UNIVERZA V LJUBLJANI

FAKULTETA ZA RAČUNALNIŠTVO IN INFORMATIKO

Tadej Humar

KONTROLNI SISTEM ZA KRMILJENJE MOTORJEV IN KOREKCIJSKIH TULJAV

DIPLOMSKO DELO

NA UNIVERZITETNEM ŠTUDIJU

Mentor: izr. prof. dr. Patricio Bulić

Ljubljana, 2013

Univerza *v Ljubljani*
Fakulteta *za računalništvo in informatiko*

Tržaška 25
1001 Ljubljana, Slovenija
telefon: 01 476 84 11
01 476 83 87
faks: 01 426 46 47
01 476 87 11
www.fri.uni-lj.si
e-mail: dekanat@fri.uni-lj.si

Št. naloge: 01903/2013
Datum: 04.02.2013

Univerza v Ljubljani, Fakulteta za računalništvo in informatiko izdaja naslednjo nalogo:

Kandidat: **TADEJ HUMAR**

Naslov: **KONTROLNI SISTEM ZA KRMILJENJE MOTORJEV IN KOREKCIJSKIH TULJAV**

A CONTROL SYSTEM FOR MOTORS AND CORRECTION COILS

Vrsta naloge: Diplomsko delo univerzitetnega študija

Tematika naloge:

Kontrolni sistem TANGO je odprtokodni porazdeljen objektno orientirani kontrolni sistem, ki se večinoma uporablja za nadzor pospeševalnikov. Undulator je naprava z alternirajočim zaporedjem zelo močnih dipolnih magnetov, ki vzdolž undulatorja ustvarijo alternirajoče statično magnetno polje. Magnete premikamo s pomočjo motorjev. Razvijte kontrolni sistem in srežnik TANGO, ki omogoča popoln nadzor nad motorji v undulatorju in korekcijskimi tuljavami. Kontrolnemu sistemu dodajte grafični uporabniški vmesnik, ki uporabniku olajša nadzor nad napravami.

Mentor:
izr. prof. dr.Patricio Bulić

Dekan:
prof. dr. Nikolaj Zimic

I Z J A V A O A V T O R S T V U

diplomskega dela

Spodaj podpisan Tadej Humar,

z vpisno številko 63050046,

sem avtor diplomskega dela z naslovom:

Kontrolni sistem za krmiljenje motorjev in korekcijskih tuljav

S svojim podpisom zagotavljam, da:

- sem diplomsko delo izdelal/-a samostojno pod mentorstvom izr. prof. dr. Patricia Bulića
- so elektronska oblika diplomskega dela, naslov (slov., angl.), povzetek (slov., angl.) ter ključne besede (slov., angl.) identični s tiskano obliko diplomskega dela
- soglašam z javno objavo elektronske oblike diplomskega dela v zbirki »Dela FRI«

V Ljubljani, dne 24.3.2013 Podpis avtorja: Tadej Humar

Zahvala

Za pomoč pri izdelavi diplomske naloge se zahvaljujem mentorju izr. prof. dr. Patriciu Buliću, podjetju Cosylab d.d. in Franku Amandu za izkazano zaupanje ter možnost sodelovanja na projektu.

Posebno zahvalo si zasluži Miroslav Pavleski, ki me je vodil, učil in svetoval skozi vse faze izdelave projekta.

Zahvalil bi se tudi staršem za podporo in potrpežljivost tekom celotnega študija.

Nedavno preminulemu Ivanu Bitežniku.

Kazalo

Seznam uporabljenih kratic in simbolov

CORBA - Common Object Request Broker Architecture

DMC – Digital Motion Controller

EPICS - Experimental Physics and Industrial Control System

ESRF - European Synchrotron Radiation Facility

IOR - Interoperable Object Reference

OPC - Open Process Control

PID - Proportional Integral Derivative

PLC - Programmable logic controller

POGO - Program Obviously used to Generate tango Object

TANGO - TAco Next Generation Objects

VM – Virtual machine

Povzetek

Diplomska naloga obravnava izdelavo kontrolnega sistema TANGO za undulator in korekcijske tuljave. V prvem delu predstavi kako kontrolni sistem TANGO izgleda in deluje ter orodja za delo z njim. V drugem delu na kratko opiše zgradbo undulatorja in zahtevanih funkcionalnosti. Nato nadaljuje s predstavitvijo celotne arhitekture kontrolnega sistema. Sledi natančnejša predstavitev generične in za undulator specifične mikrokode, napisane za napravo Galil DMC 4080. Naslednje poglavje predstavi funkcionalnosti in potrebno konfiguracijo strežnikov ControlBox in OPCaccess. V istem poglavju predstavi tudi razviti strežnik Undulator z vsemi potrebnimi funkcionalnostmi in natančneje opiše njegovo konfiguracijo ter uporabo. Zadnje poglavje predstavi grafični vmesnik in postopek konfiguracije le tega.

Ključne besede: kontrolni sistem, undulator, korekcijske tuljave, strežniki TANGO

Abstract

The thesis presents the implementation of a TANGO control system for undulator and correction coils. In the first part it presents how TANGO control system looks like, its usage and tools to work with it. Second part has a short description of undulator construction and required functionalities. It continues with a presentation of complete control system architecture followed by a more detailed presentation of generic and undulator specific Galil DMC 4080 microcode. Next chapter presents functionalities and required configuration of ControlBox and OPCaccess TANGO servers. In the same chapter it presents the developed Undulator TANGO server with all required functionalities and describes in detail its configuration and usage. Last chapter presents graphical user interface and describes its configuration.

Keywords: control system, undulator, correction coils, TANGO servers

1 Uvod

Undulator je ena ključnih naprav pri eksperimentih, ki se izvajajo v sinhrotronih. Le ti se precej širijo in nadgrajujejo, da bi zadostovali potrebam dandanašnje znanosti. Seveda to pomeni, da se veča tudi potreba po bolj zahtevnih in specifičnih kontrolnih sistemih. V ta namen so se razvili različni kontrolni sistemi namenjeni prav za potrebe znanstvenih inštitucij. Po svetu je najbolj razširjen EPICS (*Experimental Physics and Industrial Control System*), vendar se zadnje čase predvsem v Evropi širi kontrolni sistem TANGO (*TAco Next Generation Objects*) [1, 2].

Vsaka naprava potrebuje svoj kontrolni sistem in po potrebi se posamezni kontrolni sistemi lahko povezujejo v večjo celoto.

Cilj diplomske naloge je bil napisati kontrolni sistem za undulator in korekcijske tuljave, ki so potrebne za pravilno delovanje le tega. Uporabljali ga bodo v Švedskem laboratoriju MAX-Lab, ki je član kolaboracije TANGO, zato je bila izbira kontrolnega sistema samoumevna [2].

Sistem bo sestavljen iz treh delov.

Prvi del je sestavljen iz kode za premikanje s pomočjo dveh naprav Galil DMC-4080 ter sinhronizacije med njima prek digitalnih vhodov in izhodov. Undulator bi potreboval samo eno napravo DMC za delovanje, a se že načrtuje nadgradnja sistema, ki bo zahtevala sinhronizacijo undulatorja z monokromatorjem.

Drugi del zahteva razvoj strežnika TANGO za naprave. Uporabili bomo nekatere funkcionalnosti že razvitega strežnika ControlBox, ki zna komunicirati z napravo Galil DMC (Digital Motion Controller) in izvajati enostavne premike motorjev [3]. Strežnik so razvili pri SOLEIL (Francoski nacionalni sinhrotron) in ga kot je v navadi v skupnosti TANGO, ponujajo v odprti kodi na njihovi uradni strani [4].

Tretji del kontrolnega sistema predstavlja uporabniški grafični vmesnik, ki končnemu uporabniku olajša nadzor nad napravo. Seveda mora vmesnik podpirati tudi napredne funkcionalnosti, ki jih bodo potrebovali inženirji za vzdrževanje in morebitno odpravljanje napak.

2 TANGO

Za lažje razumevanje diplomske naloge poglavje predstavlja način razvijanja kontrolnega sistema s pomočjo knjižnice TANGO in delovanje le tega.

2.1 Kaj je TANGO?

Kontrolni sistem TANGO so začeli razvijati na ESRF (*European Synchrotron Radiation Facility*) v Franciji za potrebe po razvoju boljših in poenotenih kontrolnih sistemov njihovega sinhrotrona [5]. Sčasoma so se jim pridružile še druge evropske organizacije in kolaboracija trenutno šteje osem polnopravnih članic. Obstaja pa tudi pet podjetij v gospodarstvu, ki razvijajo kontrolne sisteme TANGO [2].

TANGO je distribuiran kontrolni sistem, kar pomeni, da lahko delujejo vsi procesi (strežniki za naprave, podatkovna baza, prenašalci sporočil, …) na enem glavnem strežniku ali vsak na svojem strežniku. Razvijalcem ponuja objektno orientirano programiranje kontrolnih sistemov, ki so zmogljivi, vendar enostavni za načrtovanje in vzdrževanje. Poslužuje se standarda CORBA (*Common Object Request Broker Architecture*), a implementacijo le tega pred programerjem skrije. Primarno se ga uporablja za priključitev naprav prek omrežja Ethernet. Omogoča uporabo sinhrone, asinhrone in dogodkovne komunikacije [2, 6].

Za shranjevanje trajnih podatkov, kot so imena strežnikov, konfiguracije, lastnosti in podobno uporablja podatkovno bazo MySQL. Sam projekt je odprtokoden in tudi strežnike za različne naprave je mogoče dobiti na uradnih straneh posameznih inštitucij. Omeniti je potrebno, da deluje na operacijskih sistemih Linux, Windows in Solaris [2].

2.2 Podprti programski jeziki

Strežnike TANGO lahko razvijamo v različnih programskih jezikih. Načeloma je najbolj podprt in razširjen C++, a za naprave, ki niso preveč zahtevne oziroma ne potrebujejo veliko procesiranja, se uporabljata tudi Java in Python. Obstajajo dodatni vtiči za razvijalna okolja Matlab, LabVIEW in Igor [2].

Po naših dosedanjih izkušnjah se strežnik programira v jeziku C++, odjemalec pa v jeziku Python. Slednji omogoča hiter razvoj uporabniškega vmesnika predvsem zaradi dobrih knjižnic PyTango, Taurus in Sardana. Programski jezik C++ pa se uporablja predvsem zaradi hitrosti in nadzora nad podrobnostmi, ki ga omogoča programerju.

2.3 Delovanje kontrolnega sistema TANGO

Kontrolni sistem TANGO ima več nivojev, ki jih je potrebno poznati.

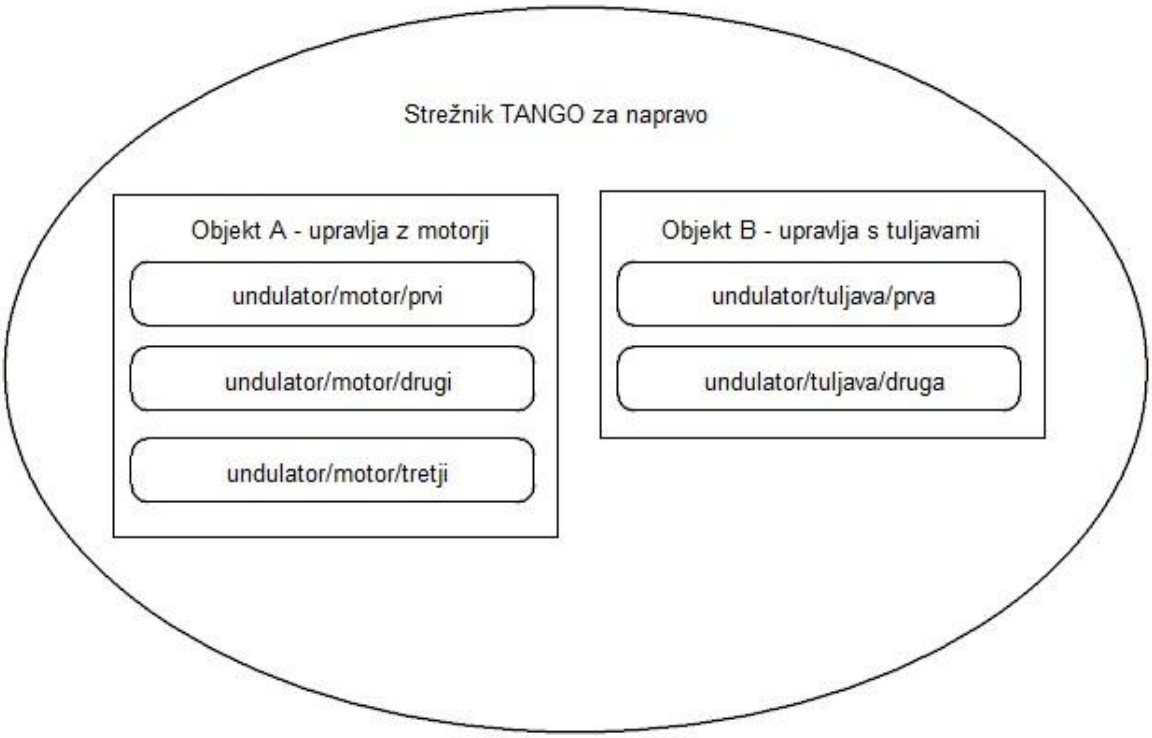
Na najvišjem nivoju imamo podatkovno bazo MySQL in strežnik TANGO DataBaseds, ki skrbi za komunikacijo s podatkovno bazo. Poleg shranjevanja podatkov, ima podatkovna baza

še dodatno nalogo, da posreduje oznake IOR (*Interoperable Object Reference*), ki nam služijo kot reference na instance objektov znotraj oddaljenega strežnika CORBA [7].

Naslednja stopnja je strežnik za napravo. To je strežnik, napisan z uporabo knjižnice TANGO. Vsak strežnik vsebuje vsaj en objekt, ki zna ravnati z določeno napravo. Primer strežnika, ki zna upravljati z motorjem in tuljavo, lahko vidimo na sliki 1.

To bi bilo dovolj v primeru, da bi vedno imeli samo eno napravo na objekt. Ker pa hočemo nadzorovati več istih naprav prek enega procesa oziroma strežnika, potrebujemo instance objektov.

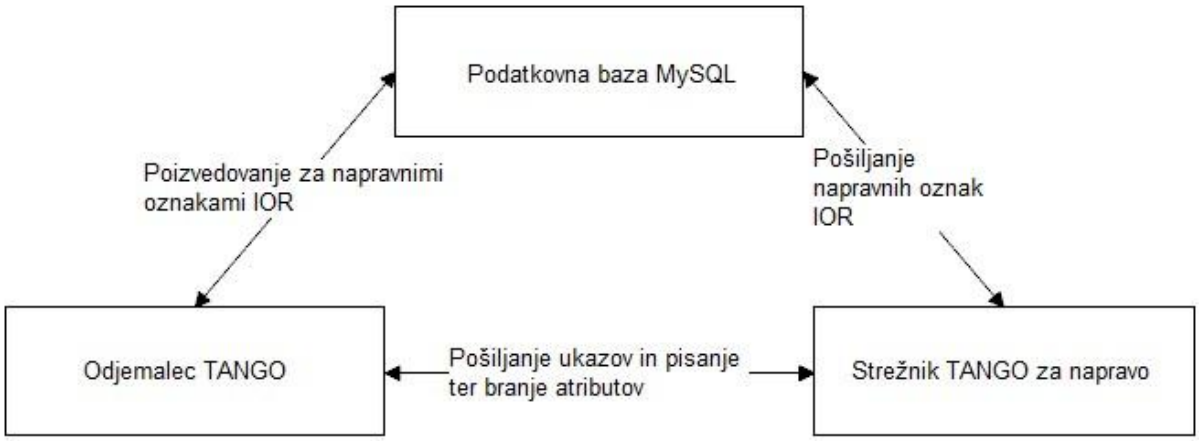
Instance objektov so tretji nivo kontrolnega sistema. Vsaka instanca predstavlja točno določeno napravo in ima ime sestavljeno iz treh delov (domena/družina/član). S tem postane sistem bolj pregleden in enostaven za vzdrževanje. Vsaka instanca ima svoja stanja, atribute in ukaze, prek katerih nadzorujemo oziroma spremljamo delovanje naprave.



Slika 1: Strežnik TANGO za napravo z dvema objektoma in petimi instancami [8]

Kot zadnji nivo lahko štejemo odjemalca, ki omogoča nadzor nad napravo. Iz podatkovne baze prebere referenco IOR, prek katere nato pošilja ukaze, piše ali bere atribute in spremlja trenutno stanje ter sporočilo strežnika.

Minimalen kontrolni sistem TANGO z enim odjemalcem in strežnikom je prikazan na sliki 2.



Slika 2: Izgled minimalnega kontrolnega sistema TANGO [8]

2.4 Orodja za razvoj

V tem podpoglavju bomo predstavili najbolj uporabljena orodja za razvoj in konfiguracijo strežnikov TANGO.

2.4.1 POGO (Program Obviously used to Generate tango Object)

Kljub dobrim knjižnicam bi za razvoj brez orodij potrebovali veliko časa. Da se izognemo vedno ponovnemu pisanju ogrodja strežnika, imamo na voljo program POGO. To je zagotovo najbolj poznano orodje v skupnosti TANGO. Omogoča nam grafično določanje atributov, ukazov, stanj in lastnosti strežnika. Ko določimo vse potrebno, nam POGO ustvari generični del kode bodisi v programskem jeziku C++, Javi ali Python [2].

Več o funkcionalnostih in izgledu programa bo opisano v poglavju razvoja strežnika TANGO za undulator.

2.4.2 JIVE

S pomočjo programa POGO smo razvili strežnik, ki ga je potrebno pred zagonom, prvo zapisati v bazo in izpolniti lastnosti. Za to nam je na voljo program JIVE, ki nam omogoča grafično urejanje podatkovne baze.

Podpira vstavljanje zapisov novih strežnikov za naprave, dodajanje instanc objektov in določanje lastnosti le teh.

Omogoča tudi enostavno testiranje naprav. V primeru, da strežnik TANGO za napravo deluje in je instanca pravilno nastavljena (izpolnjene lastnosti), nam s klikom na ime naprave program JIVE ustvari enostaven grafični vmesnik, ki prikazuje vse atribute izbranega objekta. Prek tega vmesnika lahko testiramo pisanje in branje atributov ter izvajanje ukazov. V fazi razvoja je ta funkcionalnost zelo dobrodošla in precej olajša delo programerju [2].

2.4.3 ATK

Knjižnica ATK omogoča enostaven razvoj grafičnih vmesnikov v Javi. Primer uporabe smo videli pri programu JIVE. Ta koristi prav knjižnico ATK oziroma ATK Panel, za realizacijo enostavnega grafičnega vmesnika [2].

Sam nimam preveč izkušenj z implementacijo s pomočjo knjižnice ATK, saj smo grafični vmesnik raje naredili v jeziku Python.

2.4.4 PyTango, Taurus in Sardana

Skupna točka vsem trem navedenim knjižnicam je ta, da uporabljajo za razvoj jezik Python. Prva omogoča razvoj strežnika in odjemalca (grafični vmesnik), medtem ko sta drugi dve usmerjeni predvsem v razvoj grafičnega vmesnika. Z uporabo katerekoli izmed teh knjižnic, bi lahko razvili dober grafični vmesnik. Za naš kontrolni sistem smo uporabili knjižnico Taurus, saj ima poleg grafičnega načrtovalca (Taurus Qt Designer) tudi izboljšan sistem za delovanje s pomnilnikom. Počasi se uveljavlja tudi knjižnica Sardana in prav ta naj bi se v prihodnosti najbolj uporabljala. Gre se predvsem za standardizacijo, ki bi omogočila lažje izmenjevanje že razvitih aplikacij med inštituti [2]. Sardana je tudi najnovejša in najbolj izpopolnjena, ampak smo se ji morali izogniti, ker na naši verziji strežnika TANGO in ostalih knjižnic ni delovala pravilno.

