovoljavaju stabilno upravljanje. Algoritmi upravljanja simulacijom baziraju se na upravljanje brzinom, odnosno upravljanje različitim profilima brzina. Servo upravljači podržavaju samo upravljanje brzinama. Shema upravljanja dana je slijedećom slikom.



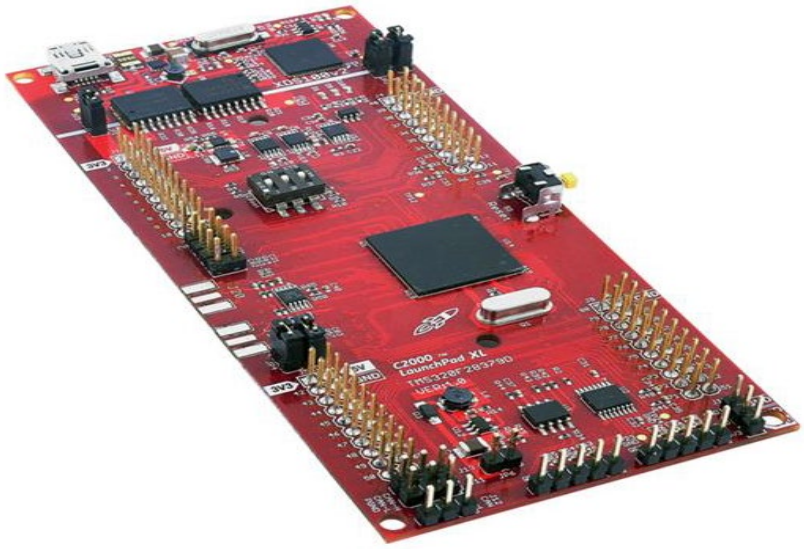
Slika 9.1 Shema upravljanja

Računalo služi kao sučelje operatora i CNC sustava. Preko GUI programa vrši se upravljanje cijelog sustava. GUI omogućuje manipulaciju G-kodom, te, pokretanje CNC stroja. Preko serijske komunikacije šalje pozicije svake osi, te, brzine kretanja na mikrokontroler. Zbog postizanja upravljanja u stvarnom vremenu, koje je u manjoj mjeri moguće izvesti na računalu, koristi se mikrokontroler. Mikrokontroler se može smatrati „mozgom“ cijelog sustava. Mikrokontroler računa sve parametre profila upravljanja. Izvršena su mjerenje kojima je potvrđeno da je mikrokontroler u stanju izračunati parametre unutar vremenske petlje, 1000 puta u sekundi. Kod računanja parametra i PID regulacije, unutar vremenske petlje, najbitnije je pozivanje vremenske prekidne rutine koja je pouzdani mehanizam vremenskog uzorkovanja. Unutar prekidne rutine računa se referentna brzina, ovisno o vremenskom intervalu. Trenutna brzina posmaka osi pretvara (derivira) se iz pozicije, zatim se uspoređuje sa referentnom brzinom profila, putem PID regulatora. Razlika referentnih vrijednosti, i stvarnih vrijednosti iz PID regulatora, SPI komunikacijom se šalje digitalno-analognom konverteru. S obzirom da sam mikrokontroler posjeduje nedovoljni broj i raspon naponskih analognih izlaza mora se prilagoditi vrstu informacijskog paketa koji koristi digitalno-analogni pretvornik. Pretvornik samo prosljeđuje informaciju o naponu prema servo upravljaču, i to u naponskim razinama +/- 10V. Odnosno, servo upravljač upravlja brzinom vrtnje motora u odnosu na referentni napon. Pri naponu 0V brzina vrtnje je 0, +10V maksimalna brzina vrtnje u jednom smjeru, dok je -10V maksimalna brzina vrtnje u drugom smjeru. Brzina vrtnje i pozicija registriraju se preko impulsa enkodera. Enkoder je spojen na servo upravljač, te sa servo upravljača je spojen na mikrokontroler.

9.1. Korištena oprema

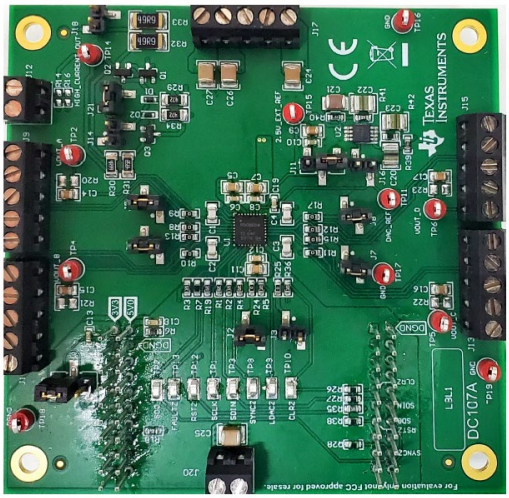
Slika 9.2 Korišteni CNC stroj Cosmec na kojeg će biti ugrađen razvijeni CNC kontroler

Stvarni stroj Cosmec, sa tri osi, i sa dvije obradne glave na koji se želi implementirati izrađena upravljačka jedinica, prikazan je na slici prije. O samom stroju zbog godine proizvodnje nema puno podataka. Dimenzije radnog stola je 1250 x 2500 x 450 mm. Masa stroja je otprilike 6 tona. Stroj se sastoji od dvije obradne glave. Svaka obradna glava ima svoju izmjenu alata. Servo motori AB Osai F-4030 kao pogoni osi mogu isporučiti 3.5 Nm momenta. Maksimalni broj okretaja je 4000 okr/min. Ovi servo motori se koriste za X i Y osi. Dok, Z os koristi servo motor istog proizvođača, AB Osai F-4050, momenta 6.9 Nm. Također, maksimalna brzina okretaja je 4000 okr/min. Oba motora koriste istu vrstu enkodera, napajanja 5V te 2000 impulsa za jedan okretaj. Uz servo motore nalaze se i servo upravljači istog proizvođača, modela BRU-200 za slabije servo motore DM-20 koji mogu isporučiti maksimalnu struju od 20A, te rade na struji od 10A. Drugi model, za jači servo motor je DM-30, te on može isporučiti maksimalnu struju od 30A, kontinuirana struja je 15A. Oba servo upravljača rade sa 325 V napona, te se upravljanje vrši ulaznim analognim naponom +/- 10V sa 13.3 Ohm impendancije.



Slika 9.3 Kontroler za upravljanje servo sustavom LAUNCHXL-F28379D.

Prikazani kontroler Texas Instruments LAUNCHXL-F28379D, odnosno razvojna pločica (kontroler je već na pločici sa izvedenim pinovima) radi na 200 MHz frekvencije sa 1MB flash memorije i 204 KB RAM memorije. Na 169 individualnih ulazno-izlaznih pinova koriste se slijedeći načini komunikacije: CAN, SPI, SCI/UART, I2C, McBSPs. Također, ima 12 16-bitnih analogno-digitalni pretvarača, 24 12-bitna analogno-digitalna pretvarača, i 8 12-bitnih digitalno-analognih pretvarača. Podržava 24 mogućnosti impulsno-širinske modulacije. Jedno također bitno svojstvo ovih kontrolera je mogućnost priključenja enkodera, te već predefinirani hardverski brojači koji mogu brojiti imulse enkodera. Mogu registrirati svaku promjenu generiranih imulsa u dva kanala, te time povećati rezoluciju čitanja stvarne pozicije.



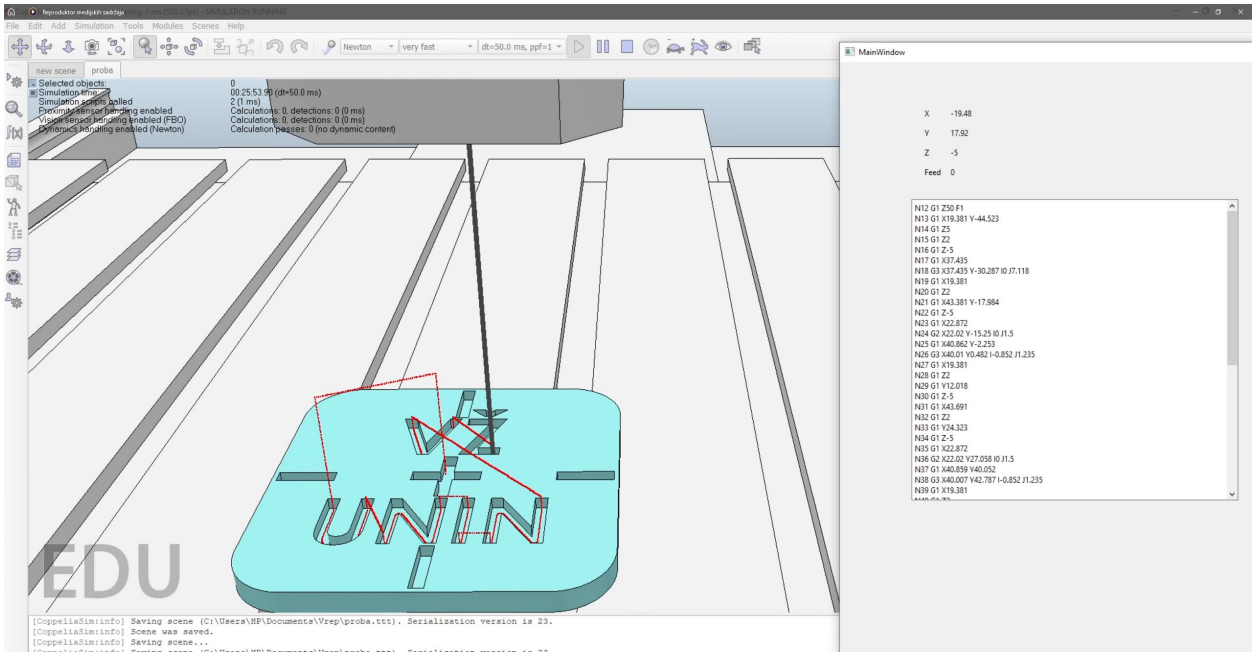
Slika 9.4 Digitalno analogni konverter Texas Instruments BP-DAC81404EVM

Digitalno analogni konverter Texas Instruments BP-DAC81404EVM, koji preko SPI komunikacije iz kontrolera prima podatak o postavljanju napona na izlazu. Digitalno-analogna razvojna pločica koja pretvara informacijski podatak u analogni naponski izlaz. Maksimalni napon koji može isporučiti je +/- 20V na 4 različita kanala, ili izlaza, sa rezolucijom od 16bit-a. Razvojna pločica na sebi već ima DAC integrirani krug sa unutarnjim referentnim naponom.

Slika 9.5 Elekrični ormar CNC stroja.

9.2. Karakteristike ponašanja različitih algoritama upravljanja u simulaciji

Provjera rada algoritama provodi se unutar simulacijskog paketa CoppeliaSim. Usporedba generiranja profila brzina izvedena je direktnim unosom brzina i putanja, bez korištenja PID regulacije, radi testiranja generiranja spomenutih parametara profila brzine.



