ANALISIS PENGARUH KOORDINAT AKHIR DAN PANJANG LENGAN TERHADAP AKURASI POSISI PADA METODE *INVERSE KINEMATICS* DAN *ITERATIVE*

SKRIPSI

KEMINATAN TEKNIK KOMPUTER

Untuk memenuhi sebagian persyaratan
memperoleh gelar Sarjana Komputer

Disusun oleh:
Farid Aziz Shafari
NIM: 135150301111023

PROGRAM STUDI TEKNIK INFORMATIKA
JURUSAN TEKNIK INFORMATIKA
FAKULTAS ILMU KOMPUTER
UNIVERSITAS BRAWIJAYA
MALANG
2018

PENGESAHAN

ANALISIS PENGARUH KOORDINAT AKHIR DAN PANJANG LENGAN TERHADAP AKURASI POSISI PADA METODE INVERSE KINEMATICS DAN ITERATIVE

SKRIPSI

KEMINATAN TEKNIK KOMPUTER

Diajukan untuk memenuhi sebagian persyaratan
memperoleh gelar Sarjana Komputer

Disusun Oleh :
Farid Aziz Shafari
NIM: 135150301111023

Skripsi ini telah diuji dan dinyatakan lulus pada
06 Juni 2018
Telah diperiksa dan disetujui oleh:

Dosen Pembimbing I	Dosen Pembimbing II
Rizal Maulana , S.T., M.T., M.Sc.	Wijaya Kurniawan, S.T, M.T
NIK: 2016078910091001	NIP: 19820125 201504 1 002

Mengetahui
Ketua Jurusan Teknik Informatika

Tri Astoto Kurniawan, S.T, M.T, Ph.D.
NIP: 19710518 200312 1 001

PERNYATAAN ORISINALITAS

Saya menyatakan dengan sebenar-benarnya bahwa sepanjang pengetahuan saya, di dalam naskah skripsi ini tidak terdapat karya ilmiah yang pernah diajukan oleh orang lain untuk memperoleh gelar akademik di suatu perguruan tinggi, dan tidak terdapat karya atau pendapat yang pernah ditulis atau diterbitkan oleh orang lain, kecuali yang secara tertulis disitasi dalam naskah ini dan disebutkan dalam daftar pustaka.

Apabila ternyata didalam naskah skripsi ini dapat dibuktikan terdapat unsur-unsur plagiasi, saya bersedia skripsi ini digugurkan dan gelar akademik yang telah saya peroleh (sarjana) dibatalkan, serta diproses sesuai dengan peraturan perundang-undangan yang berlaku (UU No. 20 Tahun 2003, Pasal 25 ayat 2 dan Pasal 70).

Malang, 06 Juni 2018

Farid Aziz Shafari
NIM: 135150301111023

KATA PENGANTAR

Segala puji syukur untuk Allah SWT karena atas rahmat dan hidayahNya penulis dapat menyelesaikan skripsi yang berjudul "**Analisis Pengaruh Koordinat Akhir dan Panjang Lengan Terhadap Akurasi Posisi Pada Metode *Inverse Kinematic* dan *Iterative***". Skripsi ini disusun untuk memenuhi sebagian persyaratan memperoleh gelar Sarjana Komputer di Fakultas Ilmu Komputer Universitas Brawijaya Malang.

Melalui kesempatan ini, penulis ingin menyampaikan rasa hormat dan terima kasih penulis yang sebesar – besarnya kepada semua pihak yang telah memberikan bantuan baik lahir maupun batin selama penulisan skripsi ini. Penulis ingin menyampaikan rasa hormat dan terima kasih penulis kepada :