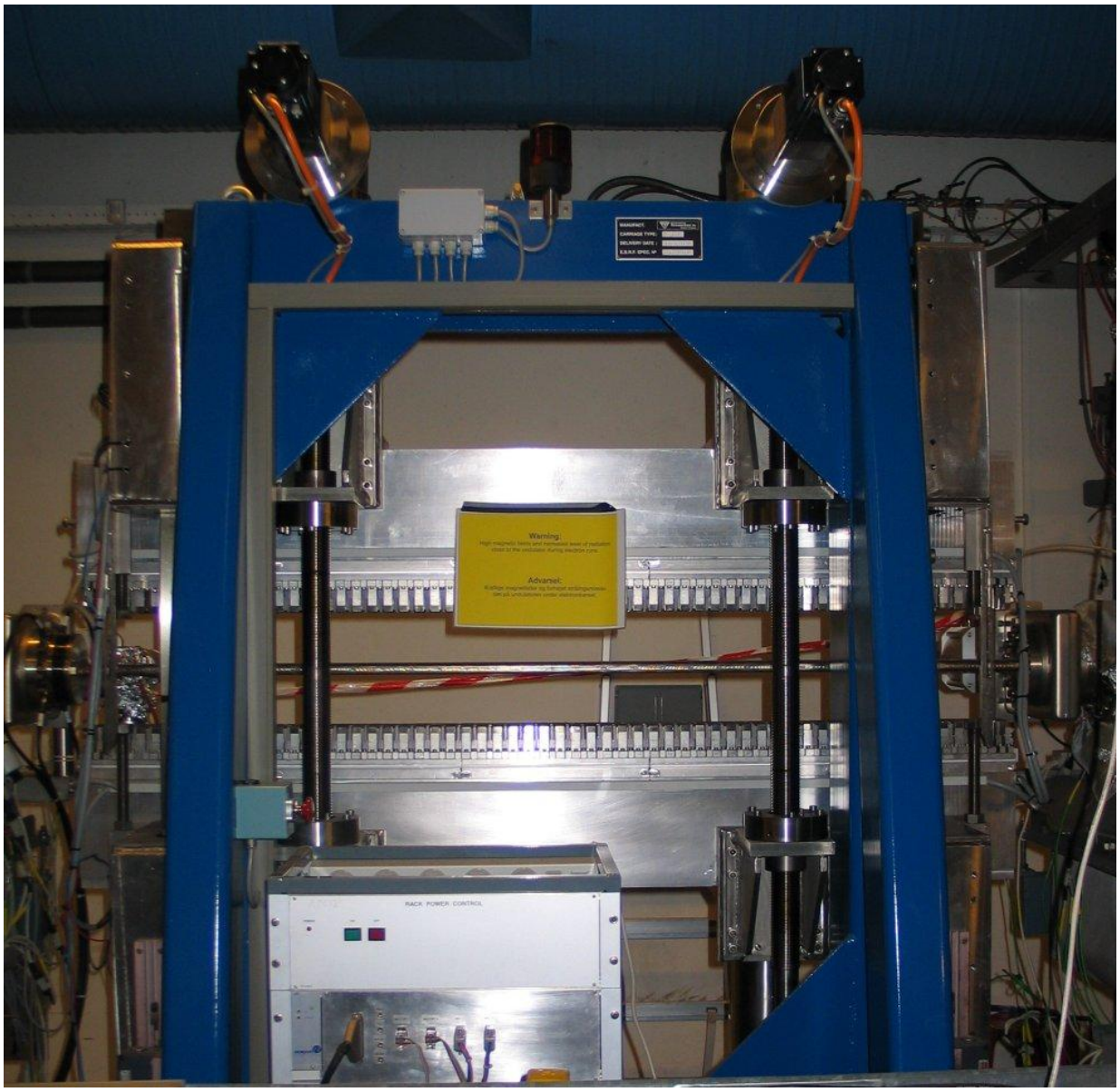
2.4.5 Tango Box in ostala orodja

Do sedaj sem opisal najpomembnejša orodja in knjižnice. Omenil bi še Tango Box, ki sicer ni orodje, gre namreč za virtualno sliko sistema (*VM image*). Uporaba je enostavna. Prenesemo datoteko, jo razširimo in zaženemo s programom VM Player. Vsebuje operacijski sistem Linux, z vsem potrebnim za razvijanje in poganjanje strežnikov TANGO [2].

Več o ostalih orodjih, kot so avtomatski zaganjalnik strežnikov do zbiratelja podatkov, si lahko preberete na uradni strani ali pa jih preizkusite znotraj sistema Tango Box.

3 Undulator

Undulator je naprava z alternirajočim zaporedjem zelo močnih dipolnih magnetov s periodo λ. Ko elektroni potujejo skozi to magnetno polje, začnejo oscilirati in oddajati energijo [9]. To energijo se nato usmeri k različnim raziskovalnim objektom. Enačbe vpliva delovanja magnetnega polja na elektrone za nas niso pomembne.

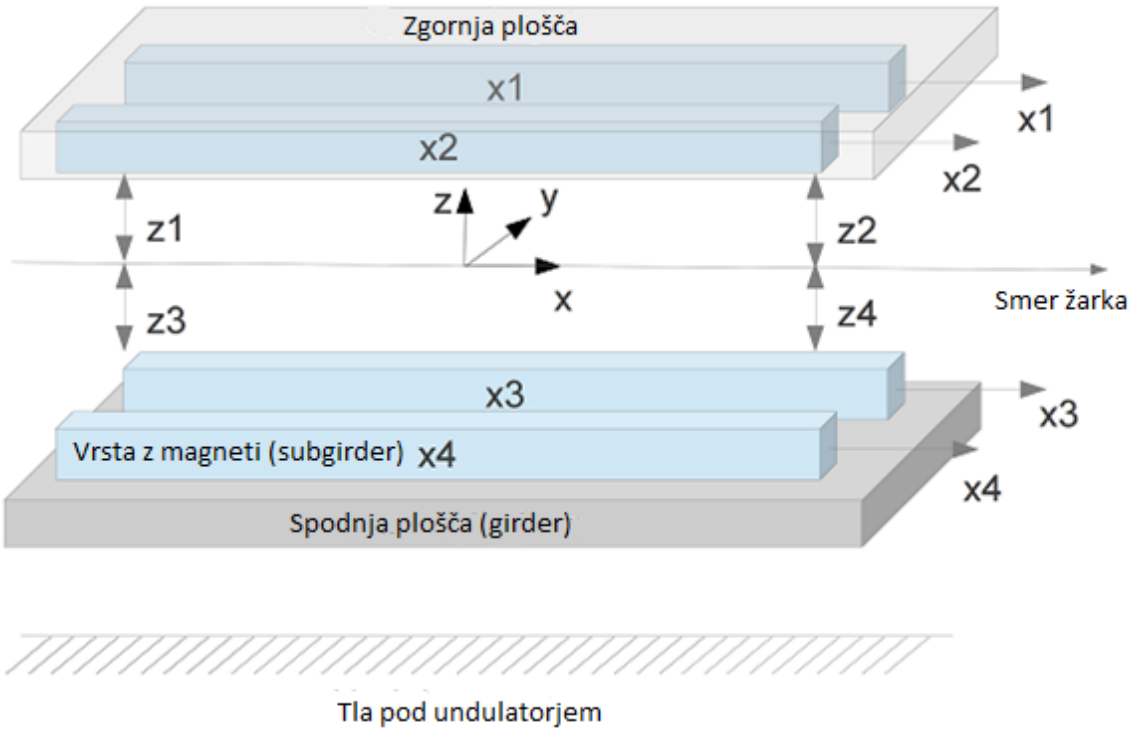


Slika 3: Undulator [10]

3.1 Zgradba

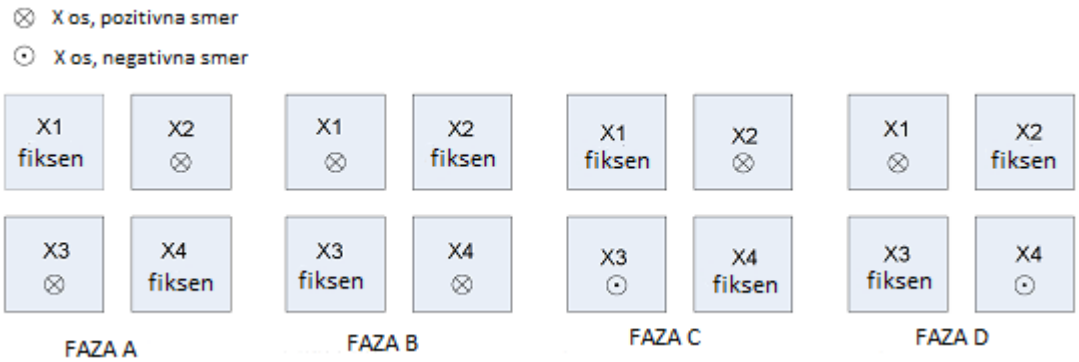
Kot je razvidno iz slike 3 in 4 ima undulator dve plošči (*girder*), ki se premikata po osi Z. S tem nadziramo razmak, ki je eden izmed najpomembnejših parametrov za končnega

uporabnika. Na vsaki plošči imamo po dve vrsti (*subgirder*) magnetov, ki se premikajo v smeri žarka po osi X.



Slika 4: Zgradba undulatorja [11]

Vrsti se lahko premikata v skupno smer ali vsaka v svojo. Vedno se premikata samo po dve vrsti in dve ostaneta fiksni na izhodišču. S tem lahko ustvarimo 4 različne faze, ki so prikazane na sliki 5.

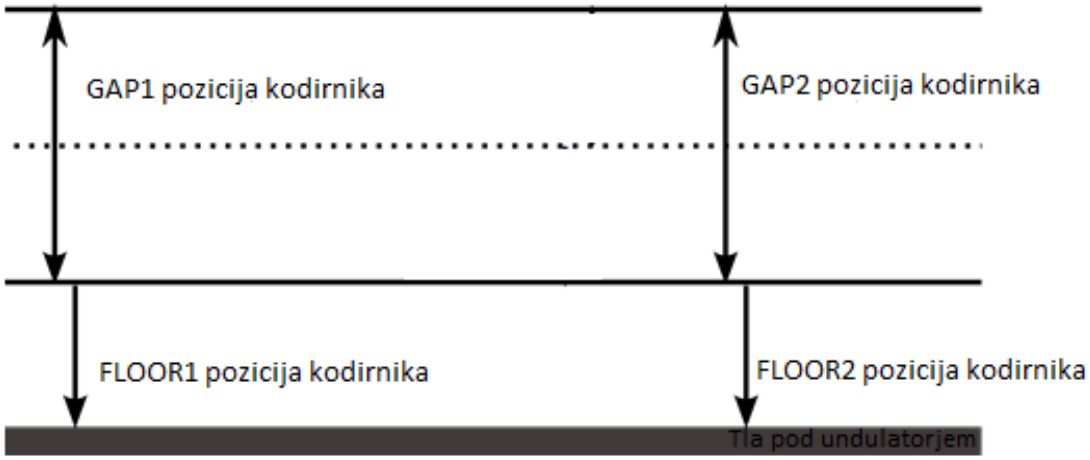


Slika 5: Vse štiri mogoče faze [11]

Izbira faze, fazni odmik (*phase offset*) in razmak so parametri, ki jih uporabnik največkrat spreminja.

3.2 Motorji in kodirniki

Za premikanje vsake plošče po osi Z potrebujemo po dva motorja in vsaka vrsta magnetov ima svoj motor za premikanje po osi X, torej potrebuje undulator za delovanje skupno osem motorjev. Motorje, ki skrbijo za premikanje po osi X, bomo poimenovali Phase (faza), medtem ko bodo motorji, ki premikajo obe plošči po osi Z, poimenovani Gap (razmak). Motorji Phase imajo vsak svoj absoluten linearen kodirnik (*absolute linear encoder*), motorji Gap pa vsak svoj rotacijski kodirnik (*rotary encoder*). Ker pa rotacijski kodirniki niso natančni oziroma potrebujejo redno ponastavitev točnega položaja (*homing*), imamo tudi štiri linearne kodirnike za razmak. Dva sta nameščena med ploščama in merita dejanski razmak (*gap*) ter dva, ki merita razdaljo spodnje plošče od tal (*floor*), kot je prikazano na sliki 6. Z njihovo pomočjo bomo zagotovili, da se bodo po vsakem premikanju rotacijski kodirniki ponastavili na točen položaj. To je potrebno, ker pri undulatorju magnetna sila med ploščama doseže v idealnih pogojih do 25000N, kar ukrivi plošči. Rotacijski kodirniki tega ne bi zaznali in s tem sporočali napačen dejanski razmak [11].



Slika 6: Postavitev kodirnikov za razmak [12]

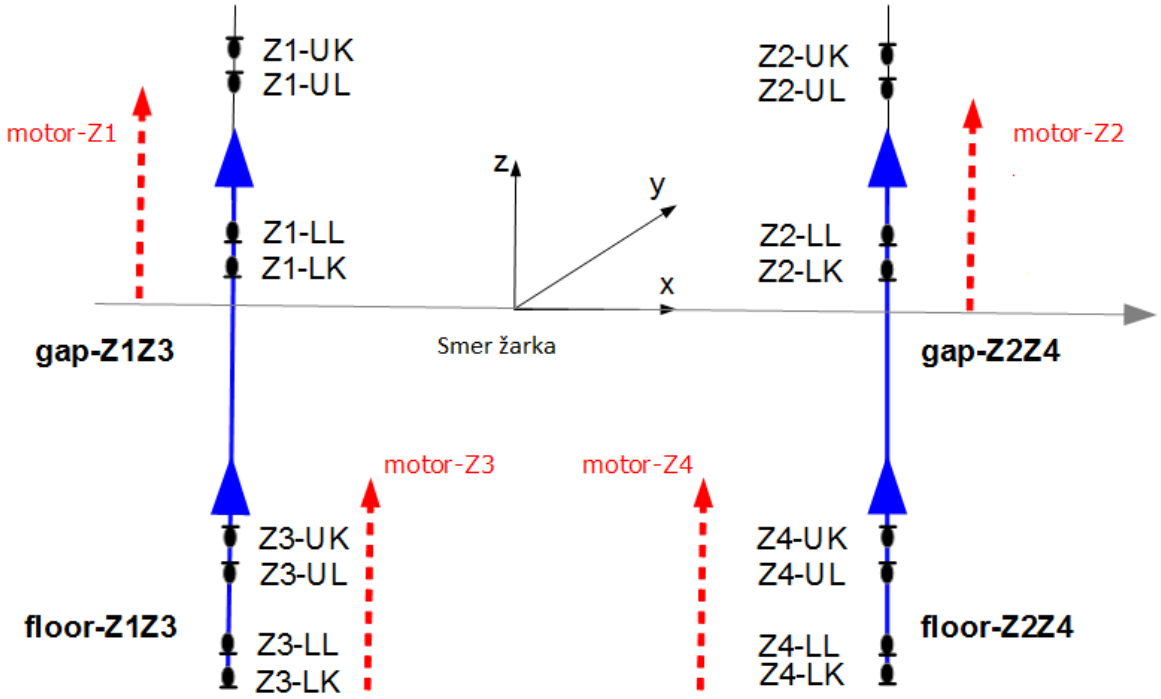
3.3 Stikala za omejevanje premikanja in merilci nagnjenosti

Na vsaki strani vrste imamo dve stikali. Prvo pošlje samo opozorilo napravi DMC, da je motor prekoračil stikalo za mehko omejitev (*soft limit switch*). Za večjo varnost in morebitne napake v programu, imamo še dodatno stikalo takoj za prvim, ki je povezano direktno na napravo PLC (*Programmable logic controller*) in ta v primeru, da dobi signal, takoj »ubije« oziroma ustavi motor. Od tod mu tudi angleško ime *kill limit switch* [11]. Sliki 7 in 8 prikazujeta postavitev kodirnikov in stikal. Z rdečimi puščicami je prikazana pozitivna smer za kodirnike. Kratice uporabljene na naslednjih dveh slikah, so razložene v tabeli 1.

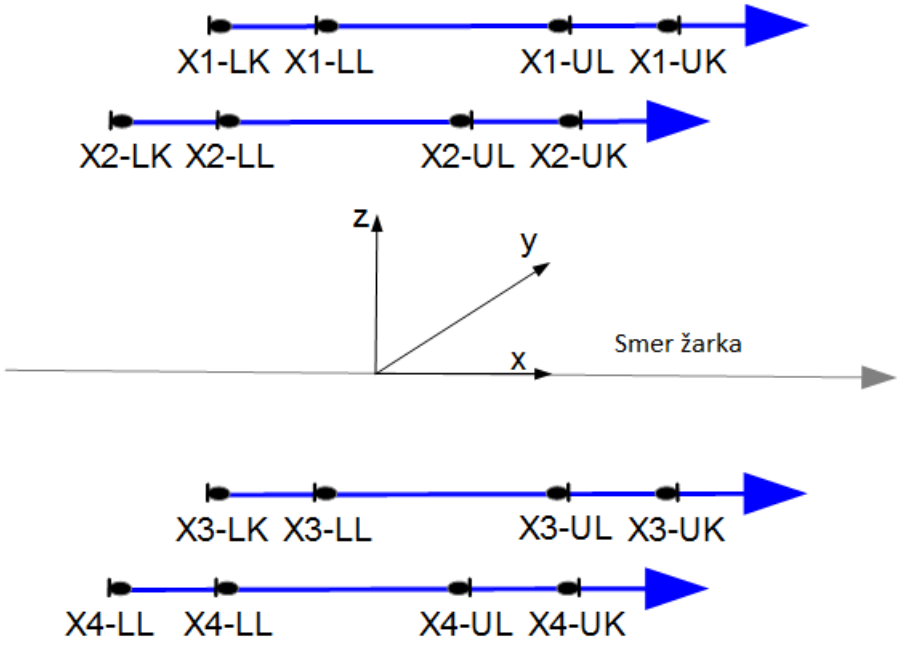
Kratica	Pomen
UL	Zgornje mehko omejitveno stikalo (Prekoračitev premika

	naprej).
LL	Spodnje mehko omejitveno stikalo (Prekoračitev premika nazaj).
UK	Zgornje ubij omejitveno stikalo (Naprava PLC ustavi motor).
LK	Spodnje ubij omejitveno stikalo (Naprava PLC ustavi motor).
Z*, X*	Oznaka in številka motorja glede na os premikanja.

Tabela 1: Razlaga kratic za stikala in motorje



Slika 7: Postavitev kodirnikov in stikal za motorje Gap [11]



Slika 8: Postavitev kodirnikov in stikal za motorje Phase [11]

Problem pri premikanju razmaka je tudi ta, da se lahko eden od motorjev premika hitreje kot drugi. V tem primeru bomo dobili nagnjenost (*tilt*), ki pa ne sme biti prevelika, saj močno vpliva na pot elektronov. Da do tega ne pride, imamo merilce nagnjenosti (*tilt meters*). Ti so povezani direktno na napravo PLC in če je nagnjenost prevelika, naprava PLC sproži tako imenovan signal Interlock.

3.4 Signala in stanji Interlock ter Stop all

Signal Interlock se sproži v primeru, ko je z napravo nekaj narobe. Ne more ga sprožiti uporabnik sam, ampak ga sproži naprava PLC. Zgodi se lahko, da kateri od motorjev ne dela, imamo preveliko nagnjenost, motor gre predaleč in podobno. Ko se signal sproži, se vsi motorji ustavijo. Poleg tega se onemogoči vse nadaljnje ukaze premikanja. Temu rečemo, da je naprava v stanju Interlock. Potrebna je prisotnost inženirja s posebnim ključem (*Interlock key override*), ki lahko prekliče to stanje. Nato s premikanjem posameznih motorjev oziroma potrebnimi popravili, postavi undulator nazaj v normalno delovanje [11].

Signal Stop all deluje podobno kot Interlock, le da ga sproži uporabnik sam. Uporablja se ga zaradi varnosti v primeru, da se nekaj popravlja na napravi, pa ne želimo, da se motorji premikajo. Ko naprava DMC zazna ta signal, postavi sistem v stanje Stop all. Pred naslednjim premikanjem mora uporabnik izključiti ta signal in naprava DMC bo postavila sistem nazaj v normalno delovanje [11].