Slika 9.6 Provjera rada algoritma upravljačke jedinice

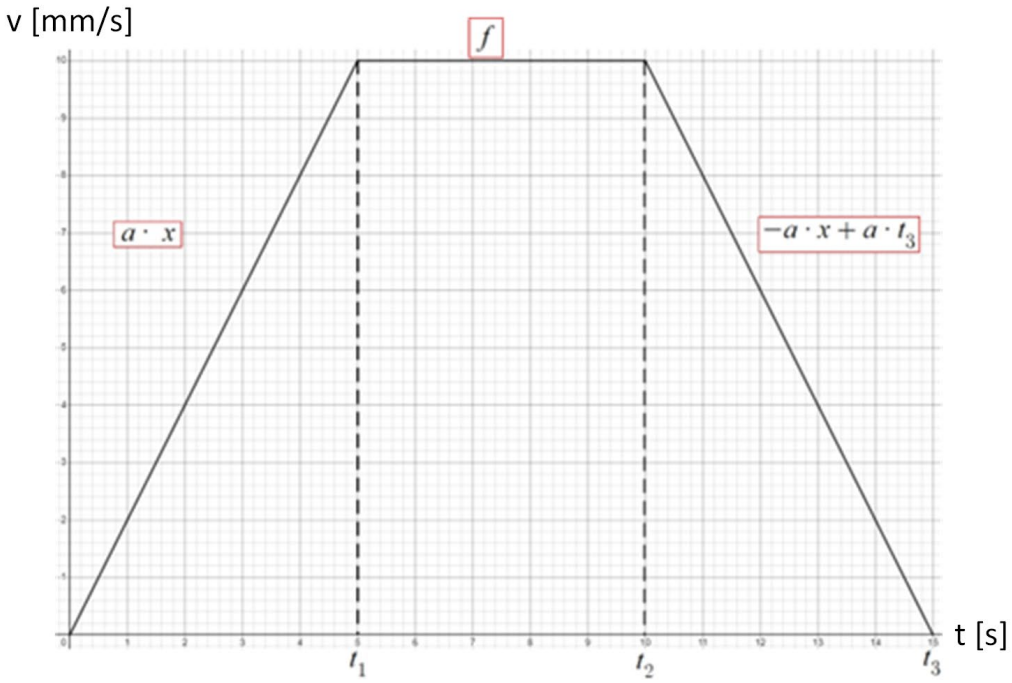
Kao što je pokazano gornjom slikom, simulacija glodanja loga, gdje se logo sastoji od linearnih i kružnih segmenta. Linearna kretanja trapeznog profila - radi lakšeg razumijevanja koristiti će se samo trapezni profil brzine, te usporedno generiranja profila puta koji je ujedno integral brzine. Prikazani profil generiran je sa slijedećim parametrima: akceleracija je usporena na 2 mm/s2 zbog bolje preglednosti profila. Posmak je 10 mm/s, kretanje je izvedeno samo sa jednom osi tako da brzina kretanja odgovara posmaku. Prevaljeni put je 100 mm. Parametri brzine su samo za potrebe analize profila, stvarno gibanje bi bilo presporo i nema svrhe kod provedbe simulacije cijelog programa. Slikom je prikazan idealni profil dobiven uvrštenjem matematičkih jednadžbi prikazanih tablicom.

Tablica 9-1. Parametri segmenata profila

a	Ubrzanje i usporenje
f	Posmak

Vremenske konstante za segmente profila	
t1	$t_1 = \frac{f}{a}$
t2	$t_2 = \frac{p}{f}$
t3	$t_3 = t_1 + t_2$
Pozicije za segmente profila	
p1	$\frac{f \cdot t_1}{2}$
p2	$\frac{f \cdot t_1}{2} + f \cdot (t_2 - t_1)$
p	Ukupni prevaljeni put

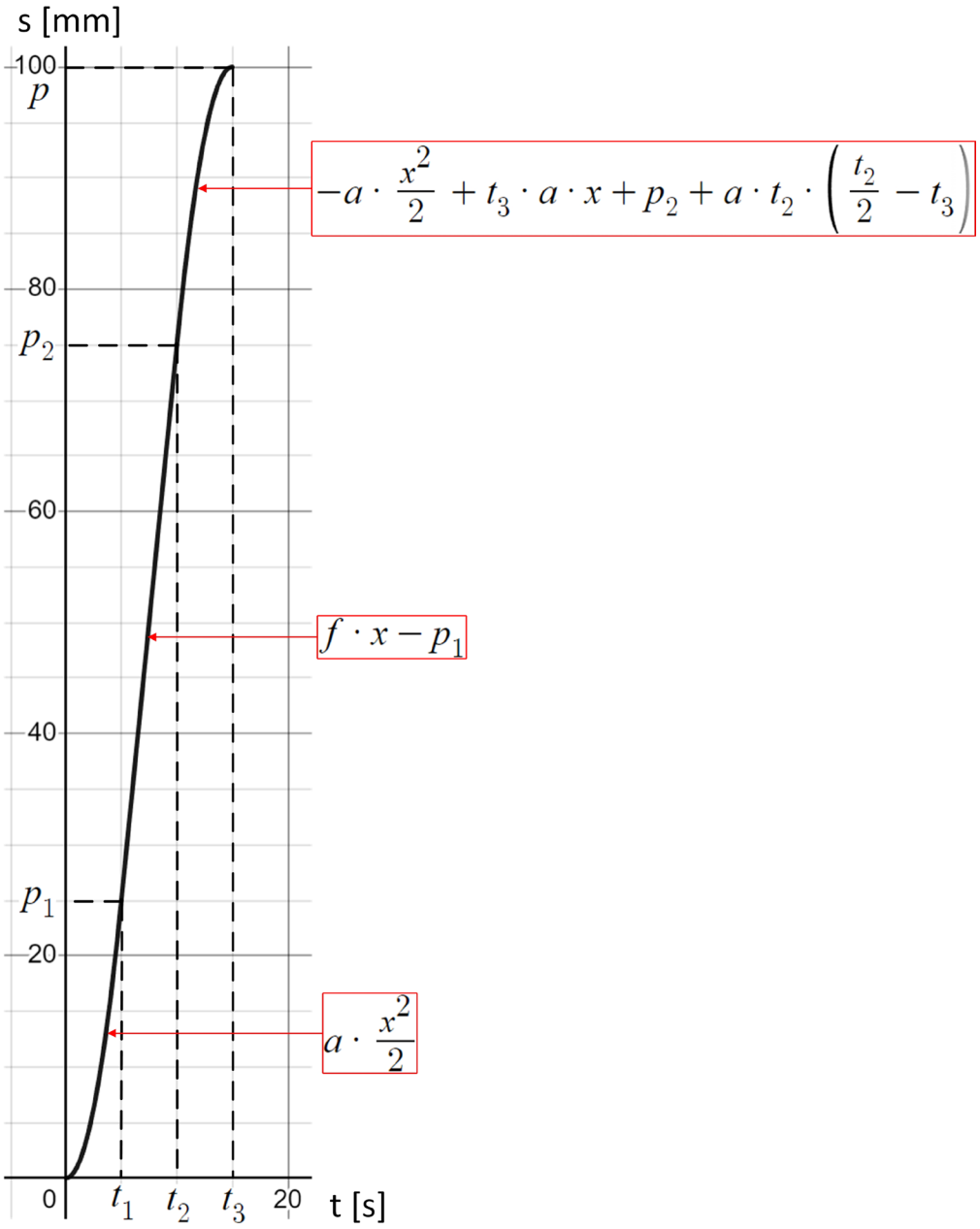
Izgradnja cijelog profila moguća je pomoću matematičkih relacija. Isto tako je moguća i implementacija u algoritam upravljačke jedinice, ali manja je brzina izvođenja u odnosu na već ranije objašnjene implementirane algoritme u upravljačku jedinicu.



Slika 9.7 Profil brzine za linearno gibanje

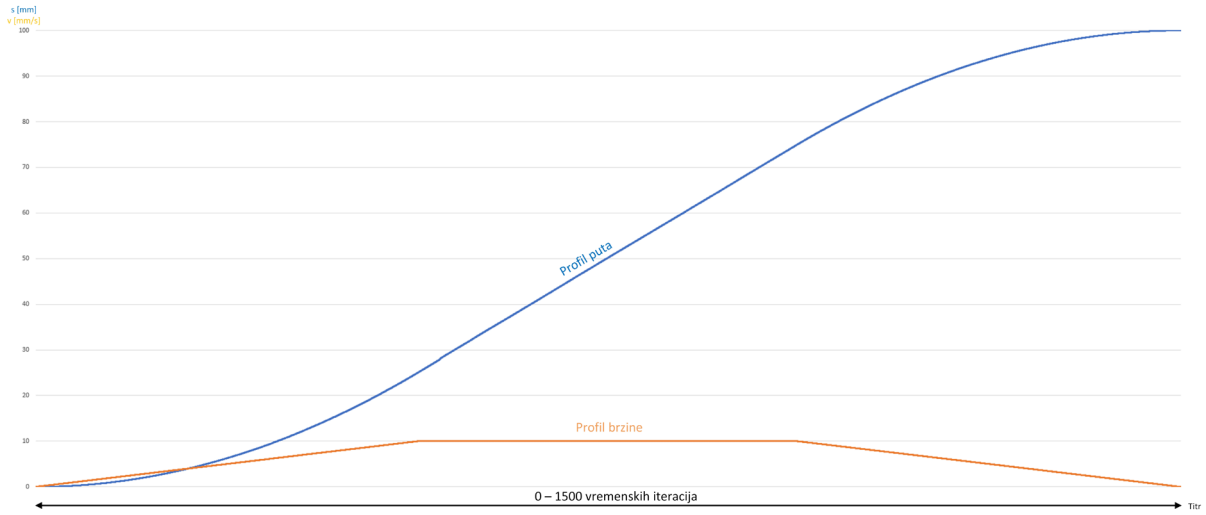
Efikasnost izvođenja upravljačkih algoritama je još istaknutija kod prijenosa na kontroler, koji ima puno manju procesorsku moć, u odnosu na simulaciju koja se izvodi na računalu. Matematičke

relacije su neophodne za potvrdu ispravnosti rada algoritama najviše dijela upravljačke jedinice interpolatora.



Slika 9.8 Profil puta za linearno gibanje

Profili brzine i puta dobiveni primjenom algoritama prikazani su slikom. Na apscisi je navedeno, u slučaju generiranja profila putem funkcija, vrijeme u sekundama. Kada se profili generiraju putem algoritama vremenska os je broj iteracija. S obzirom da je broj iteracija podešen na 1000 u sekundi, što znači da vrijednosti moraju odgovarati vrijednostima matematički generiranih profila samo pomnoženo sa 1000.



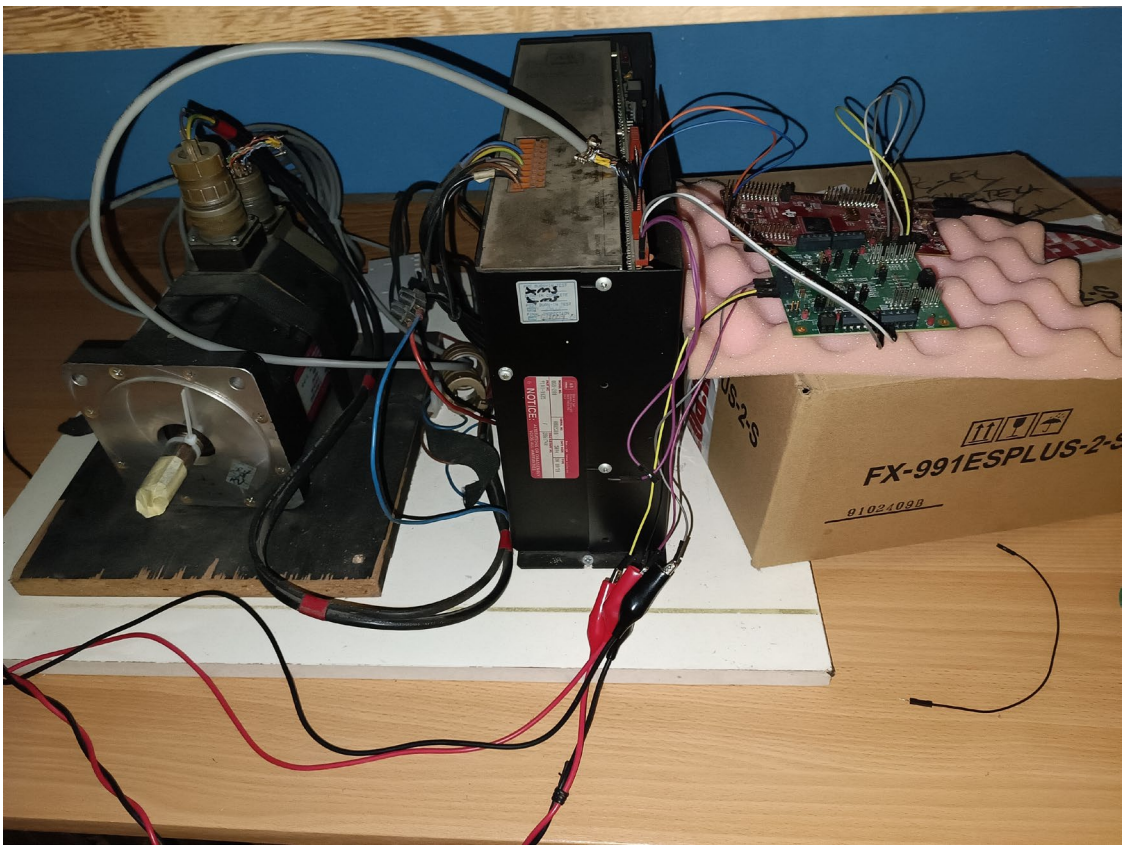
Slika 9.9 Generiranje profila putem algoritma

Za daljnja testiranja koja direktno uključuju opremu CNC stroja, potrebna je upotreba PID kontrolera. Vrijednosti brzine ili pomaka ne mogu se direktno prenesti u simulacijsko okruženje kinematskog modela CNC stroja. Mora se ostvariti i povratna veza točne pozicije svake pojedine osi u dinamičnom modelu CNC stroja. Pomoću dinamičkog modela pokušavaju se stvoriti uvjeti kao u stvarnom pomaku osi. Za testiranje na stvarnoj opremi potrebne su minimalne izmjene algoritma. Izmjene su vezane za interpretaciju ulaznih i izlaznih vrijednosti algoritma upravljačke jedinice. Sama jezgra cijelog algoritma ostaje ista.