1. Allah yang Maha Esa karena atas kehendak dan nikmat-Nya laporan skripsi ini telah selesai dengan baik.
2. Bapak Tri Astoto Kurniawan, S.T, M.T, Ph.D, selaku Ketua Jurusan Teknik Informatika.
3. Bapak Sabriansyah Rizqika Akbar, S.T, M.Eng, selaku Ketua Program Studi Teknik Komputer.
4. Bapak Rizal Maulana, S.T., M.T., M.Sc. selaku dosen pembimbing I skripsi yang telah memberikan waktu, ilmu dan saran untuk menyelesaikan skripsi ini.
5. Bapak Wijaya Kurniawan, S.T, M.T selaku dosen pembimbing II skripsi yang telah memberikan waktu, ilmu dan saran untuk menyelesaikan skripsi ini.
6. Bapak, Mama, Husna, Qonita, keluarga besar dan orang yang menyayangi penulis atas seluruh nasehat, kasih sayang, doa bantuan serta dorongan semangat kepada penulis dalam pengerjaan tugas akhir hingga selesai.
7. Teman-teman AOT, OHI, KSK-JKT48, #KNGNTRS, Fans Buih yang selalu tiada hentinya memberikan dukungan hingga selesainya tugas akhir ini.
8. Seluruh civitas akademika Teknik Komputer Univeristas Brawijaya yang telah banyak memberikan bantuan dan dukungan selama penulis menempuh studi di Teknik Komputer Universitaas Brawijaya dan selama penyelesaian tugas akhir ini.
9. Dan orang-orang yang selalu mendukung dan mendokan yang tidak dapat diucapakan satu persatu. Terimakasih atas semua doa dan dukungannya.

Penulis menyadari dalam penyusunan tugas akhir ini masih banyak kekurangan dalam penulisan ataupun pada isi laporannya, saran dan kritik yang membangun dapat diberikan kepada penulis guna perbaikan tugas akhir selanjutnya. Semoga tugas akhir ini dapat memberikan manfaat bagi semua pihak.

Malang, 30 April 2018

Penulis

faridazish@gmail.com

ABSTRAK

Salah satu hambatan dalam robotika industri adalah mengurangi besar ketidakakuratan posisi antara posisi akhir robot dengan titik target. Dalam hal ini, besar kesalahan mempunyai pengaruh besar dalam tingkat akurasi sebuah robot. Untuk meningkatkan akurasi, dibutuhkan optimasi dari metode yang sudah ada. Dalam robot industri, dibutuhkan kinematika untuk memperhitungkan sudut yang perlu dicapai untuk menuju posisi yang ditentukan. Maka dari itu akan diketahui kinematika yang dibutuhkan adalah *Inverse Kinematic*, dengan memasukkan posisi yang dituju ke dalam perhitungan maka akan menghasilkan sudut yang harus dicapai oleh masing-masing *joint*. Tetapi, pada kenyataannya, setelah persamaan tersebut menghasilkan sudut masing-masing *joint*, kemudian dimasukkan lagi ke dalam persamaan *forward kinematic*, hasilnya tidak sama dengan target awal. Dari masalah ini, maka dibuatlah metode untuk menyelesaikan masalah tersebut. Metode ini disebut *Cyclic Coordinate Descent* (CCD). Metode ini bekerja dengan pendekatan *iterative*, yaitu menggerakan *joint* satu per satu berdasarkan jarak *end-effector* ke target. Hasil dari percobaan dengan menggunakan robot SCARA dan 2 macam pengujian memperlihatkan bahwa metode CCD mampu mengurangi besar *error* rata-rata antara *end-effector* dengan target hingga 20% lebih kecil dibanding metode *Inverse Kinematic*. Walaupun, metode CCD ini memerlukan waktu perhitungan yang lebih lama 0.14463 detik dibanding *Inverse Kinematic* dikarenakan sifatnya yang *iterative*. Dari hasil percobaan tersebut dapat disimpulkan metode dengan robot SCARA cukup efektif untuk mengurangi nilai *error* dalam akurasi posisi ketika berpindah ke posisi target.

Kata kunci: Robot SCARA, *Inverse Kinematic*, *Iterative*, *Cyclic Coordinate Descent*.