3.5 Korekcijske tuljave (*correction coils*)

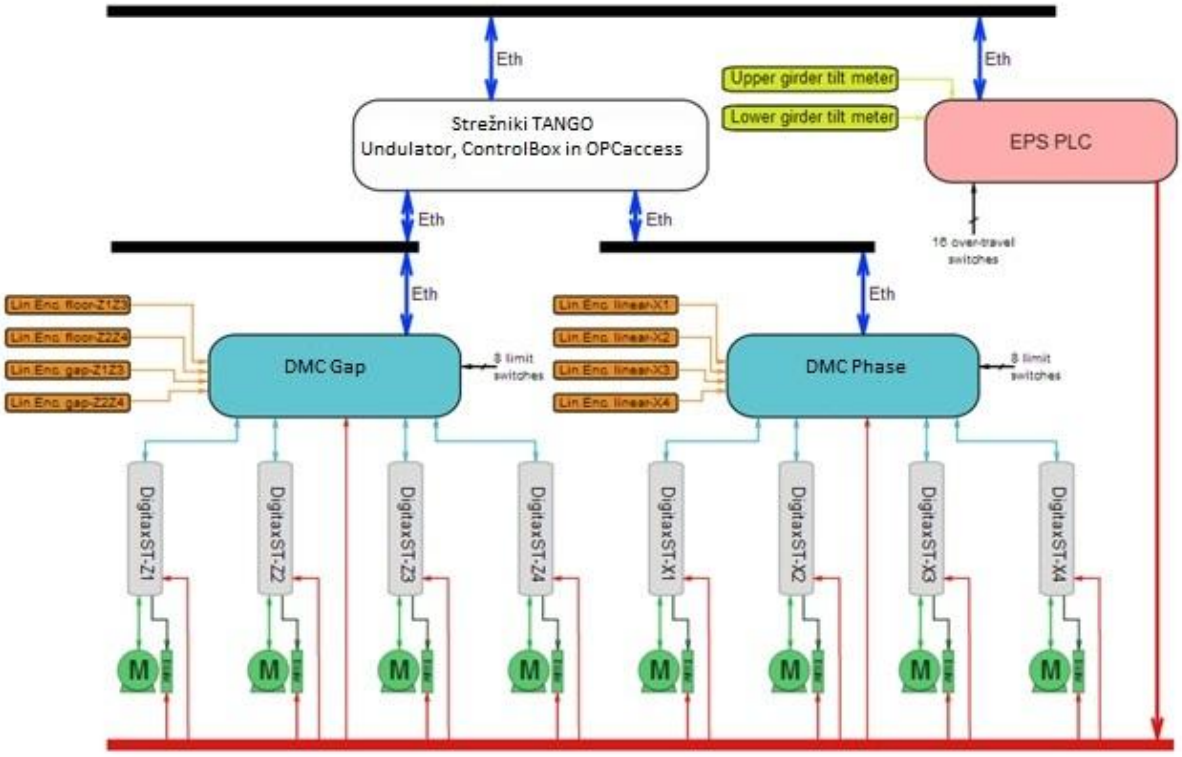
Pred vhodom in po izhodu undulatorja imamo nameščene korekcijske tuljave. Tuljave popravijo napake v magnetnem polju in poravnajo pot elektrona skozi vhod v undulator ter na izhodu zaradi odstopanj od idealnega undulatorja poskrbijo, da elektron nadaljuje po pravi poti.

Za določanje toka na tuljavah ne obstaja formula. Določa se ga glede na trenutni razmak, fazo, fazni odmik, odmik od centra žarka in nagnjenost undulatorja in sicer s poskušanjem. Ko izmerimo in določimo zadostno število točk napravimo tabelo, nad katero kasneje izvajamo logaritemsko interpolacijo za določanje toka v odvisnosti od trenutnega stanja undulatorja [11].

Tuljave se napajajo s posebnim napajalnikom (*power supply*), ki ga nadzira naprava PLC. Ukaze za nastavljanje toka pošljemo na napravo PLC prek strežnika OPC (*Open Process Control*). Za komunikacijo s strežnikom OPC se uporablja poseben strežnik TANGO imenovan OPCaccess [12]. Za potrebe diplomske naloge si lahko predstavljamo, da se pisanje v napravo PLC izvaja kar prek strežnika OPCaccess.

4 Arhitektura sistema

V tem poglavju bodo na grobo predstavljeni vsi deli kontrolnega sistema. Za lažje razumevanje je arhitektura prikazana na sliki 9.



Slika 9: Arhitektura celotnega sistema [11]

4.1 Naprava Galil DMC 4080

Na voljo imamo dve napravi Galil DMC 4080, vsaka ima zmožnost nadzorovanja osem motorjev [13]. Poimenovali smo ju glede na to, katero skupino motorjev bosta nadzorovali. Tako dobimo napravo DMC Gap, ki skrbi za razmak ter napravo DMC Phase, ki skrbi za fazo in fazni odmik. Med seboj sta povezani preko digitalnega vhoda in izhoda. To je potrebno, ker se med premikanjem faze, zaradi vpliva spreminjanja magnetnih sil, spremeni tudi razmak in ga zato sprotno popravljamo. Z napravo DMC komuniciramo prek omrežja Ethernet.

Naloge naprave DMC so naslednje.

- Premikanje posameznih motorjev
- Premikanje spodnje in zgornje plošče
- Omogočanje in onemogočanje premikanja
- Sporočanje o trenutnem dogajanju (premikanje, napake, konfiguracija, pravice, …)

- Simetrično premikanje obeh plošč za dosego želenega razmaka, nagiba in odmika od centra žarka
- Simetrično premikanje vrst z magneti za dosego željene faze in faznega odmika
- Vzdrževanje željenega razmaka
- Računanje trenutnih vrednosti razmaka, nagnjenosti, odmika od centra žarka, faze in faznega odmika

4.2 TANGO strežniki

4.2.1 ControlBox

Kot omenjeno v uvodu, bomo uporabili strežnik ControlBox. Komunicirati zna z napravo Galil DMC 4080. Omogoča premikanje posameznih motorjev in povezovanje dveh motorjev v skupino. Te dve funkcionalnosti bomo uporabili za zadostiti inženirskim potrebam premikanja. Ponuja nam konfiguracijo motorja od hitrosti do tipa in obnašanja. Poleg branja trenutne pozicije motorja, pravilno interpretira njegovo stanje in opozarja uporabnika o dogajanju na napravi DMC. Preko njega lahko pošiljamo direktne ukaze na napravo DMC in prav ta funkcionalnost je bila za nas najbolj pomembna [3].

Tako kot sta napravi DMC ločeni po tem, katero nalogo opravljajo motorji, smo enako ločili tudi dva strežnika ControlBox. Tukaj gre dejansko za ločena strežnika in ne instanco razreda ControlBox. Vsak strežnik ima namreč več instanc objektov za nadzorovanje motora (*GalilAxis)*, objekt nadzorne plošče (*ControlBox)* in odvisno od uporabe, različne instance objektov za skupno premikanje motorjev (*GalilGearedAxes*). Pri nas imamo dva taka primera skupne povezave in sicer gre za zgornjo ter spodnjo ploščo.

Potrebno je poudariti, da je strežnik ControlBox narejen za starejšo različico naprave Galil DMC (serija 2xxx), katerega smo posodobili za pravilno delovanje z napravo Galil DMC 4080. Potrebovali smo kar nekaj časa, preden nam je bilo delovanje jasno. Problem je v tem, da dokumentacija praktično ne obstaja in komentarji mikrokode so v Francoščini. V tem primeru lahko vidimo, koliko je dejansko vredna dobra dokumentacija, saj bi nam prihranila ogromno časa. Ko smo enkrat imeli polno delujoči strežnik ControlBox, smo začeli z načrtovanjem specifičnega strežnika TANGO za undulator.

Strežnik ControlBox je kot večina ostalih strežnikov TANGO prosto dostopen.

4.2.2 Undulator

Naš strežnik TANGO se imenuje Undulator. Ima naslednje naloge.

- Pošiljanje ukazov za premikanje razmaka in faze
- Medsebojno povezovanje obeh strežnikov ControlBox
- Branje pomembnih podatkov iz naprav DMC
- Izvajanje logaritemske interpolacije in računaje potrebnega toka za tuljave

- Pošiljanje in branje podatkov o korekcijskih tuljavah preko strežnika OPCaccess
- Prikazovanje trenutnih vrednosti pomembnih parametrov
- Pošiljanje ukaza za zaustavitev premikanja in onemogočanje premikanja
- Spreminjanje stanja strežnika glede na stanje vseh naprav
- Pošiljanje ukaza za izklop omejitev premikanja
- Omogočiti kalibracijo skupin motorjev
- Konfiguracija motorjev (hitrost in pospeški za Gap, Phase ali posamezne motorje)
- Izvajanje namernega nagiba plošč
- Posebno inženirsko stanje, v katerem se lahko premikajo samo posamezni motorji
- Računanje končne pozicije motorjev Phase za določeno fazo in fazni odmik

4.2.3 Strežnik OPCaccess

Pri nalogah undulatorja je omenjen tudi strežnik OPCaccess. To je strežnik TANGO, ki nam omogoča branje in pošiljanje signalov ter vrednosti na napravo PLC. Ta strežnik že obstaja in ga je bilo potrebno samo pravilno nastaviti [11].

Naloge, ki jih opravlja so naslednje.

- Posredovanje podatkov o trenutnem toku na tuljavah
- Nastavitev toka na tuljavah
- Branje stikal za prekoračitev motorjev
- Pošiljanje signalov Interlock in Interlock ključa za preklic (I*nterlock key override*)
- Posredovanje informacij o stanju posameznega motorja
- Branje merilnika za nagib (veliko bolj zanesljivo kot računanje iz položaja motorjev)

4.3 Grafični vmesnik

Grafični vmesnik omogoča uporabniku, da upravlja z undulatorjem brez kateregakoli znanja o kontrolnemu sistemu TANGO. Namenjen je tako za znanstvenike, ki bodo izvajali poizkuse, kot za inženirje, ki bodo undulator vzdrževali. Lahko rečemo, da je grafični vmesnik sestavljen iz dveh delov, prvi za navadne uporabnike in drugi za inženirje. Inženirski del je naprej razdeljen glede funkcionalnost. Inženir lahko premika posamezne motorje, izvaja kalibracijo ali spreminja nastavitve. Vsega skupaj šteje grafični vmesnik štiri zavihke.

Grafični vmesnik mora zadostiti naslednjim kriterijem.

- Navadnemu uporabniku omogočiti premikanje razmaka, nagiba, spreminja faze in faznega odmika ter prikazovanje trenutnega stanja vseh motorjev, stikal in merilcev nagiba
- Ne glede na katerem zavihku grafičnega vmesnika se uporabnik nahaja, omogočiti sprožitev signala Stop all

- Vedno prikazuje trenutno stanje strežnika in zastavic za signal Interlock ter posebnega inženirskega stanja
- Inženirju omogočiti premikanje posameznega motorja ali premikanje posamezne plošče
- Možnost kalibracije posameznega motorja ali kalibracije skupine motorjev s pomočjo posebne funkcije (priporočeno)
- Omogočiti spreminjanje parametrov kot so hitrost in pospešek za vse motorje in skupine motorjev

5 Mikrokoda za Galil DMC 4080

Začeli bomo na za nas najnižjem nivoju. Ker imamo dve napravi, ki nadzorujeta različni skupini motorjev, je tudi koda razdeljena na dva dela. Večina funkcionalnosti je enakih, kjer pa so razlike, bo to posebej navedeno. Generična koda je skupna obema in bo predstavljena na začetku. Sledi predstavitev specifičnih spremenljivk, zastavic in niti za nadzor undulatorja. Na koncu z diagrami poteka predstavimo premikanje razmaka in faze ter niti, ki skrbi za stanje naprave.

Ker napravi podpirata nadzor nad osmimi motorji, ki se označujejo s črkami A do H, se je bilo potrebno dogovoriti, kako bomo naše motorje povezali. Odločili smo se, da bomo uporabili prve štiri črke (povezave) pri obeh napravah, kot prikazuje tabela 2.

Motor	Galil oznaka
Z1	A
Z2	B
Z3	C
Z4	D
X1	A
X2	B
X3	C
X4	D

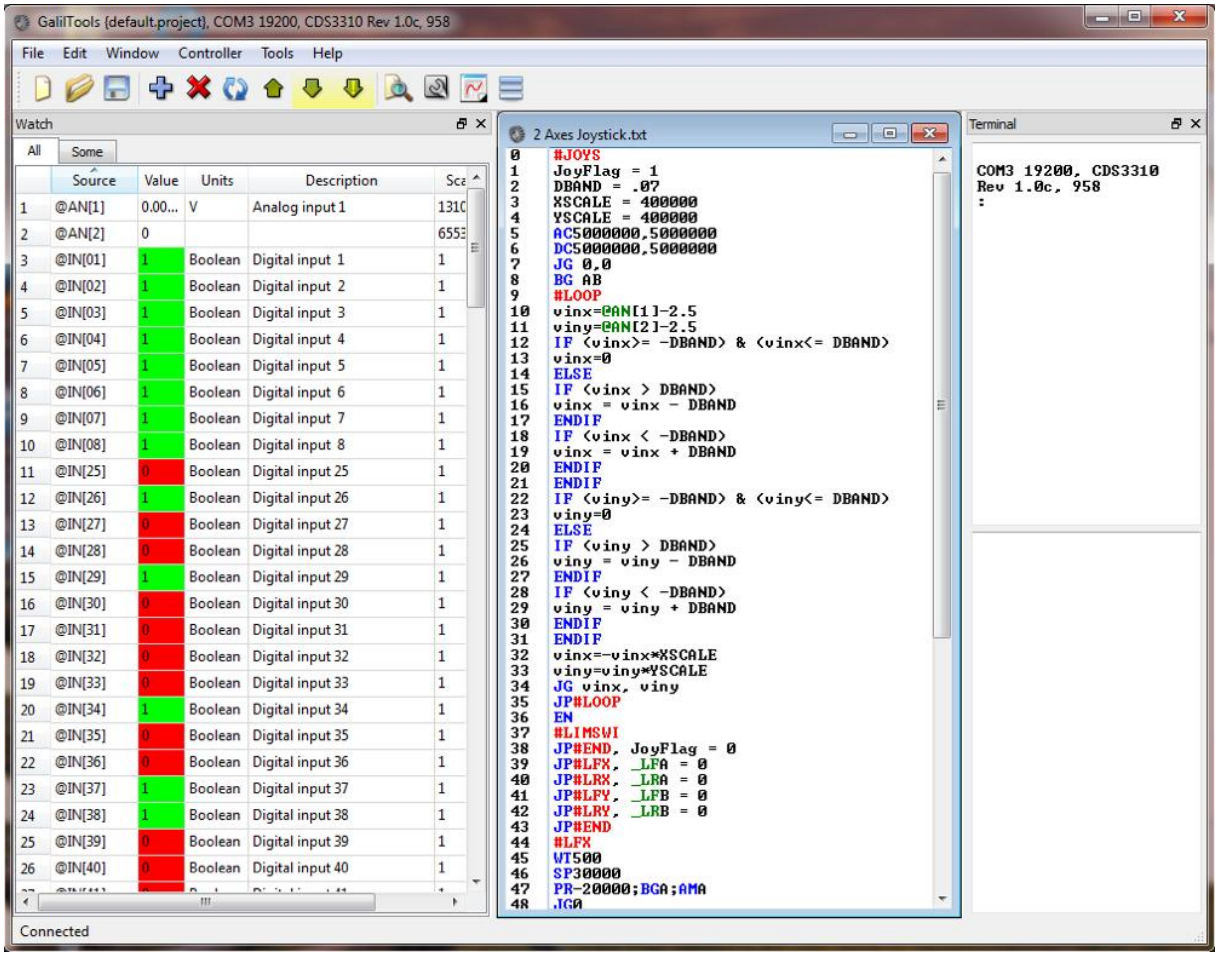
Tabela 2: Oznake motorjev na napravi Galil DMC [11]

Poskušali smo tudi to, da bi uporaba črk za motorje bila spremenljiva, a se je izkazalo, da zaradi pomanjkanja ukazov in fleksibilnosti v mikrokodi, se naša namera ni izšla [14].

5.1 Razvijalno okolje

Za programiranje, nalaganje in spremljanje dogajanja na napravi DMC smo uporabili program GalilTools [15]. Izgled programa je prikazan na sliki 10. Sicer je program zelo pomanjkljiv, programerja ne obvešča o sintaktičnih napakah, niti ne olajšuje pisanja mikrokode (samodejno zamikanje, dopolnjevanje imen ukazov, …). Ima pa tudi dobre lastnosti. Omogoča spremljanje vseh zastavic naprave DMC, tako vedno vemo, kaj se z njo dogaja oziroma lažje diagnosticiramo vzrok napake. Prek konzole lahko izvajamo ukaze in v primeru, da smo v mikrokodo dodali sporočila, le ta tudi vidimo. To je tudi edini način za iskanje semantičnih napak. Obstaja plačljiva različica, ki doda še funkcionalnosti za podrobnejše spremljanje

premikanja in optimizacijo PID (*proportional integral derivative*) parametrov [16]. Za naše potrebe je zastonjska različica zadostovala.



Slika 10: GalilTools [17]

5.2 Generični del

Obe napravi imata skupno generično kodo, ki jo je potrebno razumeti, da lahko dodamo specifične funkcionalnosti za undulator. Generična koda je namenjena delovanju skupaj s strežnikom ControlBox. Omogoča nam enostavna premikanja posameznih motorjev in nastavljanje parametrov posameznega motorja. Podpira premikanje dveh motorjev hkrati (to bomo izkoristili za inženirske premike plošč), obveščanje o stanju posameznih motorjev in nastavljanje ter spreminjanje kontrolnih zastavic.

5.2.1 Spremenljivke

Ime	Tip	Opis
Pos[x]	Double	Absolutna končna pozicija motorja v enotah kodirnika (*counts*).
Dbw[x]	Double	Potrebna natančnost v enotah kodirnika.
IniPos[x]	Double	Začetna pozicija v enotah kodirnika.

IniSpeed[x]	Double	Začetna hitrost v enotah kodirnika na sekundo.
IniTyp[x]	ULong	Strategija iskanja začetne pozicije.
Bck[x]	Double	Kompenzacija za prazni tek (*Backlash*) v enotah kodirnika.
NbRetry[x]	Ulong	Število poskusov iskanja začetne pozicije.
CpMp[x]	Uint	Zastavica za konfiguracijo.
StabTime[x]	Ulong	Čas umirjanja motorjev v milisekundah.
GrAxis[x]	int (id motorja)	Številka motorja, ki gospodari premikanju. Vrednost -1 pomeni, da ni gospodarja oziroma motor ni povezan.
GrRatio	Double	Razmerje med premikanjem gospodarja in sužnja.
Cmd[x]	$1-$100 (vrednosti)	Ukazi.
Stat[x]	$1-$20000	Status motorja.

Tabela 3: Spremenljivke pri generični mikrokodi [12]

Ker je vsaka vrednost specifična za posamezen motor so skoraj vse spremenljivke iz tabele 3 polja. Spremenljivka x lahko zavzame vrednosti od 0 do 7 (za 8 motorjev), nekje pa tudi do 15, tako lahko ohranimo tudi stare vrednosti. Za nas sta najbolj pomembni Cmd in Stat polji. Z njima lahko manipuliramo motorje po želji in brez spreminjanja generične kode, dodamo nove funkcionalnosti.

5.2.2 Zastavice

Vrednost	Pomen
$1	Ustavi motor.
$2	Premakni v pozitivno smer.
$4	Premakni v negativno smer.
$8	Premakni na absolutno pozicijo.
$10	Začetno referenčno iskanje pozicije.
$20	Vključi motor.
$40	Izključi motor.
$80	Posodobi gospodarja za skupno premikanje.
$100	Zakleni pozicijo.

Tabela 4: Pomen bitov spremenljivke Cmd [12]

Vse vrednosti predznačene z znakom $ so v šestnajstiškem sistemu. V tabeli 4 so vsi ukazi, ki jih strežnik ControlBox uporablja. Edini ukaz, ki ga bomo uporabili v naših posebnih funkcijah za premikanje razmaka in faze, ima vrednost $80. Namreč v primeru, da so motorji med seboj povezani, lahko pride do težav. Zato na začetku funkcije premikanja faze ali razmaka vedno pregledamo, če so vsi motorji samostojni in v primeru da niso, jih v to stanje postavimo. Ostale funkcije bomo uporabili samo preko strežnika ControlBox. S tem zmanjšamo možnost morebitnih napak.

Vrednost	Pomen
$1	Tip motorja (0 – koračni, 1-servo).
$2	Prisotnost kodirnika.
$4	Koračni motor v vzdrževalnem načinu.
$8	Oddaljeni ročni način je aktiviran.
$10	Specifična mikrokoda se izvaja.
$20	Avtorizacija za vžig in izklop motorja.
$40	Premikanje omogočeno (**to bomo potrebovali v naših funkcijah**).
$80	Uspešen prihod na cilj.
$100	Neuspešen prihod na cilj.
$200	Iskanje referenčne pozicije v teku.
$400	Uspešno dobljena referenčna pozicija.
$800	Referenčna pozicija ni bila dobljena.
$1000	Globalna napaka pri premikanju. Več motorjev ni prišlo na cilj.
$2000	Napaka pri sledenju.
$4000	Motor je suženj pri skupnem gibanju.
$8000	Motor je gospodar pri skupnem gibanju.
$10000	Uspešno nastavljen na začetne vrednosti.
$20000	Zaklenjen na pozicijo.

Tabela 5: Pomen bitov Stat spremenljivke [12]

Najpomembnejša zastavica za nas bo $40, saj z njo lahko onemogočimo premikanje oziroma uporabniku preprečimo izdajanje ukazov za premikanje, ko se nahajamo v stanjih Stop all ali Interlock. Ostale zastavice iz tabele 5 bomo spreminjali samo preko strežnika ControlBox.

5.2.3 Niti

Naprava DMC nam omogoča izvajanje do 8 niti [13]. Nekatere so rezerviranje za delovanje generične kode, druge smo uporabili mi.