Ulazne vrijednosti, pozicije G-koda ne mogu se direktno učitati, nego se podaci prenose serijskom komunikacijom. Način dekodiranja vrijednosti pristigle serijskom komunikacijom su iste kao i na aplikaciji za računala, analizom znakova ASCII koda.

Ulazne vrijednosti PID kontrolera su impulsi enkodera koji su pretvoreni u translacijski pomak u milimetrima. Izlaz iz PID kontrolera je informacija pripremljena za slanje SPI komunikacijom na D/A konverter. Također, stvarna pozicija se šalje serijskom komunikacijom prema računalu koje preko računalne aplikacije stvarne pozicije osi prenosi u simulaciju. Model virtualnog CNC stroja vrši pomak koji je identičan onom pomaku pravog CNC stroja.

Testiranje stvarnog servo sustava sa jednim motorom prikazano je sljedećom slikom i video prikazom u [8]. Testiranje je izvršeno samo sa jednim motorom.



Slika 9.10 Postava za testiranje ponašanja stvarnog servo sustava

10. Zaključak

Radom je pokazan osnovni pregled 3-osne CNC glodalice, počevši od opisa osnovnih dijelova pa sve do same konstrukcije. Glavni naglasak je stavljen na upravljanje CNC stroja, gdje su spomenuti osnovni načini upravljanja i njihove karakteristike. Kod upravljačke jedinice, koja je glavni dio svakog numerički upravljanog stroja, objašnjen je svaki njezin segment, počevši od interpretera koji ima zadaću dekodiranja G-koda, u, stroju „razumljiv" jezik. Osim općenitog objašnjenja o principu rada, za svaku komponentu upravljačke jedinice je razvijen algoritam koji potkrepljuje objašnjenu teoriju. Sljedeća komponenta upravljačke jedinice je interpolator koji zajedno sa algoritmom za ubrzanje i usporenje generira profil brzine za svaku os.

Osim teoretskih objašnjenja same podjele profila brzina, razvijen je, i opisan, način generiranja profila brzine koji ovisi, kako o tehnološkom napretku razvoja hardvera i softvera, tako i o preciznosti, procesorskoj moći i kompleksnosti samog sustava. Od svih pregledno prikazanih metoda odabrana je ona koja je razvijena i koja se koristi i danas [7]. Također su stvoreni algoritmi kojima se dokazuje, unutar simulacijskog okruženja Coppelia Robototics, rad istih. U simulacijski alat je ubačen modelirani CNC stroj, istovjetan realnome stroju, na koji je uspješno implementirana razvijena upravljačka jedinica [8].

Kao sljedeći korak, u daljnjem radu, planira se implementirati razvijenu upravljačku jedinicu na realni stroj, sa nekim sitnijim konstrukcijskim izmjenama samoga stroja. Simulacija, uz testiranje samoga gibanja, svojim dinamičkim modelom može poslužiti i za testiranje povratne veze, odnosno odziva sustava na dani referentni signal [7]. Glavna ideja je transfer algoritama iz simulacijskog okruženja u mikrokontroler koji upravlja servo sustavom, na sličnome principu kako je izvedeno u [17, 18]. To je postignuto u potpunosti, te je ovim radom to i uspješno pokazano. Glavno testiranje je uspješno izvršeno u simulaciji i to je najbolja potvrda da razvijeni algoritmi upravljanja uspješno izvršavaju svoju funkciju.

Daljnja implementacija je prijenos algoritama na sve tri osi CNC glodalice i završno testiranje obradom odvajanjem čestica. Osim testiranja algoritama upravljanja svih osi, nužna su testiranja rada svih komponenti u industrijskom okruženju, koje je izloženo elektromagnetskom zračenju, i, koje može poremetiti kvalitetu prijenosa signala i rad samih komponenti.

Testiranje na stvarnoj opremi je daleko opsežniji proces zbog mogućnosti trajnog oštećenja dijelova stroja, kao i ozljede operatera na stroju. Takav proces zahtijeva još i dodatne algoritme, zadužene za sigurnost kretanja svake od osi, što dodatno drastično komplicira korištenje ionako dovoljno kompliciranih algoritama interpolacije. Iz tog razloga taj dio još nije obuhvaćen u ovome radu, nego se isključivo bavi načinom rada upravljačke jedinice, provedivom u simulaciji.

Literatura

[1] S.H. Suh, S. K. Kang, D.H. Chung, I. Stroud: Theory and Design of CNC Systems, Springer-Verlag London, 2008.

[2] https://support.industry.siemens.com/cs/start?lc=en-US, dostupno 08.07.2023.

[3] CD SITRAIN Sinumerik 840D, Siemens, 2007., dostupno 08.07.2023.

[4] CNC glodalica Cosmec 3, slikano 12.11.2019., Bjelovar

[5] https://www.solidworks.com/product/students, dostupno 08.07.2023.

[6] https://www.coppeliarobotics.com/features, dostupno 08.07.2023.

[7] https://drive.google.com/file/d/1W87i0qv4rBittU0iEuL6EIj0T1IFrxRh/view?usp=drive_link, dostupno 08.07.2023.

[8] https://drive.google.com/file/d/1J0WIeenz3C2nRlUfBj_GeuWwbCHHq7GV/view?usp=drive_link, dostupno 08.07.2023.

[9] https://www.youtube.com/watch?v=rn_JLHg_Z_U&t=2s, dostupno 08.07.2023.

[10] https://www.youtube.com/watch?v=BmBuphfE22s, dostupno 08.07.2023.

[11] M. Bošnjaković, A. Stoić: Programiranje CNC Strojeva, Slavonski Brod, 2011.

[12] https://www.enciklopedija.hr/natuknica.aspx?ID=1347, dostupno 08.07.2023.

[13] T. Udiljak: Autorizirana predavanja iz kolegija Postupci obrade odvajanjem, Fakultet strojarstva i brodogradnje, Zagreb, šk. god. 2010/2011.

[14] LX Festo: Introduction to CNC Mill, https://lx.festo.com/en/home, dostupno 18.06.2023.

[15] Boščić, Drago; Kos, Sebastijan; Pavlic, Tomislav: The Approach to Stronger Connection Between Machining Oriented Manufacturing Companies and Education Sector-Joint Cooperation // Proceedings of the CIM 2015-15th International Scientific Conference on Production Engineering / Abele, Eberhard; Udiljak, Toma ; Ciglar, Damir (ur.). Zagreb: Hrvatska udruga proizvodnog strojarstva (HUPS), 2015. str. 65-70

[16] K. Kušec, T. Pavlic, D. Boščić, M. Švegović, S. Golubić, A. Britvić: Rapid Protototyping of Rifle for Underwater Fishing by the Application of the Reverse Engineering // Proceedings of the CIM 2017-16th International Scientific Conference on Production Engineering / Abele, Eberhard ; Udiljak, Toma ; Ciglar, Damir (ur.)., Zagreb: Hrvatska udruga proizvodnog strojarstva (HUPS), 2017. str. 171-176

[17] T. Pavlic, K. Kušec, D. Radočaj, A. Britvić, M. Lukas, V. Milić, M. Crneković: Cognitive Model of the Closed Environment of a Mobile Robot Based on Measurements // Applied Sciences-Basel, 11 (2021), 6; 2786, 33 doi:10.3390/app11062786

[18] T. Pavlic: Kognitivni model zatvorene okoline mobilnoga robota temeljen na mjerenjima, doktorska disertacija, Fakultet strojarstva i brodogradnje, Zagreb, 2021.

Popis slika

Popis tablica

Prilozi

Profili brzine kružnog gibanja za dvije osi

$a = 0.04$ (−10 … 10)

$w = 0.1$ (−10 … 10)

$r = 50$ (−10 … 50)

$w_1 = -\left(a \cdot \frac{x^2}{2} + d_2\right)\{0 \le x \le t_1\}$

$f_{11}(x) = r \cdot \cos(w_1)\{0 < x \le t_1\}$

$f_{12}(x) = r \cdot \sin(w_1)\{0 < x \le t_1\}$

$\frac{d}{dx}(f_{11}(x))$

$\frac{d}{dx}(f_{12}(x))$

$w_{12} = a \cdot x\{0 \le x \le t_1\}$

$f_{13}(x) = -r \cdot w_{12} \cdot \sin(w_1)\{0 < x \le t_1\}$

$f_{14}(x) = r \cdot w_{12} \cdot \cos(w_1)\{0 < x \le t_1\}$

$f(x) = \sqrt{f_{11}(x)^2 + f_{12}(x)^2}$

$f_{15}(x) = \frac{d}{dx}(w_1)\{0 < x \le t_1\}$

$w_2 = -(w \cdot x - w \cdot t_1 + p_1 + d_2)\{t_1 < x \le t_2\}$

$f_{21}(x) = r \cdot \cos(w_2)\{t_1 < x \le t_2\}$

$f_{22}(x) = r \cdot \sin(w_2)\{t_1 < x \le t_2\}$

$\frac{d}{dx}(f_{21}(x))$

$\frac{d}{dx}(f_{22}(x))$

$f_{23}(x) = -r \cdot w \cdot \sin(w_2)\{t_1 < x \le t_2\}$

$f_{24}(x) = r \cdot w \cdot \cos(w_2)\{t_1 < x \le t_2\}$

$f(x) = \sqrt{f_{21}(x)^2 + f_{22}(x)^2}$

$f_{25}(x) = \frac{d}{dx}(w_2)$

$w_3 = -\left(a \cdot x \cdot \left(-\frac{x}{2} + t_3\right) + a \cdot t_2 \cdot \left(\frac{t_2}{2} - t_3\right) + p_2 +$

$f_{31}(x) = r \cdot \cos(w_3)\{t_2 < x \le t_3\}$

$f_{32}(x) = r \cdot \sin(w_3)\{t_2 < x \le t_3\}$

$\frac{d}{dx}(f_{31}(x))$

$\frac{d}{dx}(f_{32}(x))$

$f_{33}(x) = -r \cdot w \cdot \sin(w_3)\{t_2 < x \le t_3\}$

$f_{34}(x) = r \cdot w \cdot \cos(w_3)\{t_2 < x \le t_3\}$

$f(x) = \sqrt{f_{31}(x)^2 + f_{32}(x)^2}$

$f_{35}(x) = \frac{d}{dx}(w_3)$

$t_1 = \frac{w}{a}$ → $t_1 = 2.5$

$t_2 = \frac{p_3}{w}$ → $t_2 = 15.7079632679$

$t_3 = t_1 + t_2$ → $t_3 = 18.2079632679$

$p_1 = \frac{w \cdot t_1}{2}$ → $p_1 = 0.125$

$p_2 = \frac{w \cdot t_1}{2} + w \cdot (t_2 - t_1)$ → $p_2 = 1.44579632679$

$p_3 = \text{abs}(d - d_2)$ → $p_3 = 1.57079632679$

$d = 90 \cdot \frac{\pi}{180}$ → $d = 1.57079632679$

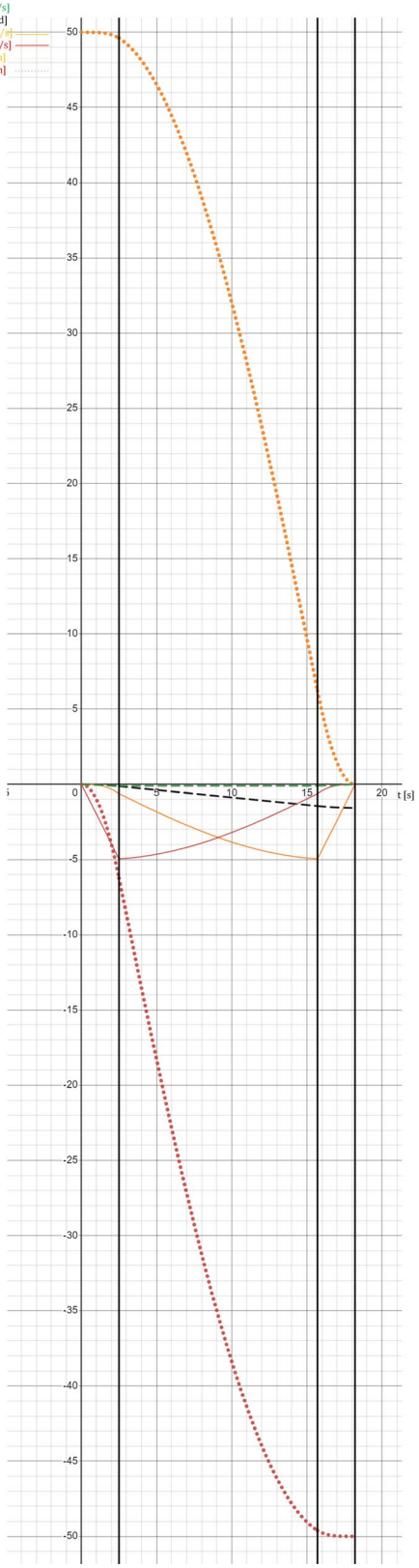
$d_2 = 0 \cdot \frac{\pi}{180}$ → $d_2 = 0$