ABSTRACT

One of the obstacles in industrial robotics is reducing errors between robots and target points because it has a great impact in the accuracy of a robot. To improve the accuracy, optimization of existing methods is needed. Kinematics is required by industrial robots to calculate the angle to get the specified position. To find out how much angle needed, Inverse Kinematics method is used by putting the target point into the calculation then the result is the angle to be reached by each joint. But, in reality, after the resulted angle being put again in Forward Kinematic equation, the result is not the same as the initial target. From this problem, Cyclic Coordinate Descent (CCD) methode is created to solve it. CCD method works with iterative approach, which moves the joint one by one based on the distance of the end-effector to the target. Results from experiments using SCARA robot and 2 kinds of testing showed that CCD method could reduce the average error between end-effector and target up to 20% than Inverse Kinematic method. However, CCD method requires 0.14462879 seconds longer than Inverse Kinematic due to its iterative behavior. From the results, it can be concluded that the CCD method with SCARA is effective enough to reduce the accuracy error when moving to the target position.

Keywords : SCARA Robot, *Inverse Kinematic*, *Iterative*, *Cyclic Coordinate Descent*.

DAFTAR ISI

DAFTAR TABEL

DAFTAR GAMBAR

BAB 1 PENDAHULUAN

1.1 Latar belakang

Dalam dunia robotika, robot dapat dibedakan menjadi 2 jenis yakni robot *manipulator* (lengan robot) dan robot mobil (Setiawan, 2015). Robot *manipulator* banyak digunakan dalam dunia industri untuk berbagai tugas, seperti *pick and place*, *filling machine*, dan tugas otomasi lainnya. Dari berbagai jenis robot *manipulator*, robot SCARA dan *cartesian* adalah yang paling mudah untuk diaplikasikan (Brumson, 2001).

Mengutip dari Peter Cavallo, *Manager* dari *Denso Robotics* "Robot SCARA merupakan pilihan pertama para pabrikan karena kecepatan, kekuatan, dan daya tahannya". Dari kutipan itu dapat disimpukan bahwa robot SCARA merupakan pilihan pertama para manufaktur dalam beroperasi. Tetapi, satu dari masalah terbesar pada robotika adalah ketepatan dalam meposisikan *end-effector* ke dalam koordinat *world* melalui sensor sudut *joint*. Biasanya, kesalahan geometrikal dalam parameter geometri bertanggungjawab untuk 90% total kesalahan, sementara 10% sisanya dikarenakan kesalahan non-gemotrikal (Baharin & Hasan, 2007).

Bidang utama dalam robotika adalah kinematika *Robot Manipulator.* Model kinematika menganggap lengan robot merupakan rentetan *joint revolute* atau *prismatic* yang saling terhubung dengan kokoh. Maka, posisi relatif dan orientasi dari 2 *link* berturut-turut dapat ditentukan oleh panjang *link* dan parameter dari *joint.* Identifikasi parameter geometri robot secara langsung relevan dengan masalah kalibrasi robot. Sebuah desain *controller robot manipulator* terdiri atas model kinematika *manipulator.* Masalahnya terletak pada variasi *gemotric* dan *non-geometric.* Dalam banyak contoh, model kinematika mempunyai parameter *geometri* dan *non-geometric*. Maka, kalibrasi kinematika *robot manipulator* sangatlah penting dalam semua prosedur pengontrolan robot. Tujuan utama mengidentifikasi nilai parameter adalah untuk meningkatkan kinerja tugas robot.

Pada robot yang sebenarnya, mengetahui posisi Cartesian dari *end-effector* dan nilai sudut (dari sensor sudut *decoder*) dapat dijadikan model yang dikembangkan dengan metode *iterative* untuk menentukan nilai parameter yang tepat. Parameter kinematika selalu saja bergantung pada *manipulator* dan juga mengalami ketidakpastian dikarenakan kesalahan pada pembuatan.

Oleh karena itu, metode *Iterative* ini akan dibandingkan dengan metode yang sudah sering digunakan dalam industri, yaitu metode *Inverse*. Penelitian ini bertujuan untuk melihat hasil dari perubahan parameter sudut dan panjang lengan yang akan diterapkan pada metode *Iterative* dan *Inverse*. Penilitian ini akan menggunakan aplikasi MATLAB sebagai media simulasi dan perhitungan kinematika nantinya. Dengan melihat hasilnya nanti, akan terlihat

apakah parameter-parameter tersebut berpengaruh dalam pengkalibrasian parameter kinematika seperti yang menjadi tujuan metode *Iterative.* Diharapkan penilitian ini dapat memberikan solusi agar tidak terjadinya masalah dalam menentukan parameter.