Številka niti	Funkcija	Opis
0		Glavna nit.
1	#POM	Uporablja ControlBox.
2	#CORCHK , #PECORR	Uporablja ControlBox.
3	Rezervirano	Rezervirano za ControlBox.
4	**#STATETH**	Pregledovanje trenutnega stanja. Izvršuje akcije stanj Interlock in Stop all.
5	**#COILS**	Računanje vrednosti za korekcijske tuljave. Pri napravi DMC Gap se računa trenutni razmak, nagnjenost in odmik od centra žarka, pri napravi DMC Phase pa računamo trenutno fazo in odmik.
6	**#CORGAP**	Nit se uporablja samo pri napravi DMC Gap. Popravlja razmak. Zažene se na koncu premikanja razmaka ter med premikanje faze.
7	**#GAP, #PHASE**	Nit za premikanje razmaka ali faze. Ena naprava DMC ima funkcijo #GAP, druga pa funkcijo #PHASE.

Tabela 6: Niti in funkcije na napravi DMC [12]

Poudarjeno napisane niti iz tabele 6 izvršujejo našo kodo in bodo podrobneje opisane v kasnejših razdelkih.

5.3 Specifični del za undulator

Ta del skrbi za premikanje razmaka in faze, računanje vrednosti za korekcijske tuljave, vzdrževanje razmaka, ponastavljanja rotacijskih kodirnikov in spreminjanje stanja ter generičnih zastavic v primeru signalov Interlock in Stop all. Imena in opisi dodanih zastavic in globalnih nastavitev so zaradi preglednosti v tabelah 7 in 8.

5.3.1 Zastavice

Ime	Opis
IlockAct	Zastavica, ki označuje, da je signal Interlock aktiven. Ustaviti je potrebno vse motorje in preprečiti izvršbo ukazov za premikanje.
StopRq	Zastavica za signal Stop all, ki ga sproži uporabnik. Ustaviti je potrebno vse motorje in preprečiti izvršbo novih ukazov za premikanje.

AlwMvmt	Zastavica za onemogočanje premikanja motorjev. Nastavi se glede na signala Interlock in Stop all.
OffInit	Postavi se na 1, ko se pošljejo odmiki (*offsets*) za vse motorje iz strežnika Undulator. Do takrat so vsa premikanja onemogočena.
CorFlag	Začasno uporabljena v niti #GAP. Vsebuje informacije, če je kateri od motorjev prešel stikalo za mehko omejitev.
PhCmd	Ukaz, ki se prenaša iz naprave DMC Gap na napravo DMC Phase preko strežnika Undulator. 1 – omogoči motorje Phase 2 – onemogoči motorje Phase
AxesFlgs	Vsebuje informacije o napakah (napake na kodirniku, komunikacijske napake in napake na motorju). V primeru, da je vrednost več od nič, bo premikanje onemogočeno.
RfshCtr	Števec specifičen za napravo DMC Gap. Ko mine določen čas brez premikanja motorjev Gap, je potrebno osvežiti pozicije na rotacijskih kodirnikih.
StopGap	Označuje ukaz za ustavitev motorjev Gap. Uporablja v niti #STATE.
StopPh	Označuje ukaz za ustavitev motorjev Phase. Uporablja v niti #STATE.
@OUT[5]	Digitalni izhod/vhod uporabljen za koordinacijo premikanja faze in razmaka. Ko premikamo fazo, je potrebno konstantno popravljati razmak (ukrivljanje zaradi magnetne sile). Uporablja invertno logiko.

Tabela 7: Zastavice specifične za mikrokodo namenjeno nadzoru undulatorja [12]

5.3.2 Globalne nastavitve

Ime	Opis
Offsets[4]	Odmik za posamezen motor. Vsaka naprava DMC ima štiri motorje. Vrednost je v enotah kodirnika.
UMOfsts[4]	Odmik za posamezen motor v mikrometrih.
SegLimit	Konstanta maksimalne velikosti segmenta pri linearni interpolaciji točk za simetrično premikanje.

Tabela 8: Globalne spremenljivke za nastavitve [12]

Zaradi nenatančnost pri postavitvi kodirnikov in motorjev potrebujemo odmik za vsak motor. Odmike smo izračunali oziroma izmerili pri kalibraciji motorjev in se načeloma ne bodo več spreminjali.

Pri premikanju razmaka in faze uporabljamo linearno interpolacijo točk. To je posebna funkcija premikanja na napravi DMC. Tako lahko zagotovimo, da bodo motorji istočasno

dosegli vsako točko in se premikali simetrično. Velikost razdalje med točkami je omejena, zato smo implementirali posebno funkcijo, ki nam pot do cilja segmentira na manjše dele velikosti SegLimit.

5.3.3 Spremenljivke za premikanje razmaka

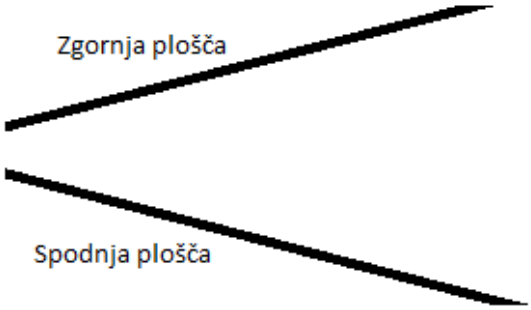
Ime	Opis
GapOfst	Željeni odmik od centra žarka v enotah kodirnika.
Gap	Željeni razmak v enotah kodirnika.
Taper	Željeni nagib v enotah kodirnika.
FinalG[4]	Izračunane končne pozicije motorjev glede na želene parametre. Uporabimo pri premikanju in kasnejši korekciji pozicije.
GapTmp[4]	Začasna spremenljivka pri premikanju razmaka.

Tabela 9: Spremenljivke za premikanje razmaka [12]

Prve tri spremenljivke iz tabele 9 je potrebno poslati iz strežnika Undulator, preden začnemo premikanje. Pri izbiri odmika od centra žarka moramo biti pozorni na to, da zmanjšuje maksimalni teoretični razmak. V primeru, da je vrednost odmika pozitivna, pomeni hitrejšo dosego stikala za omejevanje premikanja zgornje plošče. Načeloma je odmik od centra žarka majhen oziroma ga ni. Pri normalni uporabi je velikost razmaka omejena programsko in je veliko manjša, tako da do stikala sploh ne moremo priti.

Potrebno je vedeti, da spremenljivka Taper predstavlja skupni nagib, torej je za vsak motor potrebno vzeti le četrtino vrednosti.

Ker obstajata dve različni možnosti interpretacije pozitivnega nagiba, je bil potreben dogovor, katero bomo izbrali. Odločili smo se, da bo pri pozitivnem nagibu razmak na desni strani večji, kot je razvidno iz slike 11.



Slika 11: Primer pozitivnega nagiba

Izračun končne pozicije za zgornji levi motor v našem primeru, podajata naslednji enačbi.

$$aTaper = \frac{Taper}{4} \quad (1)$$

$$FinalG[0] = \frac{Gap}{2} + GapOfst + Offsets[0] - aTaper \quad (2)$$

Izračuni za ostale motorje so izpeljanke enačbe 2.

5.3.4 Nastavitve za motorje Gap

Ime	Opis
ERNum	Števec za konverzijo iz linearnih v rotacijske enote kodirnika. Uporabljamo dve spremenljivki zaradi natančnosti.
ERDen	Imenovalec za konverzijo iz linearnih v rotacijske enote kodirnika. Uporabljamo dve spremenljivki zaradi natančnosti.
UMDen	Konstanta za konverzijo iz enot linearnega kodirnika (LinCts) v mikrometre (Ums). $Ums = \frac{LinCts}{UMDen}$ (3)
GapSpd	Hitrost spreminjanja razmaka v enotah rotacijskega kodirnika na sekundo.
GapAcc	Pospešek pri premikanju razmaka v enotah rotacijskega kodirnika na kvadratno sekundo.
TprSpd	Hitrost spreminjanja nagiba v enotah rotacijskega kodirnika na sekundo.
TprAcc	Pospešek pri premikanju nagiba v enotah rotacijskega kodirnika na kvadratno sekundo.

Tabela 10: Nastavitve za premikanje razmaka [12]

Nastavitve za hitrosti in pospeške iz tabele 10 veljajo samo pri premikanju z uporabo funkcije #GAP.

Ker uporabljamo pri motorjih Gap dva različna tipa kodirnikov, je potrebna pretvorba iz enot linearnega v enote rotacijskega kodirnika. RotKod predstavlja število enot rotacijskega kodirnika, medtem ko število enot linearnega kodirnika predstavlja LinKod.

$$RotKod = LinKod * \frac{ERNum}{ERDen} \quad (4)$$

Enačba 4 zadošča za posodobitev obeh rotacijskih kodirnikov motorjev spodnje plošče, saj uporabljata samo po en linearni kodirnik na stran (talni – floor). Pri motorjih zgornje plošče dodamo še en člen, saj je pozicija določena iz vrednosti dveh linearnih kodirnikov na stran (talnega – floor in razmak – gap kodirnika).

V tem primeru dobimo:

$$RotKodZgoraj = (LinKodGap - LinKodFloor) * \frac{ERNum}{ERDen} \quad (5)$$

Potrebno je vedeti, da se vrednostim kasneje doda odmik (*offset*) in tako dobimo dejansko oddaljenost motorja od centra žarka.

5.3.5 Spremenljivke za premikanje faze

Ime	Opis
PhMdRq	Želena faza.
PhMd	Trenutna faza.
PhDest[4]	Končne pozicije motorjev glede na izbrano fazo in odmik. Izračunamo jih že na strežniku Undulator in jih pošljemo na napravo DMC pred začetkom premikanja.
PhTmp[4]	Začasna spremenljivka pri premikanju motorjev Phase.

Tabela 11: Spremenljivke za premikanje faze [12]

Pred zagonom funkcije #PHASE, je potrebno na napravo DMC poslati vrednosti prvih treh spremenljivk iz tabele 11.

Pri premikanju faze je potrebno vedeti to, da se v primeru spremembe faze, motorji najprej poravnajo na izhodiščno pozicijo (vsi na 0) in šele nato se dva premakneta, da dosežeta želeno fazo in odmik. V tem trenutku spremenljivka PhMd zavzame vrednost spremenljivke PhMdRq. To vrednost potrebuje nit, ki osvežuje trenutne vrednosti za korekcijske tuljave.

5.3.6 Nastavitve za motorje Phase

Ime	Opis
PhSpd	Hitrost spreminjanja odmika v enotah kodirnika na sekundo.
PhAcc	Pospeški pri premikanju faze v enota kodirnika na kvadratno sekundo.
UMDen	Konstanta za pretvorbo iz enot linearnega kodirnika v mikrometre. Enačba 3.
PhThold	Minimalen zahtevan odmik za začetek premikanja. V primeru, da je zahtevan odmik od trenutnega premajhen, se premikanje ne bo začelo.

Tabela 12: Nastavitve za premikanje faze [12]

Nastavitve potrebne za delovanje funkcije #PHASE so razvidne iz tabele 12.

Pri motorjih Phase imamo samo linearne kodirnike in zato ni potrebe po pretvarjanju.

5.3.7 Spremenljive za korekcijske tuljave

Ime	Opis
coilG	Trenutni razmak v mikrometrih.
coilT	Trenutni nagib v mikrometrih.
coilC	Trenutni odmik od centra žarka v mikrometrih.
coilPM	Trenutna faza.
coilPO	Trenutni fazni odmik v mikrometrih.

Tabela 13: Spremenljivke za korekcijske tuljave [12]

Vse spremenljivke se konstantno posodabljajo v neskončni zanki in prenašajo na strežnik Undulator. Ena nit na napravi DMC je namenjena samo za to. Povprečen čas posodobitve vseh spremenljivk je 15 milisekund.

5.4 Niti in funkcije

Generičnih niti ne bomo posebej predstavljali, ker na delovanje undulatorja ne vplivajo veliko. Uporabljajo se samo za spremljanje stanja naprave DMC in za inženirske premike.

Posebno se bomo osredotočili na obe funkciji premikanja in nit za posodabljanje stanja. Vse bomo predstavili tudi z diagramom poteka, saj bo tako razumevanje delovanja lažje.

5.4.1 #COILS

Nit prisotna na obeh napravah DMC. Razlikuje se le v tem, da računa veljavne parametre za določeno napravo. Torej na napravi DMC Gap računa trenutni razmak, nagib in odmik od centra žarka, na napravi DMC Phase pa trenutno fazo in fazni odmik. Zažene se ob inicializaciji in se nikoli ne konča. Posodablja spremenljivke prikazane v tabeli 13, ki jih strežnik Undulator bere in uporablja pri izračunu potrebnega toka za korekcijske tuljave.

5.4.2 #CORGAP

Funkcija, ki se izvede v posebni niti na napravi DMC Gap. Zažene se jo v dveh primerih.

- Po končanem premikanju razmaka
- Med premikanjem faze (signal preko digitalnega vhoda)

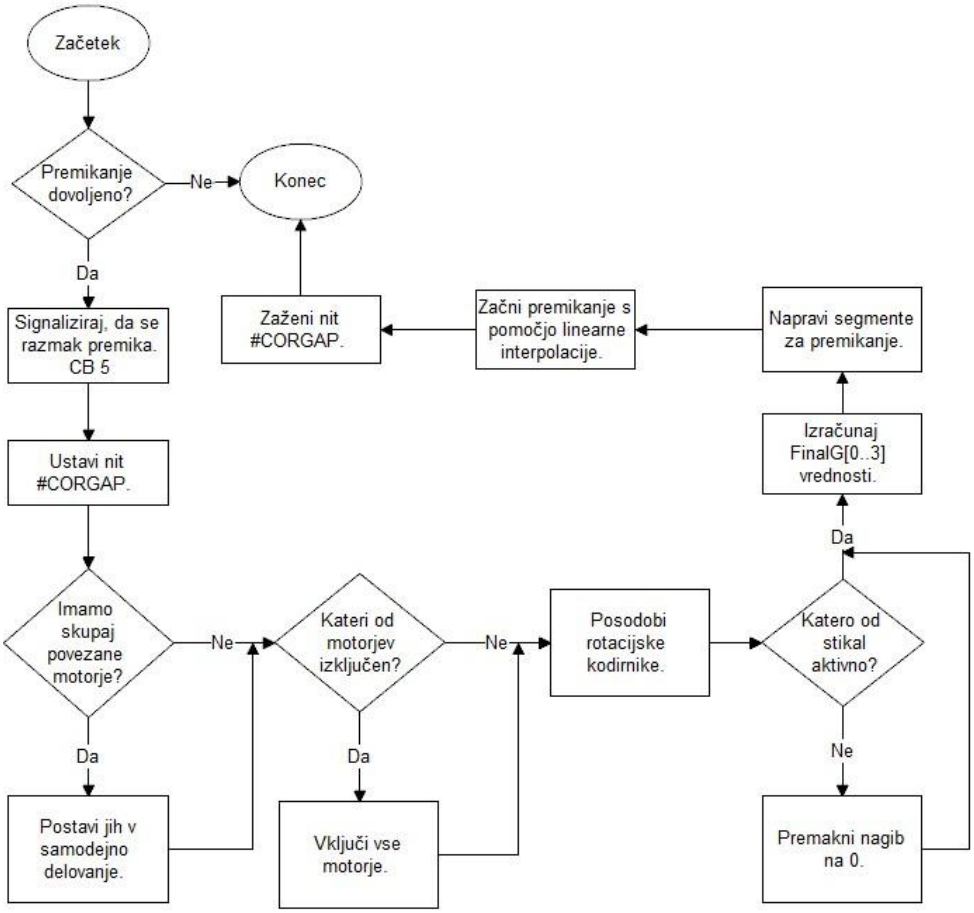
V prvem primeru poskrbi, da res dosežemo željeni razmak, v drugem pa popravi pozicijo zaradi upogiba plošč. Upošteva izračunane končne cilje motorjev (spremenljivka FinalG) in jih poskuša obdržati na tem mestu. To ni vedno mogoče, zato dovolimo minimalna odstopanja.

5.4.3 #GAP

Funkcija, ki jo izvedemo v posebni niti, ko želimo spremeniti razmak, nagib in odmik od centra žarka. Pred izvedbo mora uporabnik prek strežnika Undulator poslati prve tri parametre iz tabele 9. S pomočjo teh izračunamo končne pozicije motorjev. Naslednje akcije so pomembne pri premikanju razmaka.

- Preveriti ali je vse v redu z motorji
- Nagib na 0
- Premakni razmak upoštevajoč določenega odmika od centra žarka
- Nagib na želeno vrednost
- Popraviti pozicije

Zaradi varnosti se pred postavljanjem nagiba na nič, preveri stikala za mehko omejitev. To prepreči morebitno zatikanje pri premikanju nagiba na nič. Raje kot z besedami, bomo izvajanje prikazali z diagramom poteka na sliki 12.



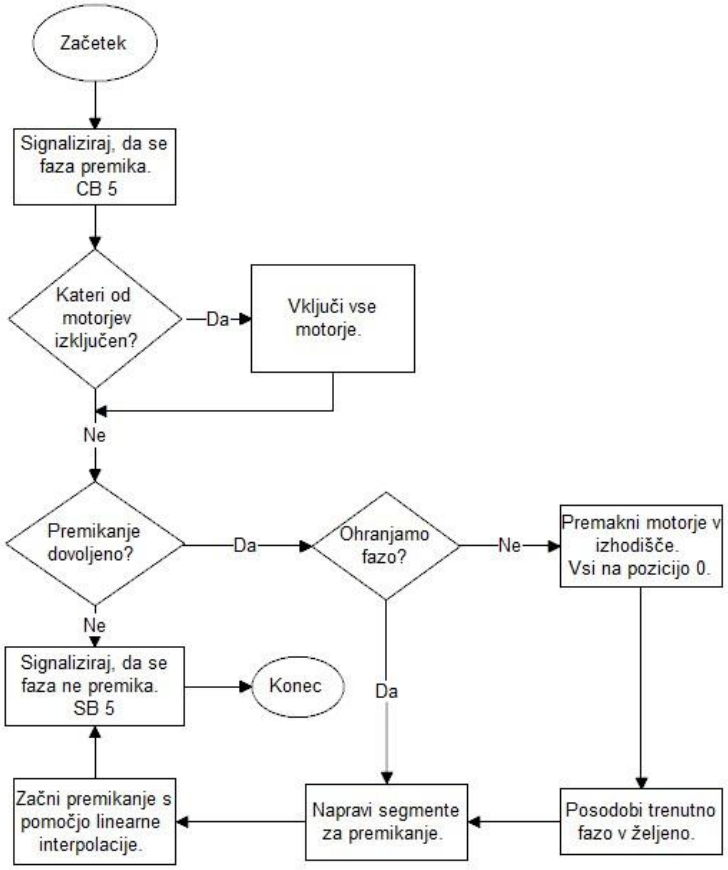
Slika 12: Diagram poteka funkcije #GAP [12]

5.4.4 #PHASE

Funkcija, ki jo izvedemo v posebni niti, ko želimo spremeniti fazo ali fazni odmik. Za razliko od funkcije #GAP, ki sama izračuna končne pozicije motorjev, funkcija #PHASE pričakuje vrednosti že izračunane. To je predvsem zato, ker je računsko logiko lažje in lepše napisati v višje nivojskem jeziku. Poleg tega potrebuje še parameter, ki določa fazo. Naprava DMC faz ne pozna in samo uporablja vrednosti, ki jih dobi kot parametre. Za konsistenco faze in končnih položajev motorjev je torej odgovoren strežnik Undulator. Naslednje akcije so pomembne pri premikanju faze.

- Preveri ali je vse v redu z motorji
- Signaliziraj premikanje napravi DMC Gap
- Če faze ne spreminjamo premaknemo samo odmik
- V primeru spremembe faze vse motorje poravnamo na izhodišče
- Premaknemo dva motorja na željeni fazni odmik
- Signaliziramo konec premikanja

Na sliki 13 je prikazan diagram poteka funkcije #PHASE.