$x = t_1$ → $x = 2.5$

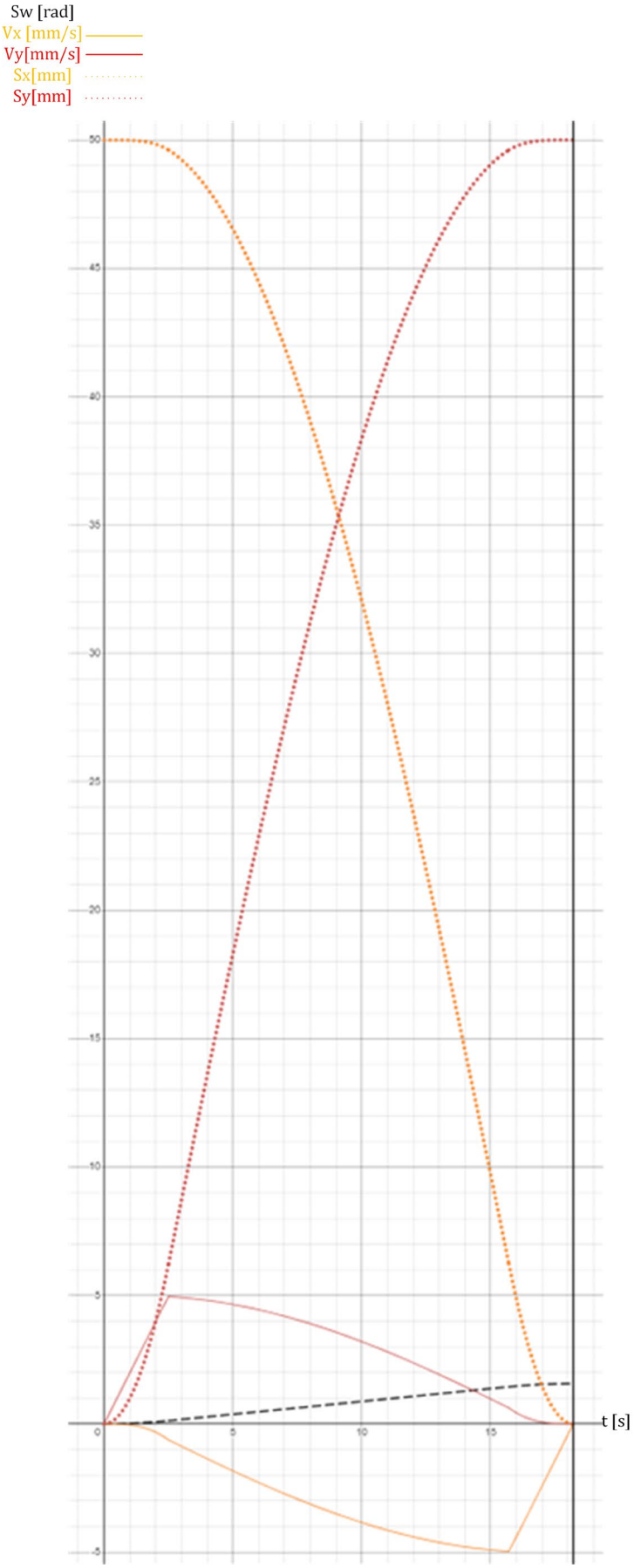
$x = t_2$ → $x = 15.7079632679$

$x = t_3$ → $x = 18.2079632679$

$x = 0$ (−10 … 10)



Ostali profili kružnih gibanja



Svi dijelovi su prikazani u dva dijela. 1.dio je deklaracija funkcija, 2. dio je implementacija algoritama

Glavni program upravljačke jedinice – GUI i interpreter

```
#ifndef MAINWINDOW_H
#define MAINWINDOW_H

#include <QMainWindow>
#include <QObject>
#include <QDebug>
#include <thread_1.h>
#include <QPainter>
#include <QColor>
#include <QKeyEvent>
#include "QtCore"
#include "QtGui"
#include <QString>
#include <QFile>
#include <QTextStream>

QT_BEGIN_NAMESPACE
namespace Ui { class MainWindow; }
QT_END_NAMESPACE

class MainWindow : public QMainWindow
{
    Q_OBJECT

public:
    MainWindow(QWidget *parent = nullptr);
    ~MainWindow();

private:
    Ui::MainWindow *ui;
    Thread_1 thread_1;

    void ReadFile();
    void ReadText();

    // variabels from reading text
    QString gcode_text;
    bool read_text;

    //float gcode_line[8]; // [Mode, X_pos, Y_pos, Feed]
    struct gLine{
        unsigned short G = 0;
        float X = 0.0f;
        float Y = 0.0f;
        float Z = 0.0f;
        float I = 0.0f;
        float J = 0.0f;
        float F = 0.0f;
        unsigned short M = 0; };
    struct gLine read_gLine;
```

```
    int i; // position in text
    int line_num; // number of line

private slots:
    void on_pushButton_clicked();

    void ReciveStatus(const float posX, const float posY, const float posZ,
const float feed);
    void ReciveNextLine();

    void on_pushButton_2_clicked();
    void on_pushButton_3_pressed();
    void on_pushButton_4_clicked();
    void on_pushButton_4_released();
    void on_pushButton_3_released();
    void on_pushButton_4_pressed();
    void on_pushButton_5_pressed();
    void on_pushButton_6_released();
    void on_pushButton_7_pressed();
    void on_pushButton_8_released();
    void on_pushButton_5_released();
    void on_pushButton_6_pressed();
    void on_pushButton_7_released();
    void on_pushButton_8_pressed();
    void on_pushButton_9_clicked();
};
#endif // MAINWINDOW_H
```

```
#include "mainwindow.h"
#include "ui_mainwindow.h"

MainWindow::MainWindow(QWidget *parent)
    : QMainWindow(parent)
    , ui(new Ui::MainWindow)
{
    ui->setupUi(this);

    read_text = false;
    i = 0;
    //thread_1.start();

    connect(&thread_1, &Thread_1::SendStatus, this,
&MainWindow::ReciveStatus);
    connect(&thread_1, &Thread_1::NextLine, this,
&MainWindow::ReciveNextLine);

    ui->pushButton_2->setDisabled(true);
    ui->spinBox->setMaximum(5000);
```

```
    thread_1.start();
}

MainWindow::~MainWindow()
{
    thread_1.thread_end = true;
    while(thread_1.isRunning()){}
    delete ui;
}

void MainWindow::ReadFile()
{

}

void MainWindow::on_pushButton_clicked()
{
    if(read_text == false)
    {
        gcode_text = ui->textEdit->toPlainText();
        read_text = true;
        ui->pushButton->setDisabled(true);
        ui->pushButton_9->setDisabled(true);
        ui->pushButton_2->setEnabled(true);
        thread_1.pos_realize = true;
    }
}

void MainWindow::on_pushButton_2_clicked()
{
    thread_1.stop = true;

    read_text = false;
    ui->pushButton->setEnabled(true);
    ui->pushButton_2->setDisabled(true);
    ui->pushButton_9->setEnabled(true);
}

void MainWindow::ReciveNextLine()
{
    ReadText();
    thread_1.UpdateGcodeLine(read_gLine.G, read_gLine.X,
read_gLine.Y, read_gLine.Z, read_gLine.I, read_gLine.J,
read_gLine.F, read_gLine.M);
}
```

```
void MainWindow::ReadText()
{
    float temp_read = 0.0f;
    float decimal_place = 0.0f;
    short tmp_read_sign = 1;
    bool I_read = false;
    bool J_read = false;

    for(;i<gcode_text.length(); i++)
        {
            if((int)gcode_text[i].unicode() == 71) // G
            {
                for(i++; (int)gcode_text[i].unicode() < 58 &&
(int)gcode_text[i].unicode() > 47; i++)
                {
                    temp_read = temp_read * 10 +
((int)gcode_text[i].unicode() - 48);
                }
                read_gLine.G = temp_read;
                temp_read = 0;
            }
            if((int)gcode_text[i].unicode() == 88) // X
            {
                if((int)gcode_text[i+1].unicode() == 45)
{tmp_read_sign = -1;i++;} else{ tmp_read_sign = 1;}
                for(i++; (int)gcode_text[i].unicode() < 58 &&
(int)gcode_text[i].unicode() > 47; i++)
                {
                    temp_read = temp_read * 10 +
((int)gcode_text[i].unicode() - 48);
                }
                if((int)gcode_text[i].unicode() == 46)
                {   decimal_place = 0.1;
                    for(i++; (int)gcode_text[i].unicode() < 58
&& (int)gcode_text[i].unicode() > 47; i++)
                    {
                        temp_read = temp_read +
(((int)gcode_text[i].unicode() - 48) * decimal_place);
                        decimal_place = decimal_place / 10;
                    }
                }
                read_gLine.X = tmp_read_sign * temp_read;
                temp_read = 0;
            }
```

```
            if((int)gcode_text[i].unicode() == 89) // Y
            {
                if((int)gcode_text[i+1].unicode() == 45)
{tmp_read_sign = -1;i++;} else{ tmp_read_sign = 1;}
                for(i++; (int)gcode_text[i].unicode() < 58 &&
(int)gcode_text[i].unicode() > 47; i++)
                {
                    temp_read = temp_read * 10 +
((int)gcode_text[i].unicode() - 48);
                }
                if((int)gcode_text[i].unicode() == 46)
                {   decimal_place = 0.1;
                    for(i++; (int)gcode_text[i].unicode() < 58
&& (int)gcode_text[i].unicode() > 47; i++)
                    {
                        temp_read = temp_read +
(((int)gcode_text[i].unicode() - 48) * decimal_place);
                        decimal_place = decimal_place / 10;
                    }
                }
                read_gLine.Y = tmp_read_sign * temp_read;
                temp_read = 0;
            }
            if((int)gcode_text[i].unicode() == 90) // Z
            {
                if((int)gcode_text[i+1].unicode() == 45)
{tmp_read_sign = -1;i++;} else{ tmp_read_sign = 1;}
                for(i++; (int)gcode_text[i].unicode() < 58 &&
(int)gcode_text[i].unicode() > 47; i++)
                {
                    temp_read = temp_read * 10 +
((int)gcode_text[i].unicode() - 48);
                }
                if((int)gcode_text[i].unicode() == 46)
                {   decimal_place = 0.1;
                    for(i++; (int)gcode_text[i].unicode() < 58
&& (int)gcode_text[i].unicode() > 47; i++)
                    {
                        temp_read = temp_read +
(((int)gcode_text[i].unicode() - 48) * decimal_place);
                        decimal_place = decimal_place / 10;
                    }
                }
                read_gLine.Z = tmp_read_sign * temp_read;
                temp_read = 0;
```

```
            }
            if((int)gcode_text[i].unicode() == 73 && I_read ==
false) // I
            {
                if((int)gcode_text[i+1].unicode() == 45)
{tmp_read_sign = -1;i++;} else{ tmp_read_sign = 1;}
                for(i++; (int)gcode_text[i].unicode() < 58 &&
(int)gcode_text[i].unicode() > 47; i++)
                {
                    temp_read = temp_read * 10 +
((int)gcode_text[i].unicode() - 48);
                }
                if((int)gcode_text[i].unicode() == 46)
                {   decimal_place = 0.1;
                    for(i++; (int)gcode_text[i].unicode() < 58
&& (int)gcode_text[i].unicode() > 47; i++)
                    {
                        temp_read = temp_read +
(((int)gcode_text[i].unicode() - 48) * decimal_place);
                        decimal_place = decimal_place / 10;
                    }
                }
                read_gLine.I = tmp_read_sign * temp_read;
                temp_read = 0;
                I_read = true;
            }
            else if(I_read == false) {read_gLine.I = 0.0f;}
            if((int)gcode_text[i].unicode() == 74 && J_read ==
false) // J
            {
                if((int)gcode_text[i+1].unicode() == 45)
{tmp_read_sign = -1;i++;} else{ tmp_read_sign = 1;}
                for(i++; (int)gcode_text[i].unicode() < 58 &&
(int)gcode_text[i].unicode() > 47; i++)
                {
                    temp_read = temp_read * 10 +
((int)gcode_text[i].unicode() - 48);
                }
                if((int)gcode_text[i].unicode() == 46)
                {   decimal_place = 0.1;
                    for(i++; (int)gcode_text[i].unicode() < 58
&& (int)gcode_text[i].unicode() > 47; i++)
                    {
                        temp_read = temp_read +
(((int)gcode_text[i].unicode() - 48) * decimal_place);
```

```
                        decimal_place = decimal_place / 10;
                    }
                }
                read_gLine.J = tmp_read_sign * temp_read;
                temp_read = 0;
                J_read = true;
            }
            else if(J_read == false) {read_gLine.J = 0.0f;}
            if((int)gcode_text[i].unicode() == 70) // F
            {
                for(i++; (int)gcode_text[i].unicode() < 58 &&
(int)gcode_text[i].unicode() > 47; i++)
                {
                    temp_read = temp_read * 10 +
((int)gcode_text[i].unicode() - 48);
                }
                if((int)gcode_text[i].unicode() == 46)
                {   decimal_place = 0.1;
                    for(i++; (int)gcode_text[i].unicode() < 58
&& (int)gcode_text[i].unicode() > 47; i++)
                    {
                        temp_read = temp_read +
(((int)gcode_text[i].unicode() - 48) * decimal_place);
                        decimal_place = decimal_place / 10;
                    }
                }
                read_gLine.F = temp_read;
                temp_read = 0;
            }
            if((int)gcode_text[i].unicode() == 77) // M
            {
                for(i++; (int)gcode_text[i].unicode() < 58 &&
(int)gcode_text[i].unicode() > 47; i++)
                {
                    temp_read = temp_read * 10 +
((int)gcode_text[i].unicode() - 48);
                }
                read_gLine.M = temp_read;
                temp_read = 0;
            }
            if((int)gcode_text[i].unicode() == 10)
            {
                i++;
                break;
            }
```

```
        }
}

void MainWindow::ReciveStatus(const float posX, const float
posY, const float posZ, const float feed)
{
    ui->cur_pos_X->setText(QString::number(posX));
    ui->cur_pos_Y->setText(QString::number(posY));
    ui->cur_pos_Z->setText(QString::number(posZ));
    ui->cur_feed->setText(QString::number(feed));
}

// MANUAL BUTTONS
void MainWindow::on_pushButton_3_pressed()
{
    thread_1.manual_feed = ui->spinBox->value();
    thread_1.manual_dirX = 1;
    thread_1.ManualInterpolate();
}
void MainWindow::on_pushButton_3_released()
{
    thread_1.manual_stop = true;
    thread_1.manual_dirX = 0;
}
void MainWindow::on_pushButton_4_pressed()
{
    thread_1.manual_feed = ui->spinBox->value();
    thread_1.manual_dirX = -1;
    thread_1.ManualInterpolate();
}
void MainWindow::on_pushButton_4_released()
{
    thread_1.manual_stop = true;
    thread_1.manual_dirX = 0;
}
void MainWindow::on_pushButton_4_clicked()
{  }
void MainWindow::on_pushButton_5_pressed()
{
    thread_1.manual_feed = ui->spinBox->value();
    thread_1.manual_dirY = 1;
    thread_1.ManualInterpolate();
}
void MainWindow::on_pushButton_5_released()
{
```

```
    thread_1.manual_stop = true;
    thread_1.manual_dirY = 0;
}
void MainWindow::on_pushButton_6_pressed()
{
    thread_1.manual_feed = ui->spinBox->value();
    thread_1.manual_dirY = -1;
    thread_1.ManualInterpolate();
}
void MainWindow::on_pushButton_6_released()
{
    thread_1.manual_stop = true;
    thread_1.manual_dirY = 0;
}
void MainWindow::on_pushButton_7_pressed()
{
    thread_1.manual_feed = ui->spinBox->value();
    thread_1.manual_dirZ = 1;
    thread_1.ManualInterpolate();
}
void MainWindow::on_pushButton_7_released()
{
    thread_1.manual_stop = true;
    thread_1.manual_dirZ = 0;
}
void MainWindow::on_pushButton_8_pressed()
{
    thread_1.manual_feed = ui->spinBox->value();
    thread_1.manual_dirZ = -1;
    thread_1.ManualInterpolate();
}
void MainWindow::on_pushButton_8_released()
{
    thread_1.manual_stop = true;
    thread_1.manual_dirZ = 0;
}

void MainWindow::on_pushButton_9_clicked()
{
    thread_1.ResetCordianteSys();
}
```