1.2 Rumusan masalah

1. Bagaimana menerapkan metode *Inverse* dan *Iterative* ke dalam MATLAB?
2. Bagaimana mengetahui pengaruh panjang lengan dan koordinat akhir pada pergerakan *Robot Manipulator* dengan metode *Inverse* dan *Iterative*?
3. Bagaimana perbandingan metode *Inverse* dan *Iterative* ditinjau dari akurasi posisi dan waktu?

1.3 Tujuan

Tujuan dilakukannya penelitian ini adalah sebagai berikut :

1. Menerapkan metode *Inverse* dan *Iterative* ke dalam MATLAB
2. Untuk mengetahui pengaruh panjang lengan dan koordinat akhir pada pergerakan *Robot Manipulator* dengan metode *inverse* dan *iterative*
3. Untuk mengetahui cara membandingkan metode *iterative* dan *inverse*

1.4 Manfaat

Manfaat yang utama adalah menjelaskan bagaimana metode *Iterative* dapat berpengaruh dalam keakuratan posisi dibanding dengan metode kinematika lainnya. Kemudian, manfaat selanjutnya adalah untuk mencari pengaruh dari parameter sudut dan panjang lengan, karena jika hasil penelitian ini nanti ternyata terbukti berpengaruh, dapat dilakukan optimalisasi pada parameter tersebut untuk akurasi yang lebih baik

1.5 Batasan masalah

Agar pembahasan tidak melebar dari latar belakang dan terfokus, maka diberikan batasan masalah, yaitu :

1. Pada perancangan ditambahkan limitasi dari setiap *joint* pada robot SCARA, sehingga mengakibatkan tidak semua koordinat tercapai.
2. Penelitian ini berupa simulasi, sehingga mungkin akan terjadi perbedaan saat diimplementasikan pada alat.
3. Penelitian ini tidak menyinggung spesifikasi fisika pada robot seperti massa, kecepatan, akselerasi, dll

1.6 Sistematika pembahasan

Uraian singkat mengenai metodologi penelitian pada masing-masing bab adalah sebagai berikut:

BAB I Pendahuluan

Menjelaskan tentang latar belakang, rumusan masalah, tujuan, manfaat, dan sistematika pembahasan dari "Analisis Pengaruh Parameter Sudut dan Panjang Lengan terhadap Akurasi Posisi pada Metode *Invers Kinematics* dan Metode *Iterative*"

BAB II Landasan Kepustakaan

Pada bab ini akan menjelaskan tentang landasan teori yang terkait dengan penelitian. Pada bab ini juga dijelaskan tentang penelitian serupa yang pernah dilakukan.

BAB III Metodologi

Membahas tentang langkah kerja yang dilakukan dalam penulisan diantaranya studi literatur, analisis kebutuhan system, desain sistem

BAB IV Perancangan dan Implementasi

Pada bab ini menjelaskan tentang perancangan skenario penelitian yang akan dikerjakan oleh penulis dalam melakukan penelitian untuk mendapatkan sebuah hasil dari pengujian skenario yang nantinya akan berupa data yang dapat dianalisis

BAB V Pengujian dan Analisis

Pada bab ini akan dijelaskan tentang hasil dari pengujian dari skenario yang telah diuji oleh penulis, dan menggambarkannya dalam sebuah grafik yang dapat dijelaskan sehingga memudahkan dalam analisa data dan pengambilan kesimpulan

BAB VI Penutup

Bagian penutup menjelaskan tentang pengambilan kesimpulan yang telah diuji oleh peneliti sehingga mendapatkan hasil yang efisien dan dapat menjawab dari pertanyaan pada rumusan masalah yang telah dibuat

BAB 2 LANDASAN KEPUSTAKAAN

Pada bab ini akan menjelaskan beberapa tinjauan pustaka yang terkait dengan penelitian sebelumnya yang memiliki metode dan dasar teori yang mendukung dalam penilitian.