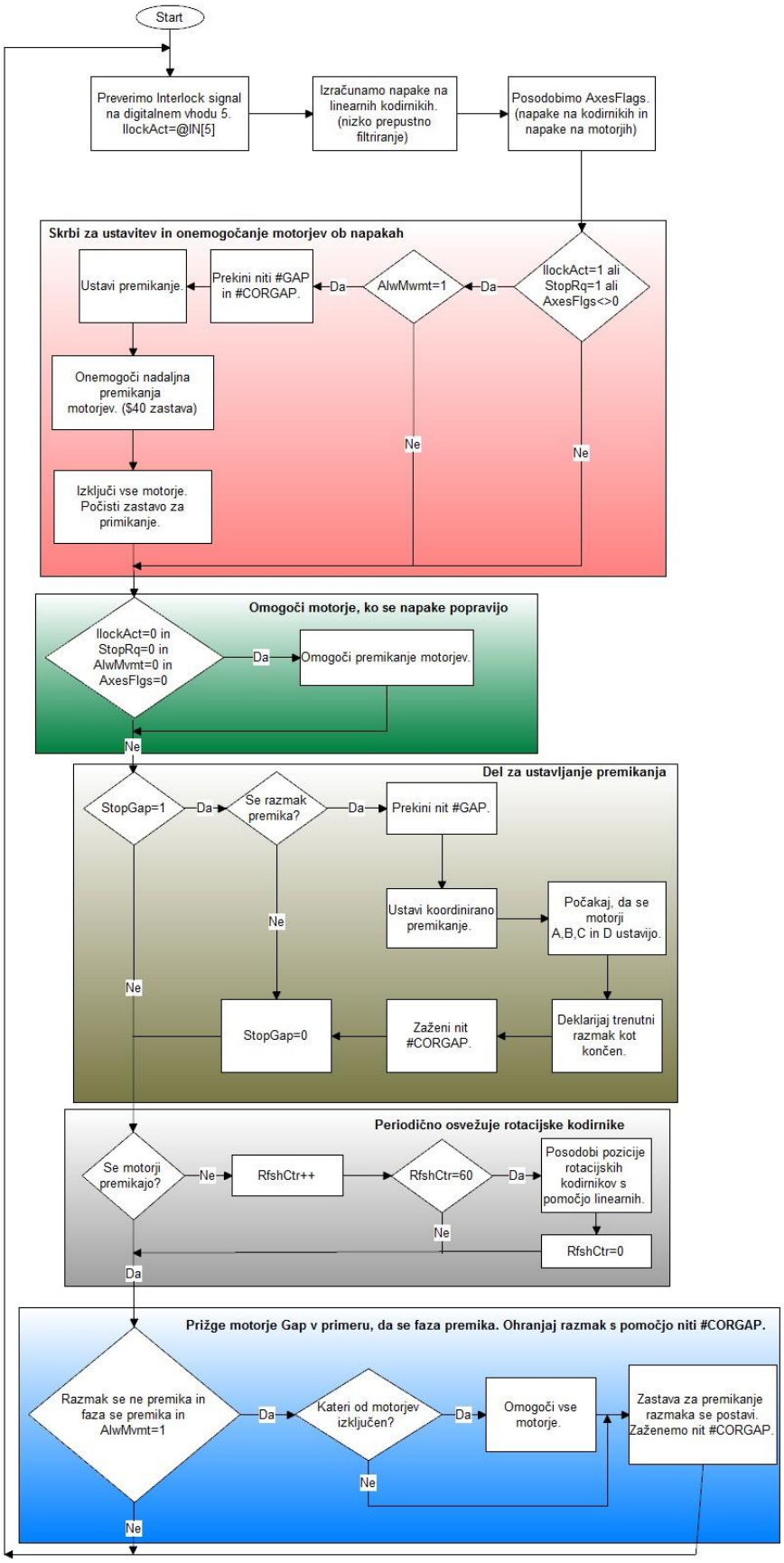
Slika 13: Diagram poteka funkcije #PHASE [12]

5.4.5 #STATETH

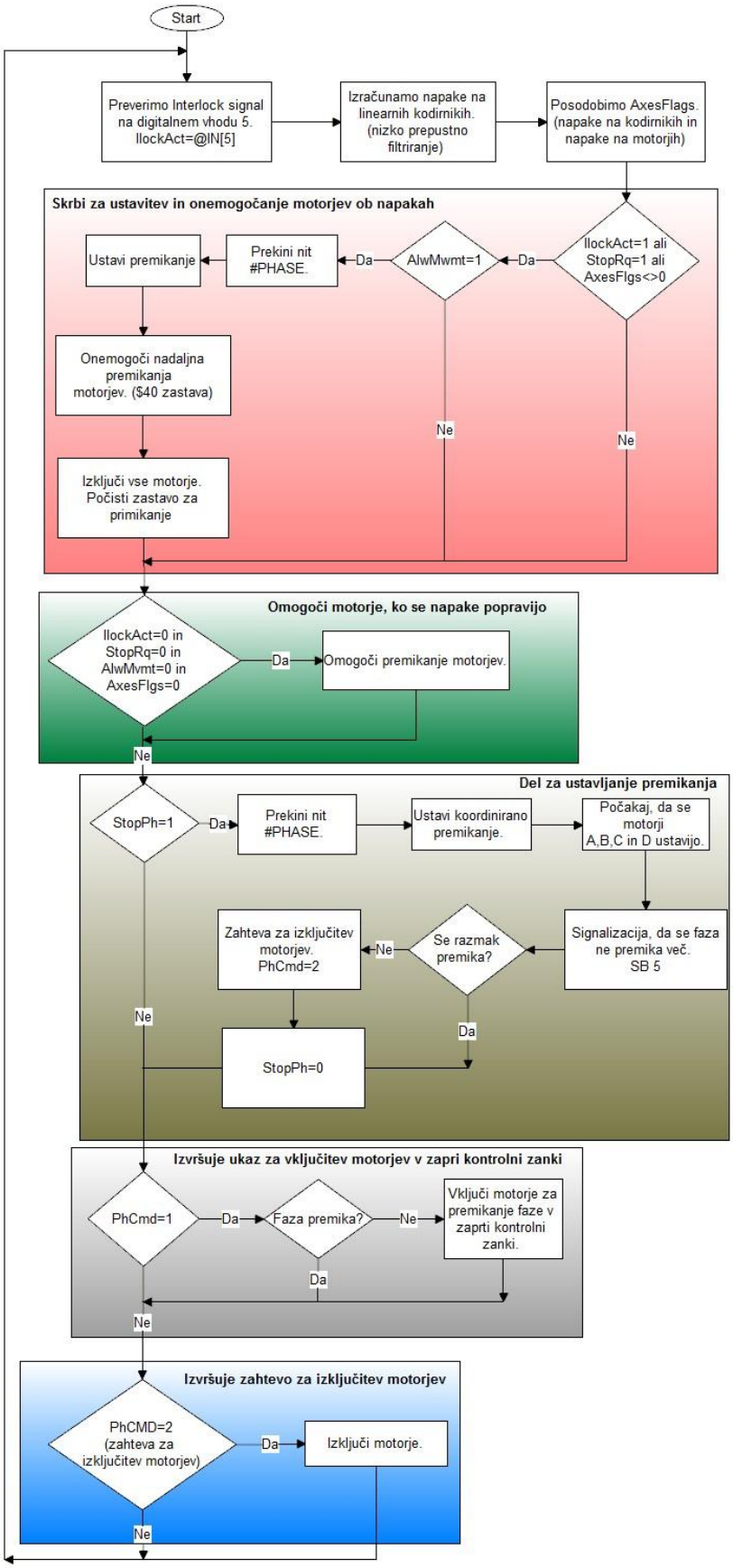
Nit se izvaja na obeh napravah DMC, a se razlikuje po implementaciji. Skrbi za naslednje naloge.

- Preverja signala Stop all in Interlock
- Preverja stanje kodirnikov in motorjev
- V primeru napak ustavi vse motorje in onemogoči nadaljnje ukaze premikanja
- Izvršuje zahteve za ustavitev motorjev
- Skrbi za koordinirano vžiganje in ugašanje motorjev (pri skupnem gibanju)

Ta nit je najpomembnejša, saj skrbi za varnosti in pravilno delovanje undulatorja. Skozi fazo razvoja smo obe verziji veliko spreminjali, da bi pravilno delovali v vseh možnih situacijah, ki smo jih predvideli. Hkrati smo poskrbeli za čim bolj optimalno izvajanje niti. Obe implementaciji bomo predstavili z diagramom poteka (sliki 14 in 15), saj sta si kljub podobnostim, še vedno precej različni.



Slika 14: Diagram poteka niti #STATETH na napravi DMC Gap [12]



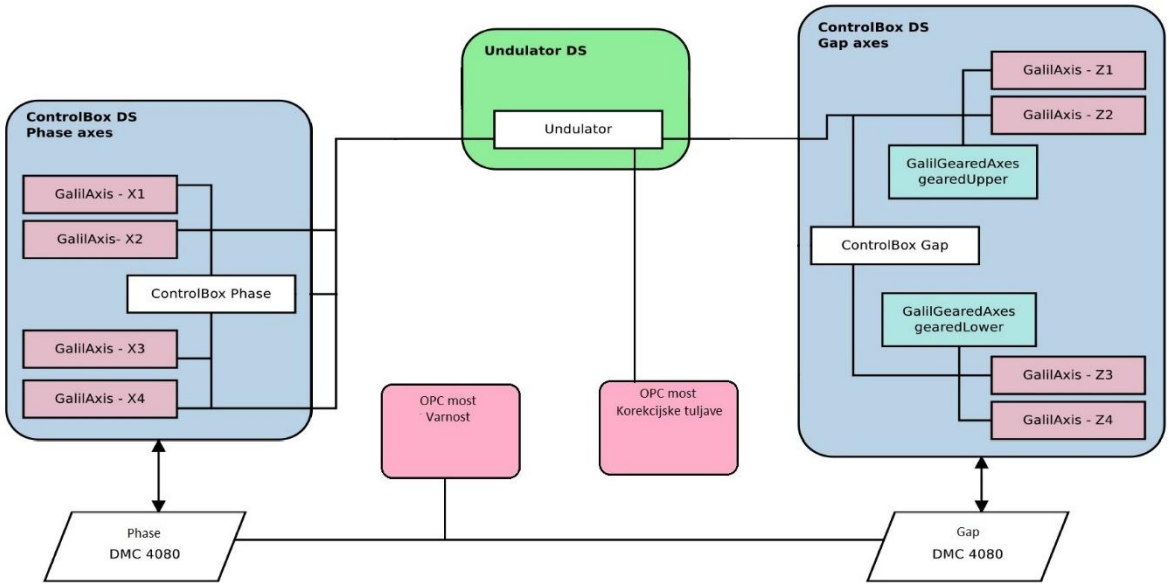
Slika 15: Diagram poteka niti #STATETH na napravi DMC Phase [12]

6 Strežniki TANGO

V tem poglavju bodo predstavljeni vsi strežniki TANGO, ki so uporabljeni za nadzor undulatorja.

6.1 Pregled

Da lahko upravljamo z undulatorjem, moramo na sistemu imeti zagnanih kar pet strežnikov TANGO za naprave. Dva strežnika ControlBox (za vsako napravo DMC po eden), dva strežnika OPCaccess (PLC za varnost in PLC za korekcijske tuljave) in strežnik Undulator, kot je razvidno iz slike 16.



Slika 16: Sistem TANGO za undulator [12]

Preden začnemo z razvojem strežnika Undulator, moramo vedeti, katere funkcionalnosti so že izdelane, kako delujejo in kako do njih dostopati. Podrobneje bomo predstavili delovanje strežnika ControlBox in OPCaccess ter njuni konfiguraciji. Nato se lahko osredotočimo na razvoj strežnika Undulator.

6.2 Konfiguracija strežnika ControlBox

Kaj nam strežnik ControlBox ponuja, smo že opisali v prejšnjih poglavjih. Potrebno ga je le pravilno nastaviti (preko lastnosti) in predstaviti način uporabe ponujenih funkcionalnosti.

Ker imamo dva strežnika ControlBox, moramo opraviti tudi dve konfiguraciji. Glede na to, da so si nastavitve pri obeh precej podobne, bomo predstavili samo konfiguracijo strežnika ControlBox, ki nadzira napravo DMC Gap (poimenovali ga bomo ControlBox Gap). Razlika pri konfiguraciji strežnika ControlBox, ki nadzoruje napravo DMC Phase (poimenovali ga

bomo ControlBox Phase), je samo v tem, da motorji uporabljajo linearne kodirnike namesto rotacijskih in ne potrebuje instanc objektov za skupno premikanje [12].

Podatke strežnika dodamo v podatkovno bazo s pomočjo programa Jive, nato je potrebno izpolniti vse lastnosti uporabljenih objektov. Pri ControlBox Gap bomo uporabili tri različne tipe objektov z več instancami. Potrebujemo.

- Eno instanco objekta ControlBox
- Štiri instance objekta GalilAxis
- Dve instanci objekta GalilGearedAxes

Pri kreiranju ControlBox Phase zapisa v podatkovno bazo lahko izpustimo instanci objekta GalilGearedAxes, saj ju ne potrebujemo.

6.2.1 Objekt ControlBox

Skrbi za komunikacijo z napravo DMC. Ustvari povezavo in osvežuje stanje strežnika. Naprava DMC ima poseben ukaz, ki vrne vse podatke v surovi (binarni) uporabniku nečitljivi obliki. Objekt ControlBox izlušči vse potrebne podatke, jih uporabi za svoje delovanje in predstavi v uporabniku prijaznejši obliki.

Omogoča nam neposredno pošiljanje ukazov na napravo DMC z ukazom **ExecLowLevelCmd** [3]. Ta ukaz nam zelo olajša delo pri načrtovanju strežnika Undulator, saj nam tako ne bo potrebno vzpostaviti dodatne povezave na napravo DMC za izvedbo dodatnih funkcij in niti, ki smo jih dodali generični mikrokodi.

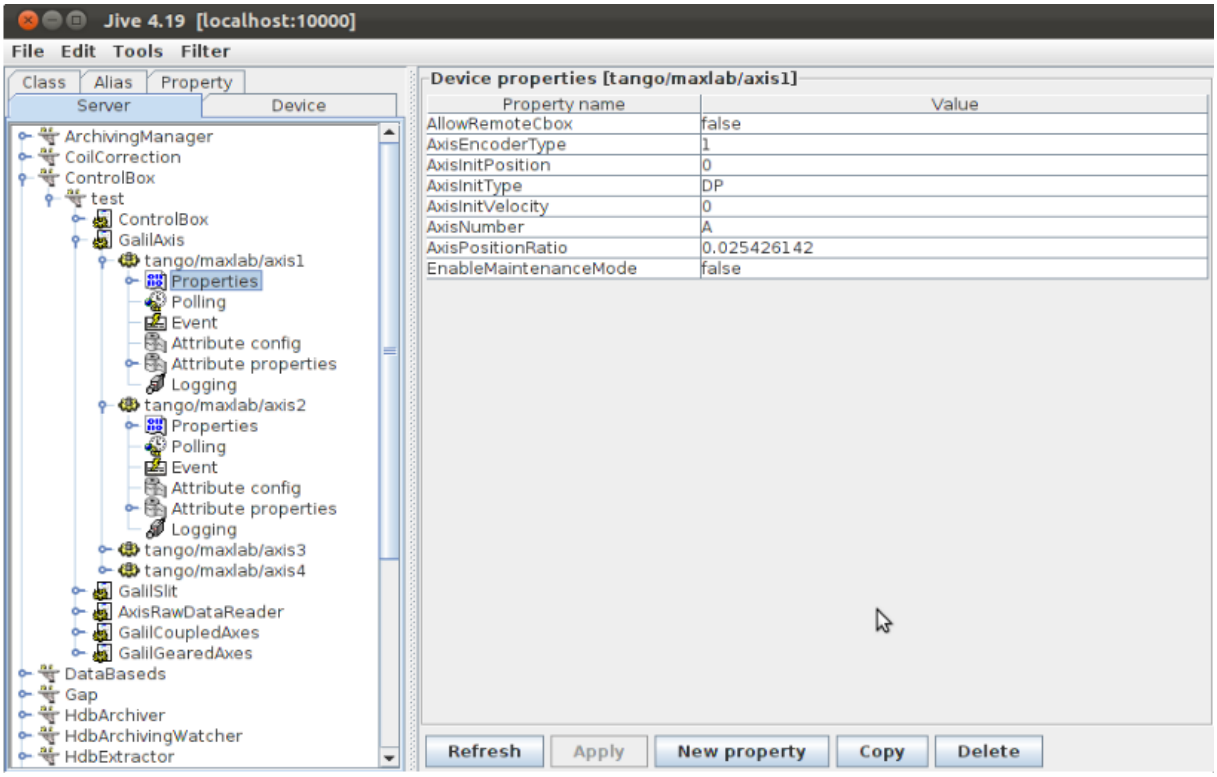
Objekt ima štiri lastnosti opisane v tabeli 14.

Ime	Opis
DataRecordPollingPeriod	Čas v milisekundah za osveževanje stanja podatkov (prek posebnega ukaza). Minimalen čas je sto milisekund.
IPAddress	Naslov IP naprave DMC.
Port	Vrata na katerih posluša naprava DMC. Načeloma je to od 5000 dalje.
TCPTimeOut	Čas v milisekundah, ki lahko poteče brez odgovora, preden sprožimo napako. Tri sekunde je priporočen čas.

Tabela 14: Lastnosti objekta ControlBox [3]

6.2.2 Objekt GalilAxis

Vsaka instanca tega objekta predstavlja natanko en motor priklopljen na napravo DMC. V našem primeru bomo imeli štiri instance. Vse štiri bodo imele enake nastavitve razen črke pri oznaki motorja. Vse potrebne nastavitve smo opisali v tabeli 15.



Slika 17: Konfiguracija objekta GalilAxis [12]

Vrednosti prikazane na sliki 17, so uporabljene pri našem projektu.

Ime	Opis
AllowRemoteCbox	Uporaba daljinskega upravljalnika za premikanje motorjev. Tega nismo uporabili.
AxisEncoderType	Tip kodirnika. 0 – Ni kodirnika 1 – Rotacijski kodirnik 2 – Absolutni kodirnik **Motorji Gap bodo tu imeli številko 1, medtem ko bodo motorji Phase imeli številko 2.**
AxisInitPosition	Začetna pozicija motorjev. Privzeto 0.0
AxisInitType	Način določanja začetne pozicije. Pri nas je to ukaz DP, ki pomeni, da se vzame absolutna vrednost iz kodirnikov brez nobenega začetnega premikanja. Obstajajo druge možnosti, ki premikajo motor do limitnih stikal in s tem določijo začetno pozicijo, ampak pri nas to ni

	potrebno oziroma je celo nezaželeno.
AxisInitVelocity	Velja samo v primeru, če ne uporabimo ukaz DP pri prejšnji lastnosti. Določa maksimalno hitrost motorja pri iskanju začetne pozicije.
AxisNumber	Oznaka motorja na napravi DMC. Pri nas so oznake od A do D.
AxisPositionRatio	Kodirniki uporabljajo svojo enoto, ki pa uporabniku ne pove veliko. Zato določimo konstanto, ki nam to vrednost pretvori v uporabniku bolj prijazno. Pri nas smo izbrali mikrometre. Konstanto se izračuna glede na specifikacije kodirnika in je pri motorjih Phase drugačna od prikazane na sliki 17.
EnableMaintenanceMode	Uporablja se samo v primeru koračnih motorjev pri dinamični korekciji.

Tabela 15: Lastnosti objekta GalilAxis [18]

Objekt nam omogoča nastavljanje vseh pomembnih parametrov za motor, premikanje, ustavljanje in ugašanje motorja. Vse funkcionalnosti razen ustavljanja in ugašanja, ki sta posebna ukaza, izvajamo prek atributov opisanih v tabeli 16.

Pri našem projektu so uporabljene enote za razdaljo mikrometri in za čas sekunde.

Ime	Opis
position	Branje nam pove trenutno pozicijo motorja. Pisanje v atribut, pošlje motor na željeno mesto. Premikanje je absolutno. Vpisana vrednosti bo tudi končna pozicija.
acceleration	Povprečni pospešek.
deceleration	Povprečni pojemek.
velocity	Maksimalna hitrost motorja.
accuracy	Dovoljen odstop motorja od zahtevane pozicije.
backlash	Kompenzacija zaradi praznega teka.
offset	Odmik motorja (kalibracija).

Tabela 16: Atributi objekta GalilAxis [18]

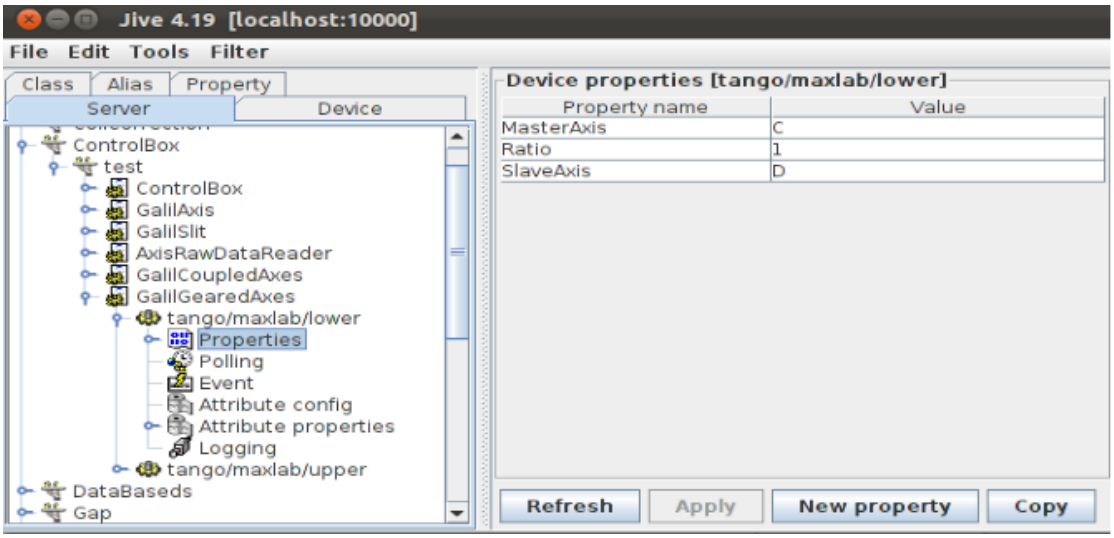
Za vsak atribut lahko nastavimo čas osveževanja. Minimalen je omejen s časom osveževanja objekta ControlBox. Če smo vse nastavitve pravilno izpolnili, lahko zaženemo strežnik ControlBox in poljubno premikamo motorje.