Za provođenje više procesa odjednom koriste se dodatni *thread*-ovi

***Thread_1* provodi interpolaciju putanje**

```
#ifndef THREAD_1_H
#define THREAD_1_H
```

```
#include <QObject>
#include <QtCore>
#include <QDebug>
#include <QtGlobal>
#include <math.h>
#include <QString>
#include <QFile>
#include <QTextStream>
#include <thread_2.h>
#include <QTimer>

#include <interpolation.h>

class Thread_1 : public QThread
{
    Q_OBJECT

public:
    Thread_1(QObject *parent = nullptr);
    ~Thread_1();
    bool thread_end;

    void run();
    Thread_2 thread_2;

    // INTERPOLATE FUNCTIONS
    void Inicialization();
    void Interpolate();
    void IntrPar(float feedX, float feedY, float feedZ, float diff_posX,
float diff_posY, float diff_posZ, int dirX, int dirY, int dirZ, float W,
float Aw, float angle, float angle_1);

    void ManualInterpolate();
    void ResetCordianteSys();
    //void WriteFile (float timer, float abs_posX, float abs_posY, float
abs_posZ, float velX, float velY, float velZ);
    void WriteSim(float posX, float posY, float posZ, float velX, float
velY, float velZ);
    void UpdateGcodeLine(unsigned short, float, float, float, float, float,
float, unsigned short);
    void UpdateGcodeLineFirstCall(float gcode_line[8]);
    float Rou(float num);

    // INTERPOLATE VARIABLES
    const float pi = 3.14159265359f;
    // CNC vaules desired position
    // N G X Y Z I J F
    struct gLine{
        unsigned short G = 0;
        float X = 0.0f;
        float Y = 0.0f;
        float Z = 0.0f;
        float I = 0.0f;
```

```
        float J = 0.0f;
        float F = 0.0f;
        unsigned short M = 0; };
    struct gLine first_gLine;
    struct gLine sec_gLine;

    float acc;

    float a1_X;
    float a3_X;
    float a1_Y;
    float a3_Y;
    float a1_Z;
    float a3_Z;
    float Ta;
    float Td;
    float Tc;
    int itr;

    float Titr; // update time

    // discrete values
    int J1;
    int J2;
    int J3;

    struct pos{
        float X = 0.0f;
        float Y = 0.0f;
        float Z = 0.0f; };
    struct pos absPos;
    struct pos incPos;
    struct pos rouPos;

    struct vel{
        float X = 0.0f;
        float Y = 0.0f;
        float Z = 0.0f; };
    struct vel curVel; // velocity in TimerIntr
    struct vel lstVel;
    struct vel rouVel;

    // mode of moving
    int mode_move;

    // for circular interpoaltion
    float radius;
    float velW; // angular velocity
    float posW; // angular acceleration
    float last_velW; // angular velocity
    float last_posW; // angular acceleration

    float W; // feed / R
    float Aw; // acc / R
```

```
    bool runTimer;
    bool read_next_line;
    bool pos_realize;

    // variables for manual
    bool manual_active;
    bool manual_stop;
    int manual_dirX; // 0 stop
    int manual_dirY; // 1 +
    int manual_dirZ; // 1 -
    int manual_feed; // feedrate for manual move

    bool stop;
    bool program_end;

    // Timer
    QTimer *timer = new QTimer(this);

    QElapsedTimer elapsed_time;

public slots:
    void TimerIntr(); // timer interrupt

signals:
    void SendStatus(float posX, float posY, float posZ, float feed);
    void NextLine();

};

#endif // THREAD_1_H
```

```
#include "thread_1.h"

Thread_1::Thread_1(QObject *parent)
    :QThread(parent)
{
    thread_end = false;
    // Timer
    connect(timer, &QTimer::timeout, this, &Thread_1::TimerIntr);

    // INTERPOLATE
    thread_2.Inicialization();
    thread_2.SetPosition(0,0,0);

    Titr = 0.001f;
    acc = 2; //1000.0f;

    last_velW = 0.0f;
    last_posW = 0.0f;

    radius = 0.0f;

    mode_move = 0;
```

```
    J1 = 0;
    J2 = 0;
    J3 = 0;

    // manual variables inicialization
    manual_active = false;
    manual_stop = false;
    manual_dirX = 0;
    manual_dirY = 0;
    manual_dirZ = 0;
    manual_feed = 0.0f;

    timer->start(1);
    runTimer = false;
    read_next_line = false;
    pos_realize = false;
    stop = false;
    program_end = false;
}

Thread_1::~Thread_1()
{
    while(thread_2.isRunning()){}
}

void Thread_1::Inicialization()
{
}

void Thread_1::UpdateGcodeLineFirstCall(float *gcode_line)
{
    /*for(int i=0; i<7; i++)
    {
        gcode_line_curr[i] = gcode_line[i];
        gcode_line_next[i] = gcode_line[i];
    }*/
}

void Thread_1::UpdateGcodeLine(unsigned short G, float X, float Y, float Z,
float I, float J, float F, unsigned short M)
{
    first_gLine.G = G;
    first_gLine.X = X;
    first_gLine.Y = Y;
    first_gLine.Z = Z;
    first_gLine.I = I;
    first_gLine.J = J;
    first_gLine.F = F;
    first_gLine.M = M;

    read_next_line = true;

    if(first_gLine.M == 30){program_end = true;}
```

```
    //qDebug() << gcode_line_curr[0]<<" "<<gcode_line_curr[1]<<"
"<<gcode_line_curr[2]<<" "<<gcode_line_curr[3]<<" "<<"
"<<gcode_line_curr[4]<<" "<<gcode_line_curr[5]<<" "<<gcode_line_curr[6];
}

void Thread_1::run()
{
while(thread_end == false)
{
    if(pos_realize == true && program_end == false)
    {
        pos_realize = false;

        emit NextLine();
        while(read_next_line == false){}
        read_next_line = false;

        Interpolate();

        runTimer = true;
    }
}
}

void Thread_1::Interpolate()
{
    // difference vaules
    float diff_posX = 0.0f;
    float diff_posY = 0.0f;
    float diff_posZ = 0.0f;

    // velocity vectors
    float feedX = 0.0f;
    float feedY = 0.0f;
    float feedZ = 0.0f;
    int dirX = 1;
    int dirY = 1;
    int dirZ = 1;
    itr = 1;

    // for circular interpolation
    float angle_1 = 0.0f;
    float angle_2 = 0.0f;
    float I_2 = 0.0f;
    float J_2 = 0.0f;
    float angle = 0.0f;


    // ---determine case of moving---
    if(first_gLine.G == 0 || first_gLine.G == 1)
    {
        diff_posX = first_gLine.X - incPos.X;
        diff_posY = first_gLine.Y - incPos.Y;
        diff_posZ = first_gLine.Z - incPos.Z;
```

```
        // calculating velocity vector
        // round number
        if(diff_posX == 0.0f && diff_posY != 0.0f && diff_posZ != 0.0f)
        { feedX = 0;
            feedY = cos(atan( abs(diff_posZ/diff_posY) )) * first_gLine.F;
            feedZ = sin(atan( abs(diff_posZ/diff_posY) )) * first_gLine.F; }
        else if(diff_posX != 0.0f && diff_posY == 0.0f && diff_posZ != 0.0f)
        { feedY = 0;
            feedX = cos(atan( abs(diff_posZ/diff_posX) )) * first_gLine.F;
            feedZ = sin(atan( abs(diff_posZ/diff_posX) )) * first_gLine.F; }
        else if(diff_posX != 0.0f && diff_posY != 0.0f && diff_posZ == 0.0f)
        { feedZ = 0;
            feedX = cos(atan( abs(diff_posY/diff_posX) )) * first_gLine.F;
            feedY = sin(atan( abs(diff_posY/diff_posX) )) * first_gLine.F; }
        else if(diff_posX == 0.0f && diff_posY == 0.0f && diff_posZ != 0.0f)
        { feedX = 0;
            feedY = 0;
            feedZ = first_gLine.F; }
        else if(diff_posX == 0.0f && diff_posY != 0.0f && diff_posZ == 0.0f)
        { feedX = 0;
            feedY = first_gLine.F;
            feedZ = 0; }
        else if(diff_posX == 0.0f && diff_posY == 0.0f && diff_posZ != 0.0f)
        { feedX = first_gLine.F;
            feedY = 0;
            feedZ = 0; }
        else
        { feedX = cos(atan( sqrt(pow(diff_posY,2) + pow(diff_posZ,2)) /
abs(diff_posX) ) ) * first_gLine.F;
            feedY = cos(atan( sqrt(pow(diff_posX,2) + pow(diff_posZ,2)) /
abs(diff_posY) ) ) * first_gLine.F;
            feedZ = cos(atan( sqrt(pow(diff_posX,2) + pow(diff_posY,2)) /
abs(diff_posZ) ) ) * first_gLine.F; }

        if(diff_posX < 0.0f){dirX = -1;}
        if(diff_posY < 0.0f){dirY = -1;}
        if(diff_posZ < 0.0f){dirZ = -1;}

        // X trapeze and Y trapeze Z triangle
        if ( (abs(diff_posX) >= abs(diff_posY) && abs(diff_posX) >=
abs(diff_posZ) ) && ((pow(feedX, 2)/(2*acc) + pow(feedX, 2)/(2*acc)) <
abs(diff_posX)) )
        { mode_move = 1; }
        else if ( (abs(diff_posY) > abs(diff_posX) && abs(diff_posY) >
abs(diff_posZ) ) && ((pow(feedY, 2)/(2*acc) + pow(feedY, 2)/(2*acc)) <
abs(diff_posY)) )
        { mode_move = 2; }
        else if ( (abs(diff_posZ) > abs(diff_posX) && abs(diff_posZ) >
abs(diff_posY) ) && ((pow(feedZ, 2)/(2*acc) + pow(feedZ, 2)/(2*acc)) <
abs(diff_posZ)) )
        { mode_move = 3; }

        // X triangle and Y triangle and Z triange
```

```
        else if ( (abs(diff_posX) >= abs(diff_posY) && abs(diff_posX) >=
abs(diff_posZ) ) && ((pow(feedX, 2)/(2*acc) + pow(feedX, 2)/(2*acc)) >=
abs(diff_posX)) )
        { mode_move = 4; }
        else if ( (abs(diff_posY) > abs(diff_posX) && abs(diff_posY) >
abs(diff_posZ) ) && ((pow(feedY, 2)/(2*acc) + pow(feedY, 2)/(2*acc)) >=
abs(diff_posY)) )
        { mode_move = 5; }
        else if ( (abs(diff_posZ) > abs(diff_posX) && abs(diff_posZ) >
abs(diff_posY) ) && ((pow(feedZ, 2)/(2*acc) + pow(feedZ, 2)/(2*acc)) >=
abs(diff_posZ)) )
        { mode_move = 6; }
        else {qDebug() << "Can't find mode move";}
    }
    // circular interpolation
    else if(first_gLine.G == 2 || first_gLine.G == 3)
    {
        // incremental position of circle center
        radius = sqrt(pow(first_gLine.I, 2) + pow(first_gLine.J, 2));

        // calculating angular vel i acc
        W = first_gLine.F / radius;
        Aw = acc / radius;
        // angle_1 & angle_2
        // angle_1 is starting point
        // angle_2 is ending point
        if(first_gLine.J > 0)
        { angle_1 = 6.283185307f - acos(-first_gLine.I/radius); }
        else if(first_gLine.I <= 0)
        { angle_1 = acos(-first_gLine.I/radius); }
        ///////////////
        /// calculate I and J on ending angle

        I_2 = first_gLine.X - ( incPos.X + first_gLine.I );
        J_2 = first_gLine.J - ( incPos.Y + first_gLine.J );

        if ( I_2 == 0 && J_2 >= 0)
        {
            angle_2 = 1.570796327f; // 90 °
        }
        else if ( I_2 == 0 && J_2 < 0)
        {
            angle_2 = 4.71238898f; // 270°
        }
        else if ( I_2 > 0 && J_2 >= 0 ) // 1. quadrant
        {
            angle_2 = atan(J_2 / I_2);
        }
        else if ( I_2 < 0 && J_2 >= 0 ) // 2. quadrant
        {
            angle_2 = pi + atan(J_2 / I_2);
        }
        else if ( I_2 < 0 && J_2 < 0 ) // 3. quadrant
        {
```

```
            angle_2 = atan(J_2 / I_2) + pi;
        }
        else if ( I_2 > 0 && J_2 < 0 ) // 4. quadrant
        {
            angle_2 = 6.283185307f + atan(J_2 / I_2);
        }

        // calculating angle
        // 2 circular
        if(angle_1 >= angle_2 && first_gLine.G == 2) { angle = angle_1 -
angle_2; }
        else if(angle_1 < angle_2 && first_gLine.G == 2) { angle = angle_1 +
(6.283185307f - angle_2); }
        // 3 circular
        else if(angle_1 <= angle_2 && first_gLine.G == 3) { angle = angle_2
- angle_1; }
        else if(angle_1 > angle_2 && first_gLine.G == 3) { angle = angle_2 +
(6.283185307f - angle_1); }

        // mode move
        // 2 circular
        if( (pow(W, 2)/(2*Aw) + pow(W, 2)/(2*Aw)) < angle && first_gLine.G
== 2)
        { mode_move = 7; }
        else if( (pow(W, 2)/(2*Aw) + pow(W, 2)/(2*Aw)) >= angle &&
first_gLine.G == 2)
        { mode_move = 8; }
        // 3 circular
        else if( (pow(W, 2)/(2*Aw) + pow(W, 2)/(2*Aw)) < angle &&
first_gLine.G == 3)
        { mode_move = 9; }
        else if( (pow(W, 2)/(2*Aw) + pow(W, 2)/(2*Aw)) >= angle &&
first_gLine.G == 3)
        { mode_move = 10; }
        else {qDebug() << "Can't find mode move";}
    }

    IntrPar(feedX, feedY, feedZ, diff_posX, diff_posY, diff_posZ, dirX,
dirY, dirZ, W, Aw, angle, angle_1);
}

void Thread_1::IntrPar(float feedX, float feedY, float feedZ, float
diff_posX, float diff_posY, float diff_posZ, int dirX, int dirY, int dirZ,
float W, float Aw, float angle, float angle_1)
{
    switch(mode_move)
       {
       case 1: // X trapeze
           // times calculating
           Ta = abs( feedX / acc );
           Td = abs( feedX / acc );
           Tc = ( abs(diff_posX) - (pow(feedX,2)/(2*acc)) -
(pow(feedX,2)/(2*acc)) ) / feedX ;
```

```
            J1 = (int)( (Ta/Titr) + 0.5f );
            J2 = (int)( (Tc/Titr) + 0.5f );
            J3 = (int)( (Td/Titr) + 0.5f );

            a1_X = dirX * feedX / (J1*Titr);
            a3_X = dirX * -feedX / (J3*Titr);
            a1_Y = dirY * feedY / (J1*Titr);
            a3_Y = dirY * -feedY / (J3*Titr);
            a1_Z = dirZ * feedZ / (J1*Titr);
            a3_Z = dirZ * -feedZ / (J3*Titr);
        break;
        case 2: // Y trapeze
            // times calculating
            Ta = abs( feedY / acc );
            Td = abs( feedY / acc );
            Tc = ( abs(diff_posY) - (pow(feedY,2)/(2*acc)) -
(pow(feedY,2)/(2*acc)) ) / feedY ;

            J1 = (int)( (Ta/Titr) + 0.5f );
            J2 = (int)( (Tc/Titr) + 0.5f );
            J3 = (int)( (Td/Titr) + 0.5f );

            a1_X = dirX * feedX / (J1*Titr);
            a3_X = dirX * -feedX / (J3*Titr);
            a1_Y = dirY * feedY / (J1*Titr);
            a3_Y = dirY * -feedY / (J3*Titr);
            a1_Z = dirZ * feedZ / (J1*Titr);
            a3_Z = dirZ * -feedZ / (J3*Titr);
        break;
        case 3: // Z trapeze
            // times calculating
            Ta = abs( feedZ / acc );
            Td = abs( feedZ / acc );
            Tc = ( abs(diff_posZ) - (pow(feedZ,2)/(2*acc)) -
(pow(feedZ,2)/(2*acc)) ) / feedZ ;

            J1 = (int)( (Ta/Titr) + 0.5f );
            J2 = (int)( (Tc/Titr) + 0.5f );
            J3 = (int)( (Td/Titr) + 0.5f );

            a1_X = dirX * feedX / (J1*Titr);
            a3_X = dirX * -feedX / (J3*Titr);
            a1_Y = dirY * feedY / (J1*Titr);
            a3_Y = dirY * -feedY / (J3*Titr);
            a1_Z = dirZ * feedZ / (J1*Titr);
            a3_Z = dirZ * -feedZ / (J3*Titr);
        break;
        case 4: // X triangle
            feedX = sqrt(acc * abs(diff_posX));
            feedY = feedX * (abs(diff_posY/diff_posX));
            feedZ = feedX * (abs(diff_posZ/diff_posX));

            // times calculating
            Ta = abs( feedX / acc );
```

```
            Td = abs( feedX / acc );