2.1 Kajian Pustaka

Tabel 2.1 Tabel kajian pustaka

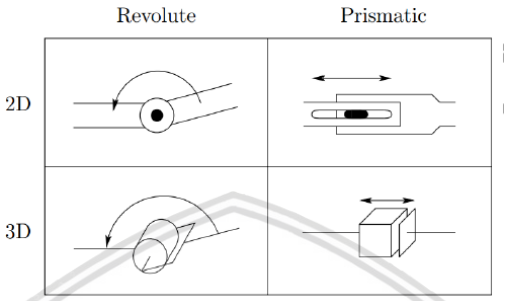
No.	Nama Penulis, Tahun dan Judul	Persamaan	Perbedaan	
			Penelitian Terdahulu	Rencana Penelitian
1	Auralius Manurung. *CCD Algorithm for Solving Inverse Kinematics Problem*	Menguji akurasi posisi metode CCD menggunakan MATLAB	Panjang Lengan mempunyai panjang yang sama. Hanya bisa untuk *Serial Chain.* Visualisasi menggunakan 2 dimensi	Dapat memproses panjang lengan yang berbeda-beda. Dapat memproses *prismatic joint.* Visualisasi berupa simulasi robot bergerak
2.	Iskandar B. Baharin & Md. Mahmud Hasan. 1994. *IDENTIFICATION OF MANIPULATOR KINEMATICS PARAMETERS THROUGH ITERATIVE METHOD*	Melakukan perbandingan *Inverse Kinematics* dengan pendekatan Jacobian dan Metode *Iterative*	Perbandingan ini berupa persamaan, belum diimplementasikan pada simulasi. Persamaan *Inverse* menggunakan Jacobian.	Perbandingan disimulasikan di MATLAB. Persamaan *Inverse* menggunakan pendekatan analisis

2.2 Dasar Teori

2.2.1 Robot Manipulator

Ada 2 jenis robot, yaitu Robot Manipulator (lengan robot) dan Robot Mobil (Eko Setiawan, 2015). Sebuah robot manipulator terdiri atas *Joint* dan *Link*. *Joint* merupakan bagian robot yang mampu membuat pergerakan. Sedangkan *Link* adalah bagian robot yang menghubungkan antar *joint*. Berdasarkan pergerakan robot, *joint* terbagi dalam 2 jenis, yaitu:

1. *Revolute joint,* merupakan penggerak robot yang dapat bergerak memutar. Contoh paling umum dan sering digunakan adalah Motor Listrik.
2. *Prismatic joint*, merupakan penggerak robot yang dapat bergerak maju-mundur (memanjang-memendek). Contoh paling umum dan yang sering digunakan adalah sistem *pneumatic*. Gambar 2.1 menunjukkan representasi *joint* robot



Gambar 2.1 Representasi *Joint* Robot

Sumber : (Setiawan, 2015)

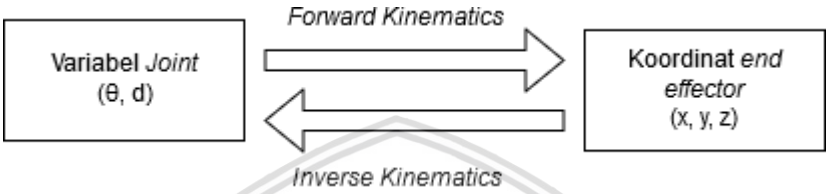
Dalam dunia industri, robot *manipulator* digunakan untuk berbagai macam tugas. Seperti, *pick and place* dan *filling machine.* Gambar 2.2 merupakan contoh robot *manipulator* yang dipakai dalam industri.

Gambar 2.2 Contoh Robot Manipulator, EPSON SCARA LS6

Sumber : EPSON LS6 Product Detail

2.2.2 Inverse Kinematics

Definisi kinematika menurut KBBI adalah "Gerakan geometri suatu titik atau benda padat, diartikan menurut suatu sistem koordinasi ruang". Sesuai dengan definisinya, kinematika akan menentukan gerakan dan koordinat suatu benda, seperti pada Gambar 2.3. Pada kinematika robot, ada istilah yang sering digunakan yaitu *degrees of freedom* (DOF). DOF merupakan derajat kebebasan yang dipunyai robot untuk bergerak dalam satu sumbu. DOF pada *robot manipulator* sama dengan jumlah *joint* pada robot tersebut. Hal ini disebabkan kebebasan robot dalam bergerak ditentukan oleh jumlah *joint*-nya.