Objekt ima dodatna dva ukaza Jog+ in Jog-, ki premikata motor naprej in nazaj, a je brez daljinskega upravljanja ta funkcionalnost neuporabna in za nas nepomembna.

Sicer je samo s tem objektom praktično nemogoče nadzirati malo kompleksnejše naprave, zato nam strežnik ControlBox ponuja možnost povezovanja motorjev v skupine preko instanc objektov GalilSlit, GalilCoupledAxes in GalilGearedAxes.

6.2.3 Objekt GalilGearedAxes

Omogoča uporabniku, da premika po dva motorja naenkrat. Dodana možnost je tudi določanje razmerja premika med gospodarjem in sužnjem. To pomeni, da se bo en motor premaknil dlje kot drugi za določen faktor. Dve instanci tega objekta smo uporabili pri ControlBox Gap za inženirsko premikanje spodnje in zgornje plošče. Ne moremo ga uporabiti za dejansko premikanje razmaka in faze, ker ne zagotavlja simetričnosti premikanja. Tako bi se lahko zgodilo, da bi en motor zaostajal za drugim in s tem povzročil nagib oziroma nezaželeno obliko magnetnega polja. Načeloma pri inženirskih premikih ni žarka, zato si to lahko privoščimo. Sam objekt je precej enostaven, saj moramo za pravilno delovanje še vedno definirati oba motorja kot instanci objekta GalilAxis in ju prek atributov ter lastnosti pravilno nastaviti. Edina naloga objekta GalilGearedAxes je ta, da poveže dva motorja skupaj in omogoči uporabniku spreminjanje načina premikanja med individualnim ali skupnim.



Slika 18: Konfiguracija objekta GalilGearedAxes [12]

Kot je razvidno iz slike 18, je vse kar je potrebno izpolnit, kateri od motorjev bo gospodar (*MasterAxis*), kateri bo suženj (*SlaveAxis*) in razmerje med njima. Pri nas je bilo razmerje nastavljeno na ena, saj smo se želeli izogniti nepotrebnim nagibom. Povezali smo motorja A in B (zgornja plošča) ter C in D (spodnja plošča) v dve skupini.

Preprostost objekta je razvidna iz njegovih atributov in ukazov. Ima namreč samo dva ukaza, eden vključi in drugi izključi skupno gibanje ter tri atribute opisane v tabeli 17.

Ime	Opis
isGeared	Prikazuje trenutno delovanje motorjev. 0 – Individualno 1 - Skupno
masterPosition	Trenutna pozicija gospodarja.
slavePosition	Trenutno pozicija sužnja.

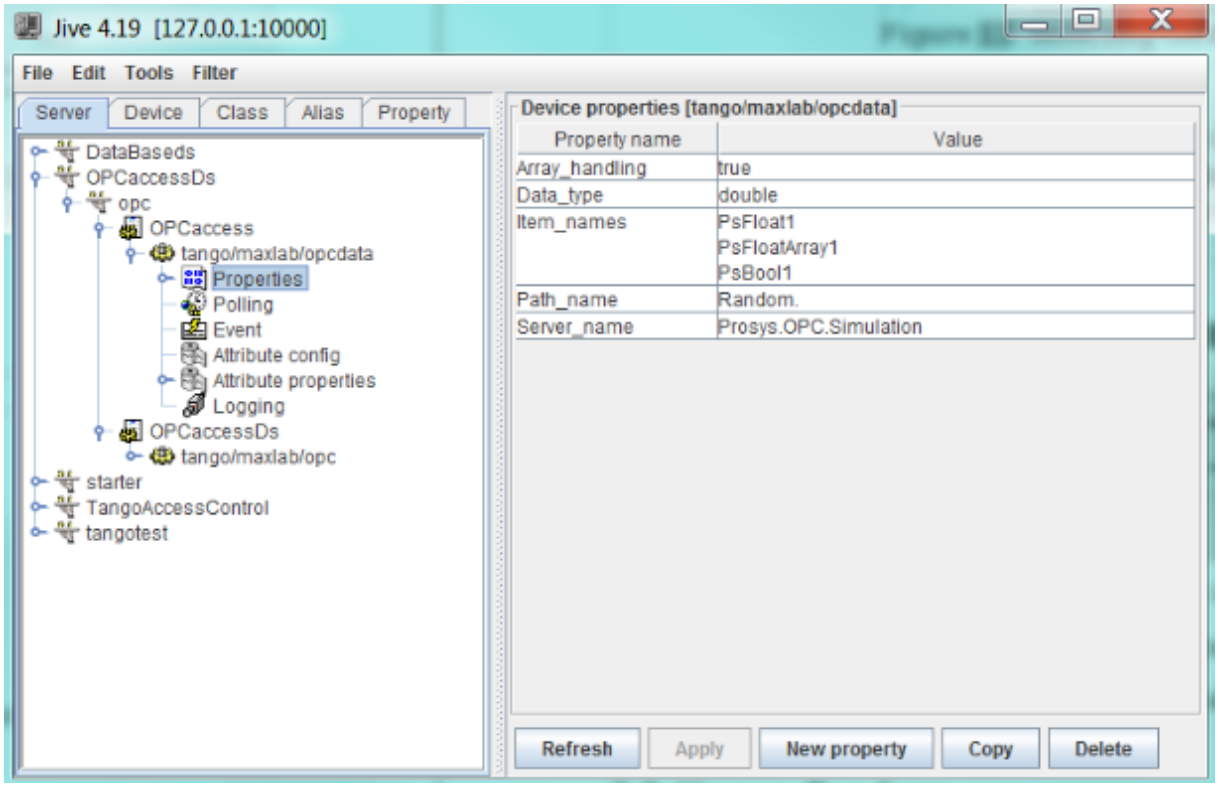
Tabela 17: Atrubiti objekta GalilGearedAxes

Ko smo v individualnem načinu, lahko s pisanjem v pozicijska atributa premaknemo posamezna motorja. V skupnem načinu pišemo samo v atribut masterPosition in s tem premaknemo oba motorja. Pomembno je vedeti, da v primeru, ko imamo pozicijsko napako na enem izmed motorjev, ju ne bo mogoče povezati v skupno delovanje.

6.3 Strežnik OPCaccess

Da sistem deluje pravilno, potrebujemo kar dva strežnika OPCaccess. Eden je namenjen nadzoru korekcijskih tuljav in ga uporablja strežnik Undulator, drugi pa pošilja signale iz stikal in senzorjev med napravami PLC in DMC. Razložil bomo osnovno konfiguracijo, saj je večina parametrov samo odvisna od poimenovanja signalov in vrednosti.

Posebnost strežnika je, da dinamično ustvari atribute ob zagonu, ki predstavljajo predmete na strežniku OPC [19].



Slika 19: Konfiguracija strežnika OPCaccess [12]

Konfiguracija iz slike 19 je bila uporabljena za testiranje strežnika s pomočjo simulatorja Prosys OPC Test Server [20]. Razlaga lastnosti objekta je podana v tabeli 18.

Ime	Opis
Server_name	Ime strežnika OPC.
Path_name	Pot do spremenljivke na OPC strežniku.
Item_names	Imena spremenljivk, definiranih kot tekstovno polje.
Data_type	Izbiramo lahko med tremi tipi spremenljivk. (double, long in short)
Array_handling	Omogočimo strežniku, da dela tudi s polji.

Tabela 18: Lastnosti strežnika OPCaccess [12]

Pravilno konfiguracijo se je določilo, ko smo opravili integracijo celotnega sistema na Švedskem.

Strežnik deluje le na operacijskem sistemu Windows.

6.4 Strežnik Undulator

Do sedaj še nismo dejansko programirali strežnika TANGO, saj smo uporabili že narejene strežnike in si s tem olajšali delo. Vso znanje pridobljeno iz testiranja in konfiguriranja prejšnjih dveh opisanih strežnikov, bomo zdaj uporabili za razvoj strežnika Undulator. Razumevanje delovanja in zmogljivosti strežnika ControlBox in OPCaccess je ključnega pomena, saj tako preprečimo brez potreben razvoj podobnih oziroma enakih funkcionalnosti ali celo napačne uporabe ponujenih funkcij. To bi kasneje lahko pripeljalo do nepravilnega delovanja ali celo do nesreče in s tem škode na napravi.

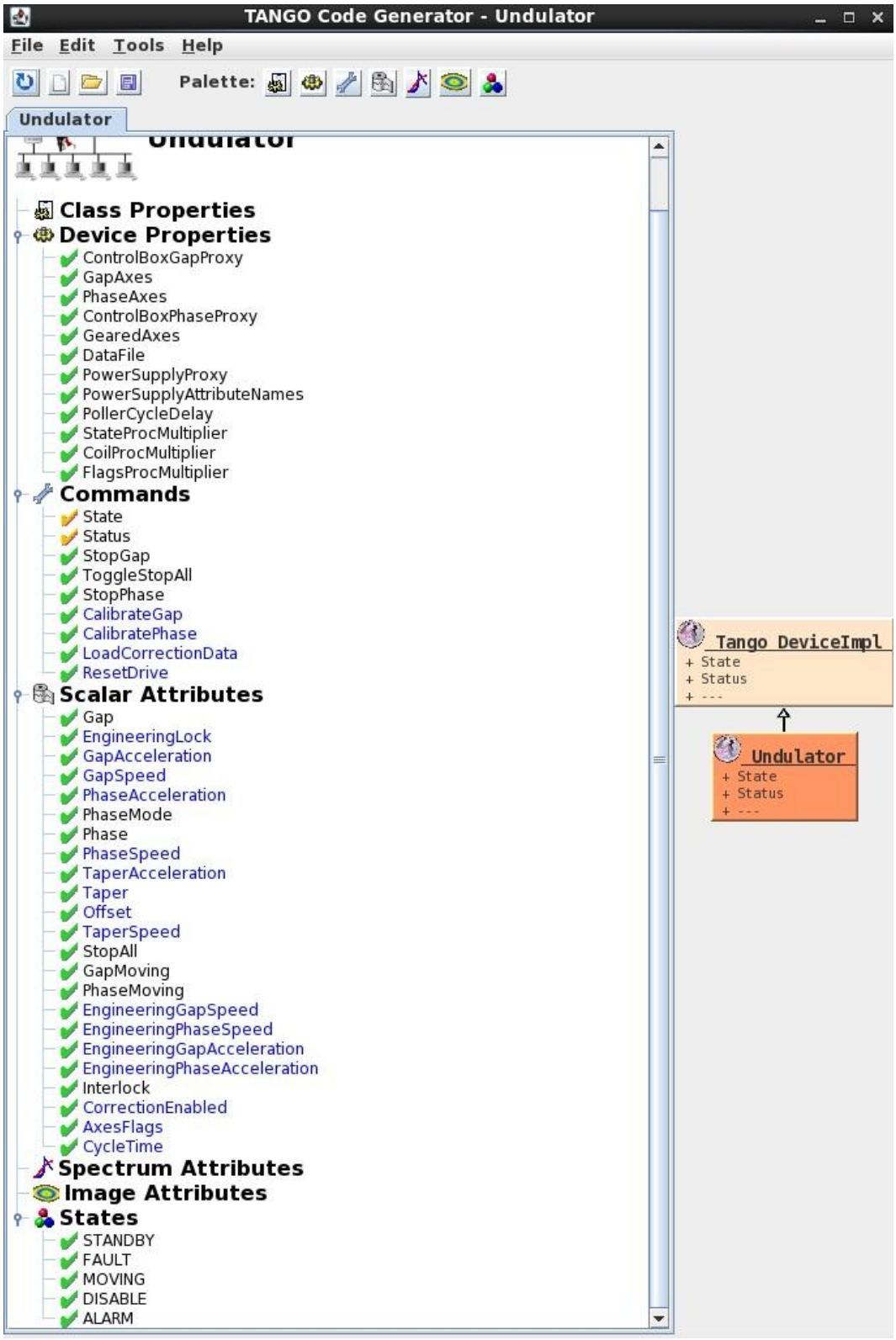
6.4.1 Pregled

Strežnik Undulator ponuja visoko nivojski vmesnik za lažje izvajanje premikanja in konfiguriranja motorjev ter diagnosticiranja vseh komponent undulatorja in korekcijskih tuljav. Izkorišča funkcionalnosti strežnika ControlBox za povezovanje do naprav DMC in za nadzor nad motorji. Prek strežnika OPCaccess nadzoruje tok za korekcijske tuljave.

Glede na stanja vseh komponent, določi trenutno stanje sistema. Ponujene funkcionalnosti so odvisne od trenutnega stanja sistema, predvsem gre tukaj za pisanje v atribute in izvajanje ukazov. Da omogočimo lažje diagnosticiranje, je branje podatkov vedno dovoljeno.

Razvoj začnemo v programu POGO, ki nas pripelje do rezultata prikazanega na sliki 20. Seveda na začetku razvoja strežnik ni izgledal tako, temveč so se elementi dodajali in spreminjali po potrebi. Z elementi tukaj mislimo na lastnosti, ukaze, atribute in stanja. Program POGO ima lepo lastnost, da nam ohranja vso našo kodo, ko spreminjamo, dodajamo

ali brišemo elemente. Tako lahko strežnik gradimo počasi in mu funkcionalnosti dodajamo, ko osnovne že delujejo. Vseeno je potrebno na začetku razvoja dodobra premisliti o arhitekturi sistema, saj se nam lahko hitro zgodi, da stvari uidejo izpod nadzora.



Slika 20: Izdelava strežnika Undulator v programu POGO

6.4.2 Lastnosti

Lastnosti pri strežnikih TANGO zamenjujejo datoteke za konfiguracijo, ki jih večina drugih programov uporablja. Tukaj definiramo vse povezave in konstante, ki jih bo strežnik Undulator uporabil za svoje delovanje. Vse lastnosti iz tabele 19 razen ene (DataFile) morajo biti pravilno izpolnjene pred zagonom strežnika, v nasprotnem primeru se bo ob zagonu postavil v stanje napake.

Ime	Opis
ControlBoxGapProxy	Povezava do strežnika ControlBox Gap.
GapAxes	Povezava do posameznih instanc objektov GalilAxis za motorje z oznako Z1 – Z4. Vrstni red je pomemben.
GearedAxes	Povezava do posameznih instanc objektov GalilGearedAxes za skupini motorjev Z1+Z2 in Z3+Z4. Vrstni red je pomemben.
ControlBoxPhaseProxy	Povezava do strežnika ControlBox Phase.
PhaseAxes	Povezava do posameznih instanc objektov GalilAxis za motorje z oznako X1 – X4. Vrstni red je pomemben.
PowerSupplyProxy	Povezava do strežnika OPCaccess, ki komunicira z napravo PLC za nadzor korekcijskih tuljav.
PowerSupplyAttributeNames	Imena atributov na strežniku OPCaccess, ki predstavljajo tok na tuljavah.
DataFile (opcijsko)	Pot do datoteke, ki hrani vrednosti za določanje toka na tuljavah s pomočjo interpolacije.
PollerCycleDelay	Zakasnitev niti za pridobivanje podatkov v milisekundah. Priporočena vrednost je 20.
CoilProcMultiplier	Določa, na koliko ciklov niti se bo izvedlo procesiranje podatkov za korekcijske tuljave. Priporočena vrednost je 1.
FlagsProcMultiplier	Določa, na koliko ciklov niti se bodo posodobile zastavice. Priporočena vrednost je 25.
StateProcMultiplier	Določa, na koliko ciklov niti se bo posodobilo stanje strežnika. Priporočena vrednost je 50.

Tabela 19: Lastnosti strežnika Undulator [12]

6.4.3 Atributi

Večino delovanja se pri strežnikih TANGO izvaja prek atributov. Atribute delimo na dva dela in sicer na tiste, ki so namenjeni vsem uporabnikom in na tiste za inženirje. Ta delitev je

narejena zaradi boljše preglednosti in ne preprečuje uporabniku, da se označi kot inženir in s tem spreminja konfiguracijo.

Poznamo tri tipe atributov, prve lahko samo beremo, v druge samo pišemo in tretje, ki podpirajo branje in pisanje. Slednji so pri nas najbolj pogosti. Atributi, ki predstavljajo zastavice, so bralni. Pri nas imamo vse atribute skalarne (enodimenzionalne vrednosti) in se zato z ostalima dvema vrstama ne bomo ubadali.

Hitrost osveževanja atributov iz tabele 20 lahko določi uporabnik sam. Pri izdelavi podpore za korekcijske tuljave smo testirali zmogljivost in teoretično je možno atribut osvežiti na tri milisekunde. Toliko namreč potrebuje komunikacija z napravo DMC, a smo ugotovili, da se vrednosti spremenljivk na napravi DMC ne posodobijo tako hitro in bi s tem samo brez potrebno obremenjevali sistem. Vrednosti za korekcijske tuljave se tako posodobijo na dvajset milisekund, atributi za zastavice na pol sekunde in vse ostalo vsako sekundo.

Uporabljene enote za razdaljo so mikrometri in za čas sekunde, razen če ni napisano drugače.

Ime	R/W tip	Opis
Gap	READ_WRITE	Trenutni razmak. Pisanje v atribut izvede premikanje razmaka upoštevajoč ostale parametre.
EngineeringLock	READ_WRITE	S pisanjem enice v atribut onemogočimo premikanje motorjev s pisanjem v atribut Gap ali Phase. Poskus pisanja v ta dva atributa sproži obvestilo.
GapAcceleration	READ_WRITE	Povprečen pospešek pri premikanju razmaka.
GapSpeed	READ_WRITE	Maksimalna hitrost pri premikanju razmaka.
Offset	READ_WRITE	Odmik od centra žarka. S pisanjem določimo parameter, ki se bo naslednjič uporabil pri premikanju razmaka.
PhaseAcceleration	READ_WRITE	Povprečen pospešek pri premikanju faze.
PhaseMode	READ_WRITE	Trenutna faza. S pisanjem v atribut določimo fazo pri naslednjem premikanju.
Phase	READ_WRITE	Fazni odmik. S pisanjem v atribut izvedemo premikanje glede na določene parametre.
PhaseSpeed	READ_WRITE	Maksimalna hitrost pri premikanju faze.
TaperAcceleration	READ_WRITE	Povprečen pospešek pri premikanju nagiba.

Taper	READ_WRITE	Trenutni nagib. S pisanjem določimo nagib, ki se bo uporabil pri naslednjem premikanju razmaka.
TaperSpeed	READ_WRITE	Maksimalna hitrost pri premikanju nagiba.
LimitsDisabled	READ	Zastavica se postavi v primeru, da se programski limiti na napravi DMC izklopijo. Uporablja se pri vzdrževanju.
StopAll	READ	Zastavica za signal Stop all.
GapMoving	READ	Signalizacija premikanja motorjev Gap.
PhaseMoving	READ	Signalizacija premikanja motorjev Phase.
EngineeringGapSpeed	READ_WRITE	Maksimalna hitrost inženirskega premikanja motorjev Gap. Če beremo atribut in dobimo vrednost nič, pomeni da hitrosti motorjev niso nastavljene enako. Potrebno je novo pisanje v atribut.
EngineeringPhaseSpeed	READ_WRITE	Maksimalna hitrost inženirskega premikanja motorjev Phase. Če beremo atribut in dobimo vrednost nič, pomeni da hitrosti motorjev niso nastavljene enako. Potrebno je novo pisanje v atribut.
EngineeringGapAcceleration	READ_WRITE	Povprečen pospešek/pojemek pri inženirskem premikanju motorjev Gap. Če beremo atribut in dobimo vrednost nič, pomeni da pospeški/pojemki motorjev niso nastavljeni enako. Potrebno je novo pisanje v atribut.
EngineeringPhaseAcceleration	READ_WRITE	Povprečen pospešek/pojemek pri inženirskem premikanju motorjev Phase. Če beremo atribut in dobimo vrednost nič, pomeni da pospeški/pojemki motorjev niso nastavljeni enako. Potrebno je novo pisanje v atribut.
Interlock	READ	Zastavica za signal Interlock.
CorrectionEnabled	READ_WRITE	Način izvajanja korekcijskih tuljav. Vrednost 1 pomeni avtomatski, 0 pa ročni način.
AxesFlags	READ	Bitni zapis za napake v komunikaciji, kodirnikih in motorjih.
CycleTime	READ	Zakasnitev niti za pridobivanje podatkov v milisekundah.