            J1 = (int)( (Ta/Titr) + 0.5f );
            J3 = (int)( (Td/Titr) + 0.5f );

            a1_X = dirX * feedX / (J1*Titr);
            a3_X = dirX * -feedX / (J3*Titr);
            a1_Y = dirY * feedY / (J1*Titr);
            a3_Y = dirY * -feedY / (J3*Titr);
            a1_Z = dirZ * feedZ / (J1*Titr);
            a3_Z = dirZ * -feedZ / (J3*Titr);
        break;
        case 5: // Y triangle
            feedY = sqrt(acc * abs(diff_posY));
            feedX = feedY * (abs(diff_posX/diff_posY));
            feedZ = feedY * (abs(diff_posZ/diff_posY));

            // times calculating
            Ta = abs( feedY / acc );
            Td = abs( feedY / acc );

            J1 = (int)( (Ta/Titr) + 0.5f );
            J3 = (int)( (Td/Titr) + 0.5f );

            a1_X = dirX * feedX / (J1*Titr);
            a3_X = dirX * -feedX / (J3*Titr);
            a1_Y = dirY * feedY / (J1*Titr);
            a3_Y = dirY * -feedY / (J3*Titr);
            a1_Z = dirZ * feedZ / (J1*Titr);
            a3_Z = dirZ * -feedZ / (J3*Titr);
        break;
        case 6: // Z triangle
            feedZ = sqrt(acc * abs(diff_posZ));
            feedX = feedZ * (abs(diff_posX/diff_posZ));
            feedY = feedZ * (abs(diff_posY/diff_posZ));

            // times calculating
            Ta = abs( feedZ / acc );
            Td = abs( feedZ / acc );

            J1 = (int)( (Ta/Titr) + 0.5f );
            J3 = (int)( (Td/Titr) + 0.5f );

            a1_X = dirX * feedX / (J1*Titr);
            a3_X = dirX * -feedX / (J3*Titr);
            a1_Y = dirY * feedY / (J1*Titr);
            a3_Y = dirY * -feedY / (J3*Titr);
            a1_Z = dirZ * feedZ / (J1*Titr);
            a3_Z = dirZ * -feedZ / (J3*Titr);
        break;
            // Circular
        case 7: // G2 trapeze
            // times calculating
            Ta = abs( W / Aw );
```

```
            Td = abs( W / Aw );
            Tc = abs( ( angle - (pow(W,2)/(2*Aw)) - (pow(W,2)/(2*Aw)) ) / W
);

            J1 = (int)( (Ta/Titr) + 0.5f );
            J2 = (int)( (Tc/Titr) + 0.5f );
            J3 = (int)( (Td/Titr) + 0.5f );

            a1_X = dirX * -W / (J1*Titr); // its only on acc
            a3_X = dirX * W / (J3*Titr); // "x" is used

            posW = angle_1;
        break;
        case 8: // G3 trapeze
            W = sqrt(Aw * angle);

            // times calculating
            Ta = abs( W / Aw );
            Td = abs( W / Aw );

            J1 = (int)( (Ta/Titr) + 0.5f );
            J3 = (int)( (Td/Titr) + 0.5f );

            a1_X = dirX * -W / (J1*Titr); // its only on acc
            a3_X = dirX * W / (J3*Titr); // "x" is used

            posW = angle_1;
        break;
        case 9: // G2 triangle
            // times calculating
            Ta = abs( W / Aw );
            Td = abs( W / Aw );
            Tc = abs( ( angle - (pow(W,2)/(2*Aw)) - (pow(W,2)/(2*Aw)) ) / W
);

            J1 = (int)( (Ta/Titr) + 0.5f );
            J2 = (int)( (Tc/Titr) + 0.5f );
            J3 = (int)( (Td/Titr) + 0.5f );

            a1_X = dirX * W / (J1*Titr); // its only on acc
            a3_X = dirX * -W / (J3*Titr); // "x" is used

            posW = angle_1;
        break;
        case 10: // G3 triangle
            W = sqrt(Aw * angle);

            // times calculating
            Ta = abs( W / Aw );
            Td = abs( W / Aw );

            J1 = (int)( (Ta/Titr) + 0.5f );
            J3 = (int)( (Td/Titr) + 0.5f );
```

```
            a1_X = dirX * W / (J1*Titr); // its only on acc
            a3_X = dirX * -W / (J3*Titr); // "x" is used

            posW = angle_1;
        break;
    }
}

void Thread_1::ManualInterpolate()
{
    /*
    if(manual_active == false && manual_stop == false)
    {
        manual_active = true;

        itr = 1;

        J1 = 0;
        J2 = 0;
        J3 = 0;

        // times calculating
        Ta = abs( manual_feed / acc );
        Td = abs( manual_feed / acc );

        J1 = (int)( (Ta/Titr) + 0.5f );
        J3 = (int)( (Td/Titr) + 0.5f );

        a1_X = (manual_dirX * manual_feed) / (J1*Titr);
        a3_X = (manual_dirX * -manual_feed) / (J3*Titr);
        a1_Y = (manual_dirY * manual_feed) / (J1*Titr);
        a3_Y = (manual_dirY * -manual_feed) / (J3*Titr);
        a1_Z = (manual_dirZ * manual_feed) / (J1*Titr);
        a3_Z = (manual_dirZ * -manual_feed) / (J3*Titr);

        runTimer = true;
    }
    */
}

void Thread_1::TimerIntr() //  ------ TIMER -----
{
if(runTimer == true)
{   // ---- LINEAR
    if(mode_move == 1 || mode_move == 2 || mode_move == 3)
    {
        if(stop == true)
        {
            curVel.X = curVel.X + (a3_X * Titr);
            curVel.Y = curVel.Y + (a3_Y * Titr);
            curVel.Z = curVel.Z + (a3_Z * Titr);

            incPos.X = incPos.X + (curVel.X * Titr);
            incPos.Y = incPos.Y + (curVel.Y * Titr);
```

```
            incPos.Z = incPos.Z + (curVel.Z * Titr);
        }
        else if(itr <= J1 && stop == false)
        {
            curVel.X = curVel.X + (a1_X * Titr);
            curVel.Y = curVel.Y + (a1_Y * Titr);
            curVel.Z = curVel.Z + (a1_Z * Titr);

            incPos.X = incPos.X + (curVel.X * Titr);
            incPos.Y = incPos.Y + (curVel.Y * Titr);
            incPos.Z = incPos.Z + (curVel.Z * Titr);
        }
        else if(itr <= J1+J2 && stop == false)
        {
            incPos.X = incPos.X + (curVel.X * Titr);
            incPos.Y = incPos.Y + (curVel.Y * Titr);
            incPos.Z = incPos.Z + (curVel.Z * Titr);
        }
        else if(itr <= J1+J2+J3 && stop == false)
        {
            curVel.X = curVel.X + (a3_X * Titr);
            curVel.Y = curVel.Y + (a3_Y * Titr);
            curVel.Z = curVel.Z + (a3_Z * Titr);

            incPos.X = incPos.X + (curVel.X * Titr);
            incPos.Y = incPos.Y + (curVel.Y * Titr);
            incPos.Z = incPos.Z + (curVel.Z * Titr);
        }

        rouVel.X = Rou(curVel.X);
        rouVel.Y = Rou(curVel.Y);
        rouVel.Z = Rou(curVel.Z);

        if (stop == true && rouVel.X == 0.0 && rouVel.Y == 0.0 && rouVel.Z
== 0.0){ runTimer = false; mode_move = 0; }
        else if (itr == J1+J2+J3){ runTimer = false; mode_move = 0;
pos_realize = true; }
        itr ++;
    }
    else if(mode_move == 4 || mode_move == 5 || mode_move == 6)
    {
        if(stop == true)
        {
            curVel.X = curVel.X + (a3_X * Titr);
            curVel.Y = curVel.Y + (a3_Y * Titr);
            curVel.Z = curVel.Z + (a3_Z * Titr);

            incPos.X = incPos.X + (curVel.X * Titr);
            incPos.Y = incPos.Y + (curVel.Y * Titr);
            incPos.Z = incPos.Z + (curVel.Z * Titr);
        }
        else if(itr <= J1 && stop == false)
        {
            curVel.X = curVel.X + (a1_X * Titr);
```

```
            curVel.Y = curVel.Y + (a1_Y * Titr);
            curVel.Z = curVel.Z + (a1_Z * Titr);

            incPos.X = incPos.X + (curVel.X * Titr);
            incPos.Y = incPos.Y + (curVel.Y * Titr);
            incPos.Z = incPos.Z + (curVel.Z * Titr);
        }
        else if(itr <= J1+J3 && stop == false)
        {
            curVel.X = curVel.X + (a3_X * Titr);
            curVel.Y = curVel.Y + (a3_Y * Titr);
            curVel.Z = curVel.Z + (a3_Z * Titr);

            incPos.X = incPos.X + (curVel.X * Titr);
            incPos.Y = incPos.Y + (curVel.Y * Titr);
            incPos.Z = incPos.Z + (curVel.Z * Titr);
        }

        rouVel.X = Rou(curVel.X);
        rouVel.Y = Rou(curVel.Y);
        rouVel.Z = Rou(curVel.Z);

        if (stop == true && rouVel.X == 0.0 && rouVel.Y == 0.0 && rouVel.Z
== 0.0){ runTimer = false; mode_move = 0; }
        else if (itr == J1+J2+J3){ runTimer = false; mode_move = 0;
pos_realize = true; }
        itr ++;
    }
    // ---- CIRCULAR
    else if(mode_move == 7 || mode_move == 9)
    {
        if(stop == true)
        {
            velW = velW + (a3_X * Titr);
            posW = posW + (velW * Titr);

            curVel.X = -radius * velW * sin( posW );
            curVel.Y = radius * velW * cos( posW );

            incPos.X = incPos.X + ( curVel.X * Titr );
            incPos.Y = incPos.Y + ( curVel.Y * Titr );
        }
        else if(itr <= J1 && stop == false)
        {
            velW = velW + (a1_X * Titr);
            posW = posW + (velW * Titr);

            curVel.X = -radius * velW * sin( posW );
            curVel.Y = radius * velW * cos( posW );

            incPos.X = incPos.X + ( curVel.X * Titr );
            incPos.Y = incPos.Y + ( curVel.Y * Titr );
        }
        else if(itr <= J1+J2 && stop == false)
```

```
        {
            posW = posW + (velW * Titr);

            curVel.X = -radius * velW * sin( posW );
            curVel.Y = radius * velW * cos( posW );

            incPos.X = incPos.X + ( curVel.X * Titr );
            incPos.Y = incPos.Y + ( curVel.Y * Titr );
        }
        else if(itr <= J1+J2+J3 && stop == false)
        {
            velW = velW + (a3_X * Titr);
            posW = posW + (velW * Titr);

            curVel.X = -radius * velW * sin( posW );
            curVel.Y = radius * velW * cos( posW );

            incPos.X = incPos.X + ( curVel.X * Titr );
            incPos.Y = incPos.Y + ( curVel.Y * Titr );
        }

        rouVel.X = Rou(curVel.X);
        rouVel.Y = Rou(curVel.Y);
        rouVel.Z = Rou(curVel.Z);

        if (stop == true && rouVel.X == 0.0 && rouVel.Y == 0.0 && rouVel.Z
== 0.0){ runTimer = false; mode_move = 0; }
        else if (itr == J1+J2+J3){ runTimer = false; mode_move = 0;
pos_realize = true; }
        itr ++;
    }
    else if(mode_move == 8 || mode_move == 10)
    {
        if(stop == true)
        {
            velW = velW + (a3_X * Titr);
            posW = posW + (velW * Titr);

            curVel.X = -radius * velW * sin( posW );
            curVel.Y = radius * velW * cos( posW );

            incPos.X = incPos.X + ( curVel.X * Titr );
            incPos.Y = incPos.Y + ( curVel.Y * Titr );
        }
        else if(itr <= J1)
        {
            velW = velW + (a1_X * Titr);
            posW = posW + (velW * Titr);

            curVel.X = -radius * velW * sin( posW );
            curVel.Y = radius * velW * cos( posW );

            incPos.X = incPos.X + ( curVel.X * Titr );
            incPos.Y = incPos.Y + ( curVel.Y * Titr );
```

```
        }
        else if(itr <= J1+J3)
        {
            velW = velW + (a3_X * Titr);
            posW = posW + (velW * Titr);

            curVel.X = -radius * velW * sin( posW );
            curVel.Y = radius * velW * cos( posW );

            incPos.X = incPos.X + ( curVel.X * Titr );
            incPos.Y = incPos.Y + ( curVel.Y * Titr );
        }

        rouVel.X = Rou(curVel.X);
        rouVel.Y = Rou(curVel.Y);
        rouVel.Z = Rou(curVel.Z);

        if (stop == true && rouVel.X == 0.0 && rouVel.Y == 0.0 && rouVel.Z
== 0.0){ runTimer = false; mode_move = 0; }
        else if (itr == J1+J2+J3){ runTimer = false; mode_move = 0;
pos_realize = true; }
        itr ++;
    }
    // ---- MANUAL MOVING
    else if(mode_move == 0 && manual_active == true)
    {
        if(manual_stop == true)
        {
            curVel.X = curVel.X + (a3_X * Titr);
            curVel.Y = curVel.Y + (a3_Y * Titr);
            curVel.Z = curVel.Z + (a3_Z * Titr);

            incPos.X = incPos.X + (curVel.X * Titr);
            incPos.Y = incPos.Y + (curVel.Y * Titr);
            incPos.Z = incPos.Z + (curVel.Z * Titr);
        }
        else if(itr <= J1 && manual_stop == false)
        {
            curVel.X = curVel.X + (a1_X * Titr);
            curVel.Y = curVel.Y + (a1_Y * Titr);
            curVel.Z = curVel.Z + (a1_Z * Titr);

            incPos.X = incPos.X + (curVel.X * Titr);
            incPos.Y = incPos.Y + (curVel.Y * Titr);
            incPos.Z = incPos.Z + (curVel.Z * Titr);
        }
        else if(itr > J1 && manual_stop == false)
        {
            incPos.X = incPos.X + (curVel.X * Titr);
            incPos.Y = incPos.Y + (curVel.Y * Titr);
            incPos.Z = incPos.Z + (curVel.Z * Titr);
        }

        rouVel.X = Rou(curVel.X);
```

```
        rouVel.Y = Rou(curVel.Y);
        rouVel.Z = Rou(curVel.Z);

        if (manual_stop == true && rouVel.X == 0.0 && rouVel.Y == 0.0 &&
rouVel.Z == 0.0){ runTimer = false; manual_stop = false; manual_active =
false; }
        itr ++;
    }
}
    //WriteFile(itr, tmp_posX, tmp_posY, tmp_posZ, velX, velY, velZ);
    WriteSim(absPos.X+incPos.X, absPos.Y+incPos.Y, absPos.Z+incPos.Z,
curVel.X, curVel.Y, curVel.Z);
}

void Thread_1::ResetCordianteSys()
{
    absPos.X = absPos.X + incPos.X;
    absPos.Y = absPos.Y + incPos.Y;
    absPos.Z = absPos.Z + incPos.Z;

    incPos.X = 0.0f;
    incPos.Y = 0.0f;
    incPos.Z = 0.0f;
}
/*
void Thread_1::WriteFile(float timer, float abs_posX, float abs_posY, float
abs_posZ, float velX, float velY, float velZ)
{
    QString text;

    QFile file("files/interpolator.txt");
    if(!file.open(QFile::Append | QFile::Text))
    {qDebug() << "File can't open";}
    QTextStream out(&file);

    text = QString::number(timer) + "\t" + QString::number(abs_posX) + "\t"
+ QString::number(abs_posY) + "\t" + QString::number(abs_posZ) + "\t" +
QString::number(velX) + "\t" + QString::number(velY) + "\t" +
QString::number(velZ) + "\n";
    out << text;

    file.flush();
    file.close();
}
*/
void Thread_1::WriteSim(float posX, float posY, float posZ, float velX,
float velY, float velZ)
{
    //while(thread_2.isRunning()){}
    thread_2.SetPosition(posX, velX, itr);
    thread_2.start();

    //emit SendStatus(Rou(-thread_2.read_posX-abs_posX ),
Rou(thread_2.read_posY-abs_posY), Rou(thread_2.read_posZ-abs_posZ),
```

```
Rou(sqrt(pow(velX, 2) + pow(velY, 2) + pow(velZ, 2))) );
    emit SendStatus(Rou(absPos.X+incPos.X), Rou(absPos.Y+incPos.Y),
Rou(absPos.Z+incPos.Z), Rou(sqrt(pow(velX, 2) + pow(velY, 2) + pow(velZ,
2))) );
}

float Thread_1::Rou(float num) // round on 2 decimal
{
    return round( num * 100 + 0.05 ) /  100;
    //return round( num * 1000 + 0.005 ) /  1000;
}
```