Gambar 2.3 Diagram *Inverse* dan *Forward*

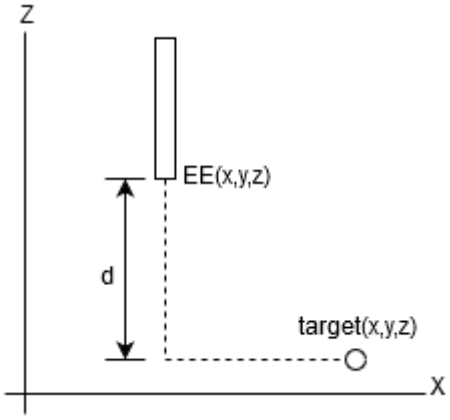
Pada *Inverse Kinematics*, sederhananya adalah mengetahui posisi akhir lengan robot dan tinggal mencari posisi tiap-tiap *joint* agar sesuai dengan posisi akhir lengan robot yang diinginkan. *Inverse* pada SCARA terbagi dalam 2 bagian, *prismatic joint* dan *serial chain*. *Prismatic Joint* menentukan sumbu Z dari posisi *end effector*, sedangkan *serial chain* akan menentukan sumbu X dan Y dari posisi *end effector*. Persamaan *inverse* untuk *prismatic joint* pada robot SCARA adalah

$$d = (d1 - d3) - z \tag{2.1}$$

Keterangan :

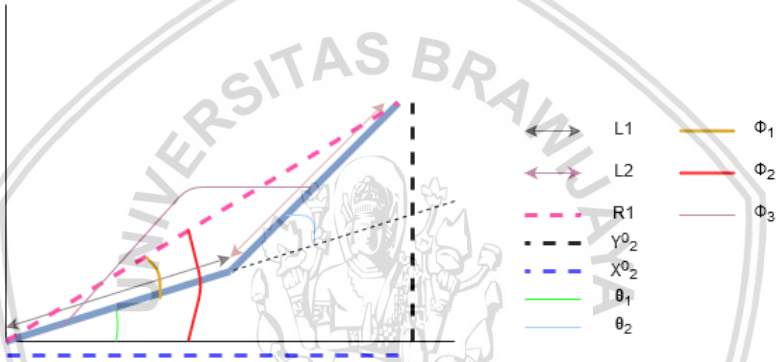
- d1 : Panjang lengan d1.
- d3 : Panjang lengan d3.
- Z : Koordinat Z dari *end-effector* (x,y,z).
- d : variable *joint* dari *prismatic joint*

Dimana Z' merupakan koordinat Z dari target, Z merupakan koordinat Z dari posisi *end-effector* saat itu, dan d merupakan selisih antara koordinat Z' dan Z. Seperti yang digambarkan oleh Gambar 2.4



Gambar 2.4 *Prismatic Joint*

Sedangkan persamaan *inverse* untuk *serial chain* pada robot SCARA dilihat dengan tampak *planar* seperti pada Gambar 2.5



Gambar 2.5 *Serial Chain* robot SCARA

Pada Gambar 2.5, L1 merupakan panjang dari lengan 1, L2 merupakan panjang dari lengan 2, R1 merupakan garis imajiner yang menghubungkan L1 dan L2, Y_2^0 merupakan garis imajiner sumbu Y dari koordinat lokal robot, X_2^0 merupakan garis imajiner sumbu X dari koordinat lokal robot, θ_1 (*theta* 1) merupakan variabel *joint* berupa perubahan sudut dari pergerakan *revolute joint* 1, θ_2 (*theta* 2) merupakan variabel *joint* berupa perubahan sudut dari pergerakan *revolute joint* 2, Φ_1 (*phi* 1) merupakan sudut antara L1 dan R1 untuk membantu perhitungan θ_1, Φ_2 (*phi* 2) merupakan sudut antara R1 dan X_2^0 untuk membantu perhitungan θ_1, dan Φ_3 (*phi* 3) merupakan sudut antara L1 dan L2 untuk membantu perhitungan θ_2. Φ_2 merupakan gabungan antara θ_1 dan Φ_1. Maka persamaannya :

$$\theta_1 = \Phi_2 - \Phi_1 \qquad (2.2)$$

Keterangan :