Tabela 20: Atributi strežnika Undulator [12]

Kot že omenjeno se dostopnost do atributov spreminja glede na stanje strežnika.

Undulator State Machine

Select Allowed Attributes	STANDBY	FAULT	MOVING	DISABLE	ALARM
Gap (Read)	●	●	●	●	●
Gap (Write)	●	○	○	○	○
EngineeringLock (Read)	●	●	●	●	●
EngineeringLock (Write)	●	○	○	●	●
GapAcceleration (Read)	●	●	●	●	●
GapAcceleration (Write)	●	○	○	●	●
GapSpeed (Read)	●	●	●	●	●
GapSpeed (Write)	●	○	○	●	●
PhaseAcceleration (Read)	●	●	●	●	●
PhaseAcceleration (Write)	●	○	○	●	●
PhaseMode (Read)	●	●	●	●	●
PhaseMode (Write)	●	○	○	●	●
Phase (Read)	●	●	●	●	●
Phase (Write)	●	○	○	○	○
PhaseSpeed (Read)	●	●	●	●	●
PhaseSpeed (Write)	●	○	○	●	●
TaperAcceleration (Read)	●	●	●	●	●
TaperAcceleration (Write)	●	○	○	●	●
Taper (Read)	●	●	●	●	●
Taper (Write)	●	○	○	●	●
Offset (Read)	●	●	●	●	●
Offset (Write)	●	○	○	●	●
TaperSpeed (Read)	●	●	●	●	●
TaperSpeed (Write)	●	○	○	●	●
StopAll (Read)	●	●	●	●	●
GapMoving (Read)	●	●	●	●	●
PhaseMoving (Read)	●	●	●	●	●
EngineeringGapSpeed (Read)	●	●	●	●	●
EngineeringGapSpeed (Write)	●	○	○	●	●
EngineeringPhaseSpeed (Read)	●	●	●	●	●
EngineeringPhaseSpeed (Write)	●	○	○	●	●
EngineeringGapAcceleration (Read)	●	●	●	●	●
EngineeringGapAcceleration (Write)	●	○	○	●	●
EngineeringPhaseAcceleration (Read)	●	●	●	●	●
EngineeringPhaseAcceleration (Write)	●	○	○	●	●
Interlock (Read)	●	●	●	●	●
CorrectionEnabled (Read)	●	●	●	●	●
CorrectionEnabled (Write)	●	○	○	●	●
AxesFlags (Read)	●	●	●	●	●
CycleTime (Read)	●	●	●	●	●

Slika 21: Dostopnost do atributov glede na stanje strežnika

Polni krogi na sliki 21 pomenijo, da je pisanje ali branje atributa v tem stanju možno.

6.4.4 Ukazi

Nekatere funkcionalnosti bi bilo nerodno implementirati s pomočjo atributov, zato smo jih implementirali kot ukaze. Prednost ukazov je v tem, da lahko sprejemajo in vračajo parametre. Prav tako kot atributi se zaradi preglednosti delijo na uporabniške in inženirske.

Dva ukaza (State in Status) sta vedno samodejno implementirana. Prvi nam prikaže trenutno stanje sistema (V premikanju, Napaka, V čakanju,…), drugi pa izpiše sporočilo o trenutnem stanju (Naprava je v premikanju motorjev Phase, Napaka na motorju X1, …). Vsi ukazi strežnika Undulator so predstavljeni v tabeli 21.

Name	Vhodni tipi spremenljivk	Opis
StopGap	DEV_VOID	Ustavi motorje Gap.
ToggleStopAll	DEV_BOOLEAN	Pošlje signal Stop all na napravo DMC in postavi strežnik Undulator v stanje Stop all (vhod 1). Prekine signal Stop all in postavi strežnik nazaj v normalno delovanje (vhod 0).
DisableLimits	DEV_BOOLEAN	Izključi (vhod 1) oziroma vključi (vhod 0) programske omejitve na napravi DMC. Uporablja se pri vzdrževanju.
StopPhase	DEV_VOID	Ustavi motorje Phase.
CalibrateGap	DEVVAR_DOUBLE ARRAY	Izračuna odmike za motorje Gap pri določenem razmaku (vhodna polje 0) in odmiku od centra žarka (vhodna polje 1).
CalibratePhase	DEV_VOID	Izračuna odmike za motorje Phase pri faznem odmiku nič.
LoadCorrectionData	DEV_STRING	Poskuša odpreti datoteko (pot do nje je vhodni parameter) in naložiti podatke za korekcijske tuljave, če so le ti zapisani v pravilnem formatu.
ResetDrive	DEV_USHORT	Pošlje ukaz za ponastavitev vseh motorjev.

Tabela 21: Ukazi strežnika Undulator [12]

Dostopnost do ukazov v odvisnosti od trenutnega stanja prikazuje slika 22.

Select Allowed Commands	STANDBY	FAULT	MOVING	DISABLE	ALARM
StopGap	◉	○	◉	○	○
ToggleStopAll	◉	○	◉	◉	○
StopPhase	◉	○	◉	○	○
CalibrateGap	◉	○	○	◉	○
CalibratePhase	◉	○	○	◉	○
LoadCorrectionData	◉	○	○	◉	○
ResetDrive	○	○	○	◉	◉

Slika 22: Dostopnost do ukazov glede na stanje strežnika

6.4.5 Zanimivosti

Predstavili bomo dve zanimivi rešitvi problemov, ki se pri razvoju podobnih kontrolnih sistemov velikokrat pojavijo.

Med razvojem strežnika smo prišli do trenutka, ko smo imeli kar tri dodatne niti.

- Prva nit je brala trenutne vrednosti parametrov za računanje toka na korekcijskih tuljavah.
- Druga je skrbela za posodobitev zastavic in stanja strežnika.
- Tretja je bila zadolžena za pošiljanje in branje podatkov iz strežnika OPCaccess.

Takšno število niti precej vpliva na kompleksnost programa in poveča možnost napak. Potrebno je bilo zaklepanje podatkov (*mutex lock*), da preprečimo smrtni objem in zagotovimo konsistenco podatkov. Zavedati se moramo, da tudi sistem TANGO uporablja zaklepanje določenih podatkov in kar hitro se zna zgoditi, da pridemo do smrtnega objema.

Odločili smo se, da vse tri niti združimo v eno in s tem zmanjšamo dodatno procesiranje in zaklepanje podatkov. Komunikacijo z strežnikom OPCaccess smo rešili z asinhronim pisanjem in zato nismo potrebovali dodatne niti. Kodo za posodobitev zastavic in spreminjanje stanja smo razdelili v dva dela.

Nova nit se mora izvajati vsaj toliko hitro, kot se je najhitrejša izmed treh. To je bila nit za korekcijske tuljave. Le ta naj bi se izvajala vsaj s hitrostjo 50Hz. Nova rešitev je vključevala števec ciklov izvajanja. En cikel predstavlja izvajanje glavnega dela niti, ki pa sam po sebi nima dodatne logike, razen štetja ciklov in preverjanja ali se bo registrirana funkcija izvedla ali ne. Dodatno zakasnitev niti uporabnik določi s spremenljivko definirano v lastnostih strežnika Undulator. Za dodajanje logike v nit, moramo ustvariti objekt s funkcijo process. Vsa potrebna logika mora biti implementirana v tej funkciji. Pri inicializaciji lahko registriramo objekte v nit z želeno zakasnitvijo izraženo v številu ciklov. Tukaj je potrebno paziti, da je izvajanje logike hitro, saj bi drugače preveč zakasnili funkcije, ki potrebujejo hitro odzivnost. Nit bo nato ob vsakem prehodu izvedla funkcije, ki izpolnjujejo pogoje.

Strežnik Undulator je na začetku razvoja bil precej prepleten z ukazi naprave DMC. Zaradi jasnosti in možnih kasnejših razširitev, smo vse dele kode, ki vsebujejo ukaze naprave DMC ločili v poseben objekt. Sprva smo jasno definirali abstraktni vmesnik, katerega objekt za delo z ukazi deduje. Vmesnik določa vse potrebne funkcije, ki jih strežnik Undulator potrebuje za delovanje.

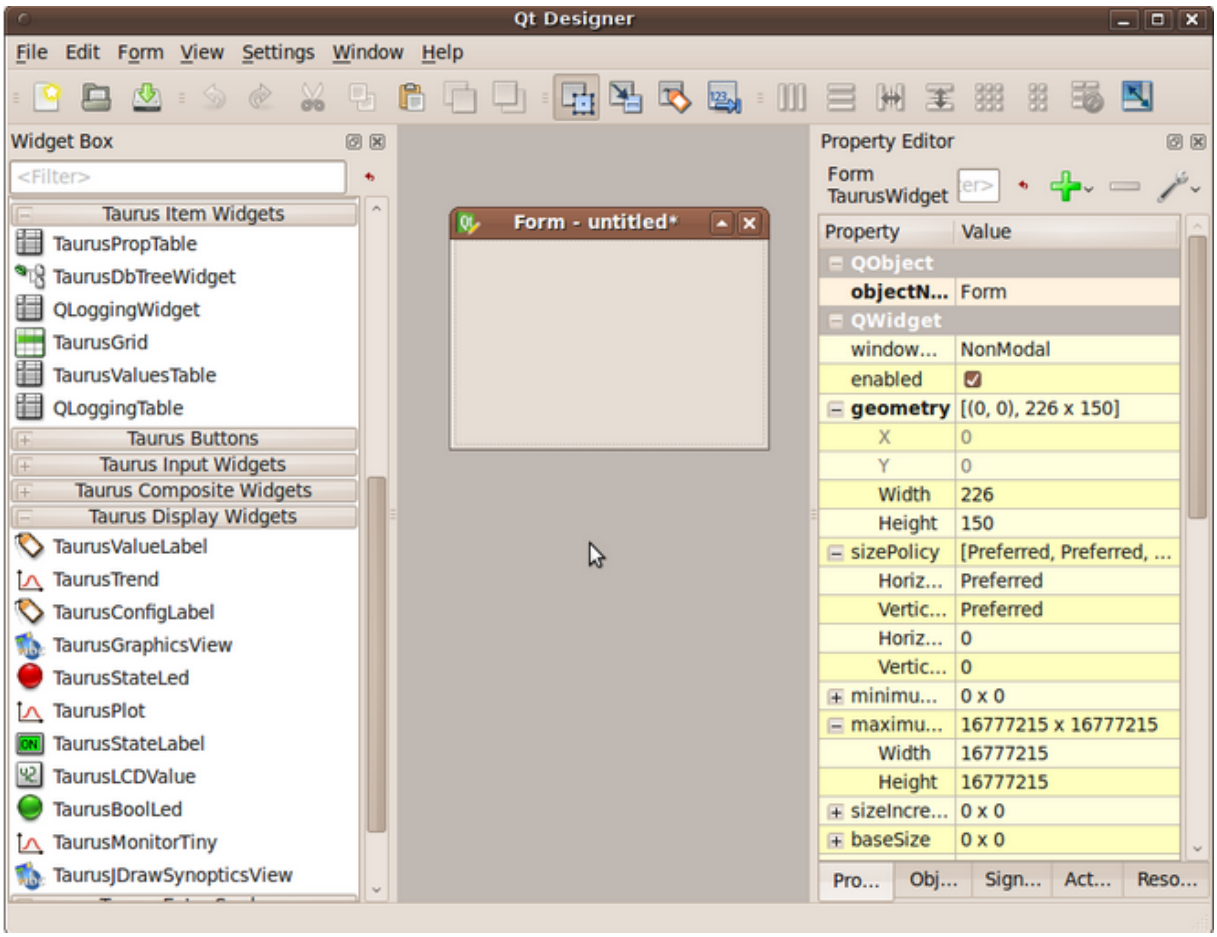
Z implementacijo tovarniškega vzorca (*factory pattern*), lahko ustvarimo poljubni objekt za komunikacijo z napravo DMC. S tem je strežnik Undulator postal neodvisen od vrste naprave DMC. Seveda bomo potrebovali tudi nov ControlBox, če hočemo imeti podprte vse funkcionalnosti.

7 Grafični vmesnik in uporaba

Vsakemu programu pripada svoj grafični vmesnik, ki olajša delo končnemu uporabniku. Strežnik Undulator je možno uporabljati s pomočjo enostavnega grafičnega vmesnika, ki nam ga ustvari program Jive, a je uporaba le tega precej nerodna in nepregledna za vsakdanjo rabo.

7.1 Razvoj

Grafični vmesnik smo razvili s pomočjo programa Taurus Qt Designer. Osnovni izgled programa prikazuje slika 23. Omogoča nam, da vizualno izdelamo vmesnik s pomočjo gradnikov (*widgets*), ki uporabljajo knjižnico Taurus [21].



Slika 23: Taurus Qt Designer [21]

Velik del gumbov in vpisnih polj tako ni potrebovalo dodatne kode. Kodo smo zaradi preglednosti razdelili v dve datoteki. Prva (*undulatorDesigner.py*) je vsebovala avtomatsko generirano kodo, druga (*undulator.py*) pa vso dodatno logiko in vstopno točko za program.

Dodatno kodo smo napisali, ker gradniki Taurus podpirajo le osnovne funkcije kot sta pisanje in branje atributa. Za maše potrebe smo potrebovali pisanje in branje konfiguracije atributov ter spreminjanje vrednosti v uporabniku bolj prijazen format.

Ker strežnik ControlBox omogoča samo absolutno premikanje, smo izdelali logiko, ki v inženirskem načinu, omogoča relativno premikanje motorjev in spreminjanje načina premikanja med individualnim ali skupnim.

Iz signala za napake (binarna oblika) izluščimo vse potrebne zastavice, ki so predstavljene z gradniki LED in opozarjajo na napake v komunikacij in na kodirnikih.

Grafični vmesnik ne prikazuje podatkov o korekcijskih tuljavah, sej se z njimi uporabnik ne ubada.

7.2 Konfiguracija

Preden zaženemo grafični vmesnik, ga je potrebno pravilno nastaviti. Vnaprej ni mogoče vedeti kakšna bodo imena povezav, ki jih potrebujemo, zato smo izdelali dodatno bash skripto (build.sh). Ta zamenja vse povezave do naprav, ki smo jih v kodi poimenovali v naslednjem formatu.

- $ime_naprave

Uporabnik mora pravilno izpolniti setup.opt datoteko z vsemi spremenljivkami definiranimi v tabeli 22.

Ime vrednosti	Povezava
UNDULATOR	Povezava do strežnika Undulator.
GAPZ1	Povezava do instance GalilAxis, ki nadzira motor Z1.
GAPZ2	Povezava do instance GalilAxis, ki nadzira motor Z2.
GAPZ3	Povezava do instance GalilAxis, ki nadzira motor Z3.
GAPZ4	Povezava do instance GalilAxis, ki nadzira motor Z4.
PHASEX1	Povezava do instance GalilAxis, ki nadzira motor X1.
PHASEX2	Povezava do instance GalilAxis, ki nadzira motor X2.
PHASEX3	Povezava do instance GalilAxis, ki nadzira motor X3.
PHASEX4	Povezava do instance GalilAxis, ki nadzira motor X4.
GEARUP	Povezava do GalilGearedAxes, ki nadzira skupno gibanje motorjev Z1+Z2.
GEARDOWN	Povezava do GalilGearedAxes, ki nadzira skupno gibanje motorjev Z3+Z4.
OPC	Povezava do strežnika OPCaccess, ki prenaša signale stikal, merilcev nagiba in signala Interlock.

Tabela 22: Konfiguracija povezav za grafični vmesnik [22]

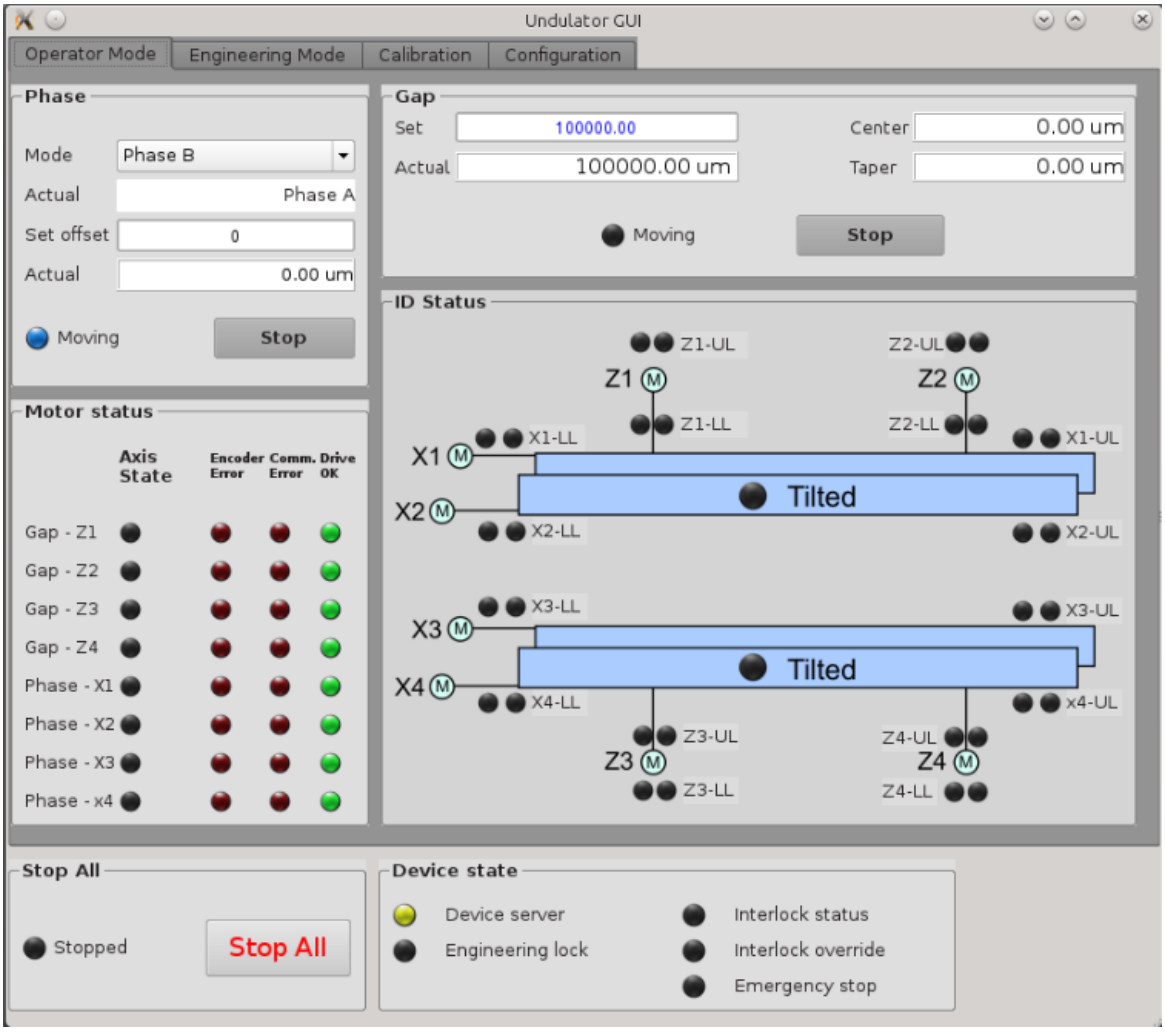
Z zagonom skripte, se ustvari nova mapa v kateri imamo obe prepisani datoteki s kodo in pravilnimi povezavami. Grafični vmesnik zaženemo z ukazom »**python undulator.py**«.

7.3 Zavihki

Grafični vmesnik je razdeljen na štiri zavihke glede na funkcionalnost. Pričakujemo samo izkušene in resne uporabnike, zato nismo implementirali nobene zaščite za spreminjanje nastavitev. Tako ima vsak uporabnik popoln dostop do vseh atributov in konfiguracije.

Zavihke bomo predstavili tako, kot si sledijo v programu.

7.3.1 Upravljalni način



Slika 24: Osnovni pogled uporabniškega vmesnika [22]

Zavihek prikazan na sliki 24 je sestavljen iz več plošč (*panel*). Spodnji dve (Stop all in Device state) sta prikazani v vseh zavihkih.

Plošča Phase omogoča uporabniku spreminjanje faze in odmika. Ko uporabnik izpolni polje za odmik in vrednost potrdi, se premikanje začne. V primeru, da želimo spremeniti fazo, je to potrebno narediti pred pisanjem v odmik. Gumb Stop ustavi vse motorje Phase.

Plošča Gap deluje podobno kot Phase, le da tukaj uporabnik spreminja razmak. Ima možnost spreminjanja nagiba (*Taper*) in odmika od centra žarka (*Center*), a se to le redko potrebuje.

Plošča Motor status prikazuje trenutno stanje motorjev (*Axis State*) in pogonov (*Drive OK*) ter napake na kodirnikih (*Encoder Error*) in komunikaciji (*Comm. Error*).