Drugi *thread* vezan je za komunikaciju sa simulacijskim programom

```
#ifndef THREAD_2_H
#define THREAD_2_H

#include <QObject>
#include <QtCore>
#include <QDebug>
#include <QtGlobal>
#include "extApi.h"
#include <QElapsedTimer>
#include <QString>
#include <QFile>
#include <QTextStream>

class Thread_2 : public QThread
{
    Q_OBJECT

public:
    Thread_2(QObject *parent = nullptr);
    ~Thread_2();

    void run();
    void Inicialization();
    void SetPosition(float posX, float posY, float posZ);
    void SetVelocity();
    void WriteFile();

    // vrep variables
    int clientID;
    int vrep_time;

    int X_handle;
    int Y_handle;
    int Z_handle;
```

```
    float velX;
    float velY;
    float velZ;

    int time_step = 0;

    float posX[3];
    float posY[3];
    float posZ[3];

    float read_posX;
    float read_posY;
    float read_posZ;

    float error_X;
    float error_Y;
    float error_Z;

    float last_error_X;
    float last_error_Y;
    float last_error_Z;

    // PID parametars
    float P;
    float I;
    float D;

    float dt;
    float PX;
    float IX;
    float DX;
    float PY;
    float IY;
    float DY;
    float PZ;
    float IZ;
    float DZ;


    // write in file
    QString text;
    QFile file;
    QTextStream out;
};

#endif // THREAD_2_H
```

```
#include "thread_2.h"

Thread_2::Thread_2(QObject *parent)
    : QThread{parent}
{
    posX[0] = 0.0f;
```

```
    posX[1] = 0.0f;
    posX[2] = 0.63f;

    posY[0] = -0.5445f;
    posY[1] = 0.0f;
    posY[2] = 1.365f;

    posZ[0] = -0.284f;
    posZ[1] = 0.0f;
    posZ[2] = 1.1f;

    time_step = 0;

    error_X = 0.0f;
    error_Y = 0.0f;
    error_Z = 0.0f;
    last_error_X = 0.0f;
    last_error_Y = 0.0f;
    last_error_Z = 0.0f;

    velX = 0.0f;
    velY = 0.0f;
    velZ = 0.0f;
    // PID parametars
    dt = 0.001f;

    P = 0.05f;
    I = 0.002f;
    D = 0.002f;

    PX = 0.0f;
    IX = 0.0f;
    DX = 0.0f;
    PY = 0.0f;
    IY = 0.0f;
    DY = 0.0f;
    PZ = 0.0f;
    IZ = 0.0f;
    DZ = 0.0f;


    // write in file
    file.setFileName("files/interpolator.txt");
    if(!file.open(QFile::Append | QFile::Text))
    {qDebug() << "File can't open";}
    out.setDevice(&file);

}
Thread_2::~Thread_2()
{
    file.flush();
    file.close();

    simxStopSimulation(clientID, simx_opmode_blocking);
```

```
    simxFinish(-1);
}

void Thread_2::Inicialization()
{
    simxFinish(-1);
    clientID = simxStart("127.0.0.1", 19997, true, true, 5000, 5);

    if(clientID != -1){
        qDebug() << "connected to remote API server";
    }
    else{
        qDebug() << "connection not sucessful";
        return;
    }

    //simxSynchronous(clientID, true);
    simxStartSimulation(clientID, simx_opmode_oneshot);

    //simxSynchronousTrigger(clientID);
    //simxGetPingTime(clientID, &vrep_time);

    //object handle
    simxGetObjectHandle(clientID, "X_axis", &X_handle,
simx_opmode_oneshot_wait);
    simxGetObjectHandle(clientID, "Y_axis", &Y_handle,
simx_opmode_oneshot_wait);
    simxGetObjectHandle(clientID, "Z_axis", &Z_handle,
simx_opmode_oneshot_wait);

    simxGetJointPosition(clientID, X_handle, &read_posX,
simx_opmode_streaming);
    simxGetJointPosition(clientID, Y_handle, &read_posY,
simx_opmode_streaming);
    simxGetJointPosition(clientID, Z_handle, &read_posZ,
simx_opmode_streaming);
}

void Thread_2::run()
{
    //elapsed_time.start();

    simxSetObjectPosition(clientID, X_handle, -1, &posX[0],
simx_opmode_oneshot);
    simxSetObjectPosition(clientID, Y_handle, -1, &posY[0],
simx_opmode_oneshot);
    simxSetObjectPosition(clientID, Z_handle, -1, &posZ[0],
simx_opmode_oneshot);

/*
    read_posX = read_posX * 1000.0f;
    read_posY = read_posY * 1000.0f;
    read_posZ = read_posZ * 1000.0f;
```

```
    simxGetJointPosition(clientID, X_handle, &read_posX,
simx_opmode_buffer);
    simxGetJointPosition(clientID, Y_handle, &read_posY,
simx_opmode_buffer);
    simxGetJointPosition(clientID, Z_handle, &read_posZ,
simx_opmode_buffer);

    // PID control
    // X axis
    error_X = posX - read_posX;

    PX = P * error_X;
    IX = IX + (I * error_X * dt);
    DX =  (D * (error_X - last_error_X)) / dt;

    velX = PX + IX + DX;
    // Y axis
    error_Y = posY - read_posY;

    PY = P * error_Y;
    IY = IY + (I * error_Y * dt);
    DY =  (D * (error_Y - last_error_Y)) / dt;

    velY = PY + IY + DY;
    // Z axis
    error_Z = posZ - read_posZ;

    PZ = P * error_Z;
    IZ = IZ + (I * error_Z * dt);
    DZ =  (D * (error_Z - last_error_Z)) / dt;

    velZ = PZ + IZ + DZ;

    // set velocity
    simxSetJointTargetVelocity(clientID, X_handle, velX,
simx_opmode_oneshot);
    simxSetJointTargetVelocity(clientID, Y_handle, velY,
simx_opmode_oneshot);
    simxSetJointTargetVelocity(clientID, Z_handle, velZ,
simx_opmode_oneshot);

    last_error_X = error_X;
    last_error_Y = error_Y;
    last_error_Z = error_Z;
*/

    //simxSynchronousTrigger(clientID);
    //simxGetPingTime(clientID, &vrep_time);
    WriteFile();
    //qDebug() << elapsed_time.elapsed();
    //qDebug() << error_X << " " << error_Y << " " << error_Z;
}

void Thread_2::SetPosition(float X, float Y, float Z)
```

```
{
    posX[0] = X; //-(X / 1000.0f);
    posY[1] = Y; //Y / 1000.0f;
    //posZ[1] = Y / 1000.0f;
    posZ[2] = Z; //(Z / 1000.0f) + 1.1f;

    //simxSetObjectPosition(clientID, pen_handle, -1, &pen_position[0],
simx_opmode_blocking);
}

void Thread_2::WriteFile()
{
    text = QString::number(posX[0]) + "\t" + QString::number(posY[1]) + "\t"
+ QString::number(posZ[2]) + "\n";

    //text = QString::number(posX[0]) + "\t" + QString::number(posY[1]) +
"\t" + QString::number(posZ[2]) + "\n";
    out << text;
}
```