- θ_1 : perubahan sudut/variabel *joint* dari *revolute joint* 1
- Φ_1 : sudut antara L1 dan R1
- Φ_2 : sudut antara R1 dan X_2^0

Untuk mencari θ_1 seperti pada persamaan 2.2, maka perlu dicari Φ_2 dan Φ_1 dahulu, maka persamaannya :

$$\Phi_2 = \tan^{-1} \frac{Y_2^0}{X_2^0} \quad (2.3)$$

Keterangan :

- $\mathbf{X_2^0}$: garis imajiner sumbu X dari koordinat lokal robot
- $\mathbf{Y_2^0}$: garis imajiner sumbu Y dari koordinat lokal robot
- Φ_2 : sudut antara R1 dan X_2^0

Φ_2 (*phi* 2) merupakan sudut antara R1 dan X_2^0 dan dapat dicari dengan menggunakan rumus trigonometri dari tangen, dengan melihat Φ_2 dalam segitiga R1 - Y_2^0 - X_2^0. Maka persamaannya adalah seperti persamaan 2.3. Sedangkan untuk Φ_1 adalah :

$$R1 = \sqrt{(X_2^0)^2 + (Y_2^0)^2} \quad (2.4)$$

$$\Phi_1 = \cos^{-1} \left(\frac{L2^2 - L1^2 - R1^2}{-2\, L1\, R1}\right) \quad (2.5)$$

Keterangan :

- $\mathbf{X_2^0}$: garis imajiner sumbu X dari koordinat lokal robot
- $\mathbf{Y_2^0}$: garis imajiner sumbu Y dari koordinat lokal robot
- Φ_1 : sudut antara L1 dan R1
- R1 : garis imajiner yang menghubungkan L1 dan L2
- L1 : panjang dari lengan/*link* 1
- L2 : panjang dari lengan/*link* 2

Pada persamaan 2.5, digunakan hukum cosinus pada segitiga R1 – L1 – L2. Nilai R1 pada persamaan 2.4 dapat menggunakan rumus Pythagoras pada segitiga R1 – L1 – L2. Setelah Φ_1 dapat dikerjakan, maka persamaan 2.2 dapat dikerjakan. Setelah θ_1 dapat dikerjakan, maka selanjutnya adalah mencari θ_2.

$$\theta_2 = 180° - \Phi_3 \quad (2.6)$$

Keterangan :

- θ_2 : perubahan sudut/variabel *joint* dari *revolute joint* 2
- Φ_3 : sudut antara L1 dan L2

Untuk mencari θ_2, dapat menggunakan garis L1 yang dipanjangkan. Sehingga, sudut antara garis L1 akan menjadi 180°. Untuk mencari θ_2, maka perlu dicari Φ_3. Φ_3 dapat dicari dengan menggunakan sekali lagi segitiga R1 – L1 – L2 dan hasilnya adalah persamaan 2.7 di bawah ini :

$$\Phi_3 = \cos\text{-}1 \left(\frac{R1^2 - L1^2 - L2^2}{-2\ L1\ L2}\right) \quad (2.7)$$

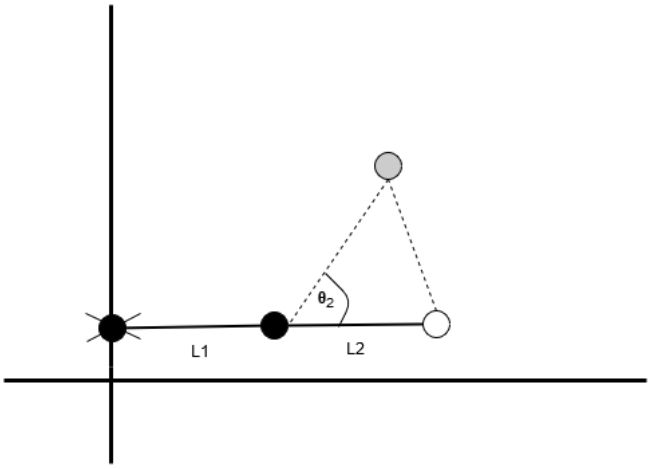
Keterangan :