Ker ima motor več različnih stanj, imajo tudi gradniki LED več barv, kot prikazuje tabela 23.

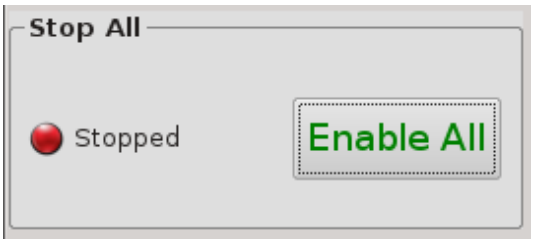
Stanje	Barva in ime
Motor izključen, zavore vključene	**ZAPRTO**
Motor aktiven in v premikanju	**PREMIKANJE**
Motor aktiven in v pripravljenosti	**V PRIPRAVLJENOSTI**
Napaka na motorju oziroma strežniku	**NAPAKA**
Alarm na strežniku (stikalo, napačna pozicija, …)	**ALARM**
Neznano stanje	**NEZNANO**

Tabela 23: Barve gradnika LED za stanja motorja [22]

Ostali gradniki LED se samo prižgejo v svoji barvi (rdeča ali zelena).

Plošča ID Status prikazuje poenostavljeno sliko undulatorja z vsemi stikali. Ob aktivaciji stikala se obarva ustrezni gradnik LED. Notranji predstavljajo mehka limitna stikala (obarvajo zeleno), med tem ko zunanji predstavljajo varnostna ubij limitna stikala (obarvajo rdeče).

Plošča Stop All ima samo en gumb, ki se ob pritisku spremeni. Na sliki 24 vidimo primer, ko je undulator v normalnem stanju. Ob kliku na gumb Stop All bo undulator prešel v stanje Stop all in plošča bo izgledala kot je prikazano na sliki 25.



Slika 25: Plošča Stop All, ko je strežnik Undulator v stanju Stop All [22]

Uporabnik lahko Undulator postavi v normalno stanje s klikom na gumb Enable All.

Plošča Device state prikazuje trenutni status strežnika Undulator (Device server) in sledi barvni shemi iz tabele 24.

Stanje	Barva in ime
Undulator v pripravljenosti. Lahko izvedemo premikanje.	**V PRIPRAVLJENOSTI**
Vsaj eden od motorjev se premika. Ne moremo izvajati dodatnih premikov, dokler ne preidemo nazaj v stanje pripravljenosti.	**PREMIKANJE**
Napaka na strežniku. Uporabnik naj kontaktira inženirja, da jo odpravi.	**NAPAKA**
Strežnik v alarmu. Napaka ni kritična je pa priporočeno, da se kontaktira inženirja.	**ALARM**
V to stanje preidemo, ko se sproži signal Interlock ali Stop All. Onemogočeno je kakršnokoli gibanje. Potrebno je takoj obvestiti vzdrževalce.	**ONEMOGOČEN**

Tabela 24: Barvna shema za stanja strežnika Undulator [22]

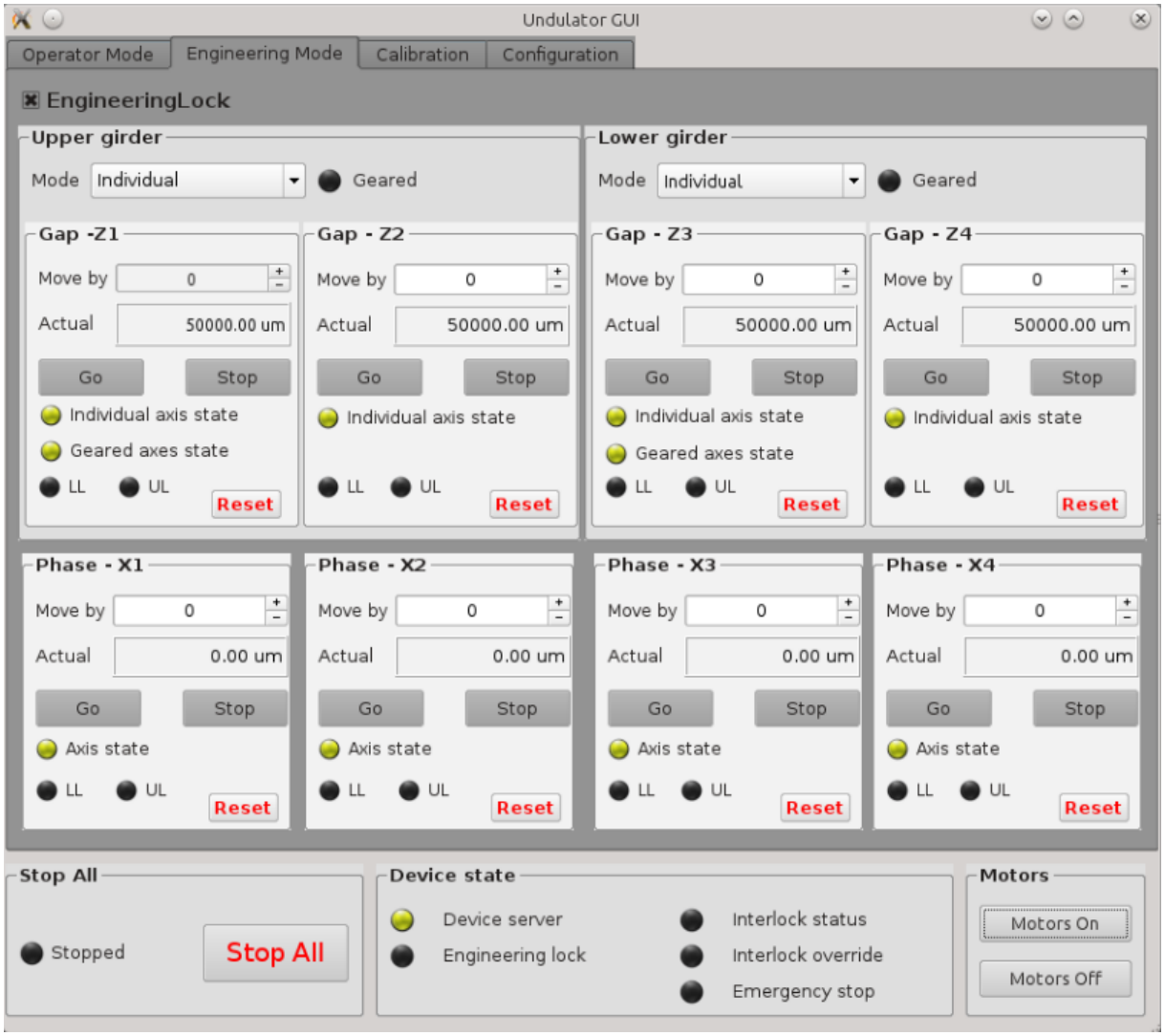
Engineering lock je posebno stanje, ki dovoli samo inženirske premike. Tako preprečimo, da bi uporabnik nehote izvedel premikanje faze ali razmaka. Uporablja se pri vzdrževanju.

Signala Interlock in Interlock Key Override sta že bila razložena.

Signal Emergency stop je praktično enak signalu Stop all, le da ni izveden programsko, ampak s stikalom (prikazanim na sliki 26) nameščenim na ogrodju undulatorja.

Slika 26: Stikalo Emergency stop [22]

7.3.2 Inženirski način

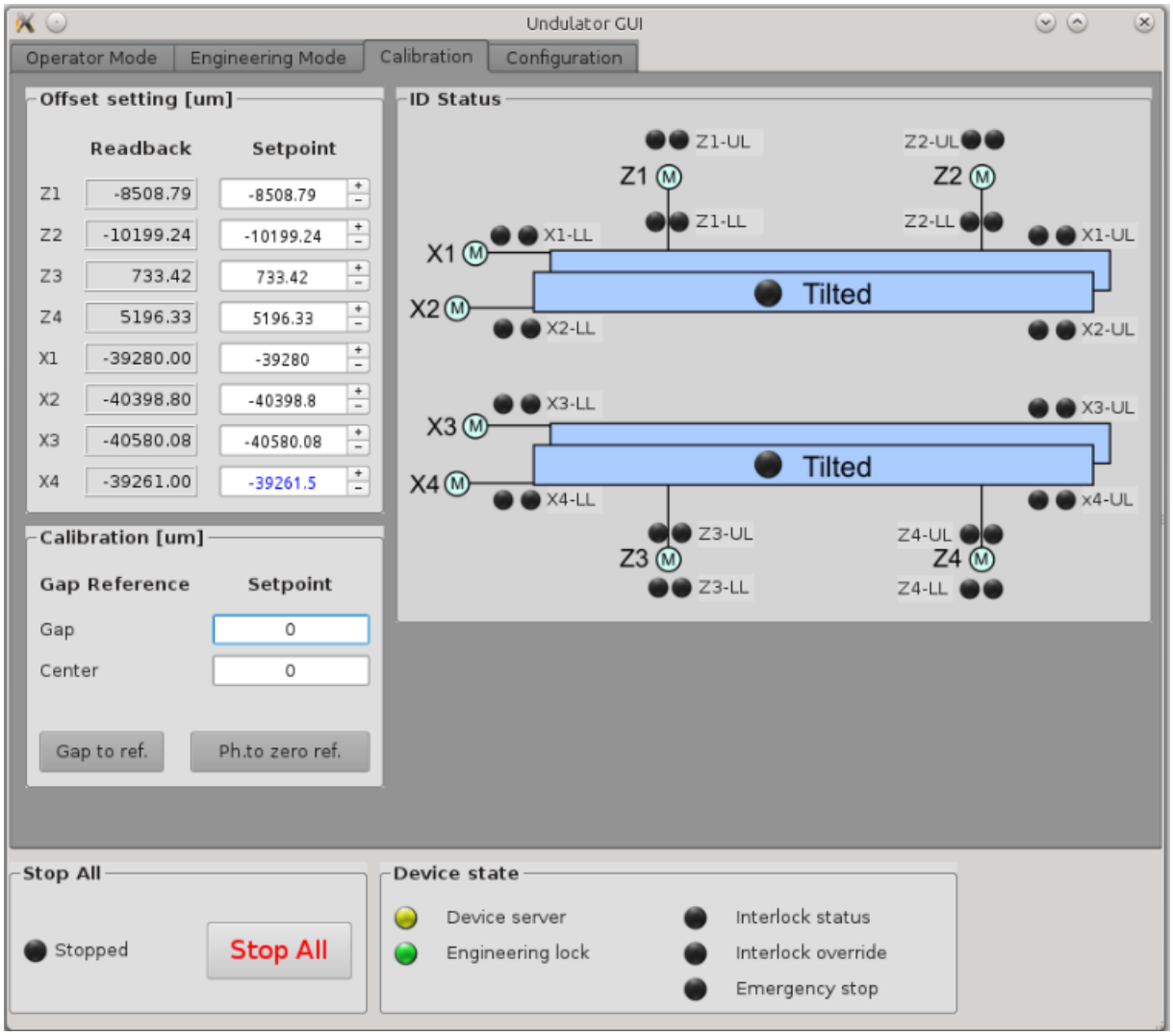


Slika 27: Zavihek za inženirska premikanja [22]

Zavihek prikazan na sliki 27 je namenjen premikanju posameznih motorjev ali plošč. Za razliko od premikanja razmaka in faze se tukaj gibanje ne začne po potrjeni vrednosti, ampak je potrebna dodatna potrditev s klikom na gumb Go. Gre za relativna premikanja, torej uporabnik vstavi vrednost za katero se bo motor premaknil v določeno smer. Ko imamo skupno povezane motorje, gradniki LED za individualno stanje motorja (*Individual axis state*) ne predstavljajo več realnega stanja in jih lahko ignoriramo. Ob napaki lahko poskušamo s ponastavitvijo posameznega motorja s pritiskom na gumb Reset, če se napaka ne odpravi, je potreben pregled strojne opreme.

Po želji se lahko vse motorje ugasne oziroma prižge.

7.3.3 Kalibracija



Slika 28: Zavihek za kalibracijo [22]

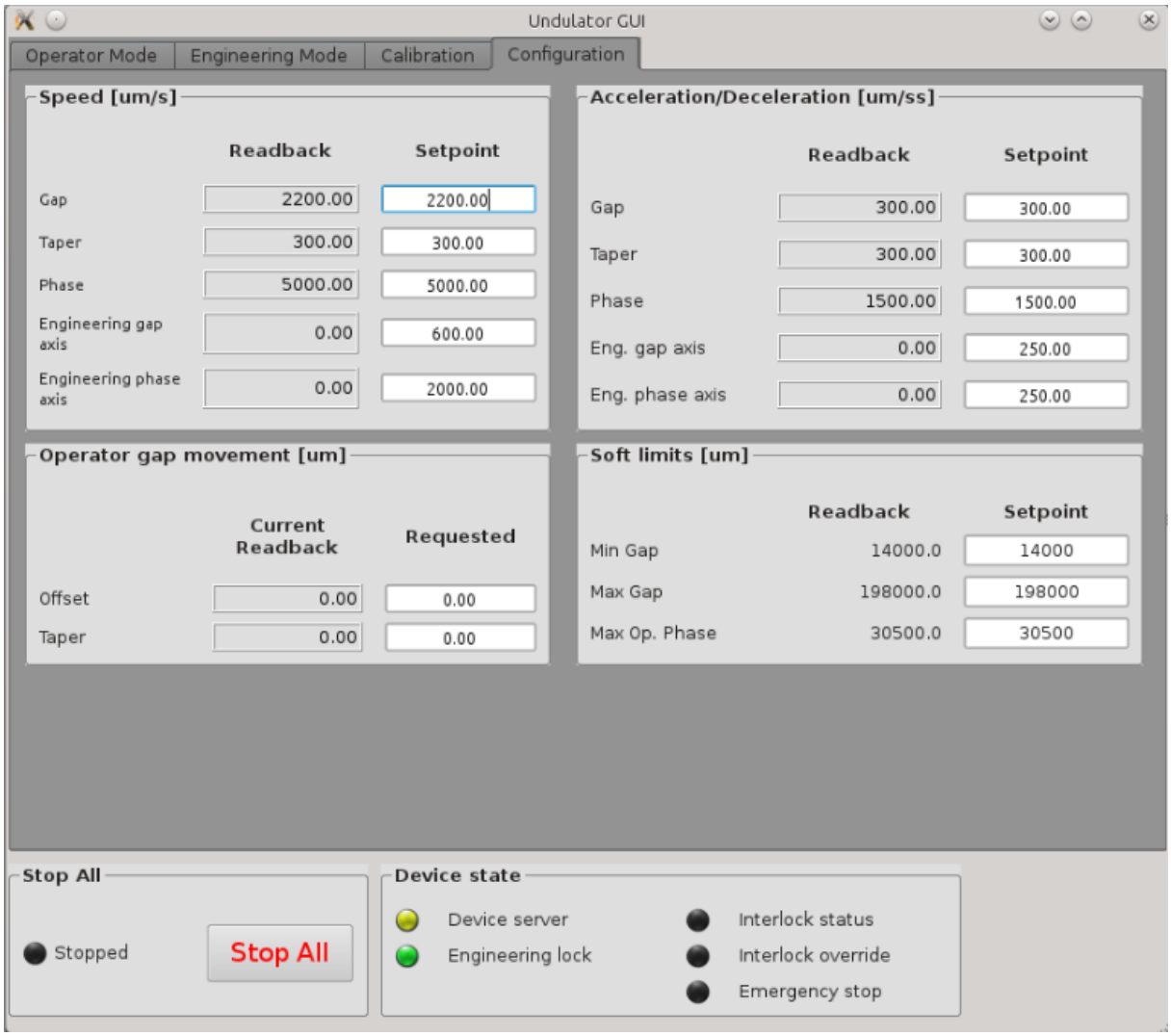
Zavihek prikazan na sliki 28 je namenjen za kalibracijo in naj bi se uporabljal zelo redko, v idealnih pogojih samo ob prvi nastavitvi undulatorja. Za vsak motor lahko nastavimo odmik, a ker je to preveč zamudno oziroma lahko pripelje do napak, je priporočena uporaba posebnih funkcij za kalibracijo, ki jih ponuja strežnik Undulator.

Inženir poravna vse štiri vrste z magneti, tako da je fazni odmik enak nič in pritisne na gumb Ph. To zero ref.. Izračunali se bodo potrebni odmiki za motorje Phase.

Za kalibracijo motorjev Gap je potrebno natančno izmeriti razmak in odmik od centra žarka ter se prepričati, da je nagib obeh plošč nič, šele nato lahko pritisnemo na gumb Gap to ref..

Ko sta obe kalibraciji uspešno opravljeni, se lahko uporaba undulatorja začne.

7.3.4 Konfiguracija



Slika 29: Zavihek za konfiguracijo [22]

Zavihek prikazan na sliki 29 je namenjen spreminjanju vseh pomembnih parametrov za delovanje motorjev. Večina polj je povezanih z atributom na strežniku Undulator.

Ploščo Soft limits ne smemo zamešati s stikali za mehki limit, predstavlja namreč lastnosti (minimalna in maksimalna vrednost) atributa za razmak in fazni odmik na strežniku Undulator.

8 Zaključek

Kljub temu, da sam kontrolni sistem ni ogromen, je preteklo od začetka projekta do končne namestitve na Švedskem devet mesecev. Nekaj časa smo izgubili na čakanju opreme in zamudi pri postavitvi undulatorja, velik del pa se je porabil za testiranje in iskanje optimalnih rešitev, saj mora kontrolni sistem biti, kar se le da stabilen. To nam je tudi uspelo in smo januarja 2013 kontrolni sistem integrirali v laboratoriju Max-Lab, kjer je uspešno prestal vse tovarniške in varnostne teste.

Trenutno se sistem uporablja za nadzor undulatorja na manjšem sinhrotronu. V prihodnosti se ga bo uporabilo tudi na veliko večjem, ki je sedaj še v izgradnji, poimenovanem MAX IV [23].

Čeprav je v svetu kontrolnih sistemov za pospeševalnike najbolj priznan EPICS, se je kontrolni sistem TANGO izkazal kot dobra alternativa. Z njim nismo imeli posebnih težav in tudi očitki, da je precej nestabilen, so se izkazali za neresnične. Kompleksnost razvoja pa je bila vsaj enaka, če ne celo lažja.

Literatura

[1] (2013) Experimental Physics and Industrial Control System, Dostopno na:
http://www.aps.anl.gov/epics/

[2] (2013) TANGO, Dostopno na:
http://www.tango-controls.org/

[3] J. Coquet, Système de contrôle d'axes GalilV2 : ControlBox, Maj 2007

[4] (2013) SOLEIL, Dostopno na:
http://www.synchrotron-soleil.fr/

[5] (2013) European Synchrotron Radiation Facility, Dostopno na:
http://www.esrf.eu/

[6] (2013) Common Object Request Broker Architecture, Dostopno na:
http://en.wikipedia.org/wiki/Common_Object_Request_Broker_Architecture

[7] (2013) Interoperable Object Reference, Dostopno na:
http://en.wikipedia.org/wiki/Interoperable_Object_Reference

[8] (2013) Tango Workshop – ICALEPCS 2011 presentation Tango Basics, Dostopno na:
http://www.tango-controls.org/tutorials/workshop-2011/at_managed_file.2011-10-12.5045470333

[9] (2013) Undulator, http://en.wikipedia.org/wiki/Undulator

[10] (2013) Slika undulatorja, Dostopno na:
http://www.isa.au.dk/photos/astrid/undulator/undulator2-2009.jpg

[11] M. Pavleski, MAX IV EPU-61 Undulator Control System Design Document, Avgust 2012

[12] T. Humar, M. Pavleski, EPU-61 Undulator Control System Technical Documentation, Januar 2013

[13] (2013) DMC-40x0 User Manual, Dostopno na:
www.galilmc.com/support/manuals/man40x0.pdf

[14] (2013) Microcode, Dostopno na:

http://en.wikipedia.org/wiki/Microcode

[15] (2013) GalilTools, Dostopno na:

http://www.galilmc.com/products/galiltools.php

[16] (2013) PID controller, Dostopno na:

http://en.wikipedia.org/wiki/PID_controller

[17] (2013) Slika programa GalilTools, Dostopno na:

http://www.newmarksystems.com/images/Galiltools-large.jpg

[18] J. Coquet, Système de contrôle d'axes GalilV2, Maj 2007

[19] (2013) OPCaccess, Dostopno na:

http://tango-ds.cvs.sourceforge.net/tango-ds/Communication/OPCaccess/

[20] (2013) Prosys OPC, Dostopno na:

http://www.prosysopc.com/

[21] (2013) Taurus's 3.0 documentation, Dostopno na:

http://www.tango-controls.org/static/taurus/v300/doc/html/index.html

[22] T. Humar, M. Pavleski, EPU-61 Undulator Control System User Guide, Januar 2013

[23] (2013) MAX IV, Dostopno na:

https://www.maxlab.lu.se/

Seznam slik

Seznam tabel