Programska *scripta* iz CoppeliaSim za generiranje putanje alata

```
function handleUI(d)
    local s=sim.getObjectSel()
end
function sysCall_init()
    --model=sim.getObject('.')
    sensor = sim.getObject(':/Y_axis/Z_axis/pen/sensor')
    tabl = sim.getObject(':/X_axis')

    local color={1,0,0}

table_draw1=sim.addDrawingObject(sim.drawing_cyclic+sim.drawing_painttag+sim
.drawing_lines,2,0,tabl,99999,color)

    mill_pos = sim.getObjectPosition(sensor, -1)
    table_pos = sim.getObjectPosition(tabl, -1)
```

```
    ptm1 = { mill_pos[1], mill_pos[2], mill_pos[3] }

    ptm2 = ptm1

    pos1 = table_pos[1]
    pos2 = pos1

end

function sysCall_sensing()

    mill_pos = sim.getObjectPosition(sensor, -1)
    table_pos = sim.getObjectPosition(tabl, -1)
    ptm1 = { mill_pos[1], mill_pos[2], mill_pos[3] }
    pos1 = table_pos[1]

    if pos1 == pos2 then
        sim.addDrawingObjectItem(   table_draw1,{   ptm1[1],ptm1[2],ptm1[3],
ptm1[1],ptm2[2],ptm2[3] } )
    else
        sim.addDrawingObjectItem(   table_draw1,{   ptm1[1],ptm1[2],ptm1[3],
ptm1[1] + (pos1-pos2),ptm2[2],ptm2[3] } )
    end

    ptm2 = ptm1
    pos2 = pos1

    --sim.addStatusbarMessage(mill_pos[1]-table_pos[1])

end

function sysCall_actuation()

end

-- sensor x-pos -0.149
```

Izjava o autorstvu

Sveučilište Sjever

SVEUČILIŠTE SJEVER

IZJAVA O AUTORSTVU

Završni/diplomski rad isključivo je autorsko djelo studenta koji je isti izradio te student odgovara za istinitost, izvornost i ispravnost teksta rada. U radu se ne smiju koristiti dijelovi tuđih radova (knjiga, članaka, doktorskih disertacija, magistarskih radova, izvora s interneta, i drugih izvora) bez navođenja izvora i autora navedenih radova. Svi dijelovi tuđih radova moraju biti pravilno navedeni i citirani. Dijelovi tuđih radova koji nisu pravilno citirani, smatraju se plagijatom, odnosno nezakonitim prisvajanjem tuđeg znanstvenog ili stručnoga rada. Sukladno navedenom studenti su dužni potpisati izjavu o autorstvu rada.

Ja, Krunoslav Kušec *(ime i prezime)* pod punom moralnom, materijalnom i kaznenom odgovornošću, izjavljujem da sam isključivi autor/ica završnog/diplomskog (*obrisati nepotrebno*) rada pod naslovom Razvoj upravljačke jedinice za3-osnu glodalicu *(upisati naslov)* te da u navedenom radu nisu na nedozvoljeni način (bez pravilnog citiranja) korišteni dijelovi tuđih radova.

Student/ica:
(upisati ime i prezime)
Kušec
(vlastoručni potpis)

Sukladno čl. 83. Zakonu o znanstvenoj djelatnost i visokom obrazovanju završne/diplomske radove sveučilišta su dužna trajno objaviti na javnoj internetskoj bazi sveučilišne knjižnice u sastavu sveučilišta te kopirati u javnu internetsku bazu završnih/diplomskih radova Nacionalne i sveučilišne knjižnice. Završni radovi istovrsnih umjetničkih studija koji se realiziraju kroz umjetnička ostvarenja objavljuju se na odgovarajući način.

Sukladno čl. 111. Zakona o autorskom pravu i srodnim pravima student se ne može protiviti da se njegov završni rad stvoren na bilo kojem studiju na visokom učilištu učini dostupnim javnosti na odgovarajućoj javnoj mrežnoj bazi sveučilišne knjižnice, knjižnice sastavnice sveučilišta, knjižnice veleučilišta ili visoke škole i/ili na javnoj mrežnoj bazi završnih radova Nacionalne i sveučilišne knjižnice, sukladno zakonu kojim se uređuje znanstvena i umjetnička djelatnost i visoko obrazovanje.