- Φ_3 : sudut antara L1 dan L2
- R1 : garis imajiner yang menghubungkan L1 dan L2
- L1 : panjang dari lengan/*link* 1
- L2 : panjang dari lengan/*link* 2

2.2.3 Metode *Iterative*

Metode *Iterative* digunakan untuk mengurangi kesalahan posisi hingga sebesar 30% (Iskandar & Hasan, 1994). Metode ini akan meminimalisir kesalahan posisi dan orientasi dengan merubah variable *joint* sekaligus. Setiap iterasi akan menghasilkan satu lintasan dari *manipulator* dari lengan dengan jarak paling jauh menuju *manipulator base*. Sama seperti perancangan *inverse*, perancangan CCD selanjutnya akan dibagi 2 berdasarkan jenis *joint*-nya, yaitu *Prismatic Joint* dan *Revolute Joint*. Dua bagian ini akan dirancang dalam bentuk *planar* untuk mempermudah perhitungan.

Algoritma CCD menggunakan pendekatan heuristik untuk mencari solusi dengan memutar *link* berulang-ulang kali, sehingga *end effector* berpindah mendekati target. Pada bagian *revolute joint,* setiap iterasi akan memutar *link i-th*, dimulai dari *end effector* hingga *base*. Iterasi dilakukan untuk meminimalisasi sudut θ_i diantara vektor dari *link* menuju *end effector* dan vektor dari *link* menuju target, seperti gambar 1.12.



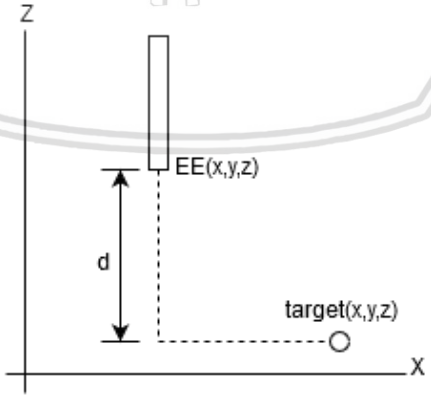
Gambar 2.6 Visual CCD pada *Revolute Joint*

Pada bagian *prismatic joint* tidak berbeda dengan perhitungan dengan *inverse kinematics*. Dimana yang harus dicari adalah nilai d, dimana d merupakan panjang yang harus dicapai agar sejajar dengan titik Z dari koordinat target sesuai dengan gambar 2.7. Maka persamaan d-nya seperti persamaan 2.8 di bawah ini :

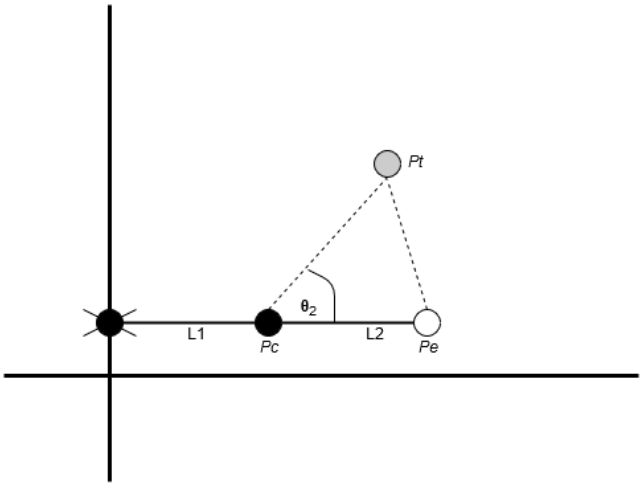
$$d = (d1 - d3) - z \tag{2.8}$$

Keterangan :

- d1 : Panjang lengan d1.
- d3 : Panjang lengan d3.
- Z : Koordinat Z dari *end-effector* (x,y,z).
- d : variable *joint* dari *prismatic joint*

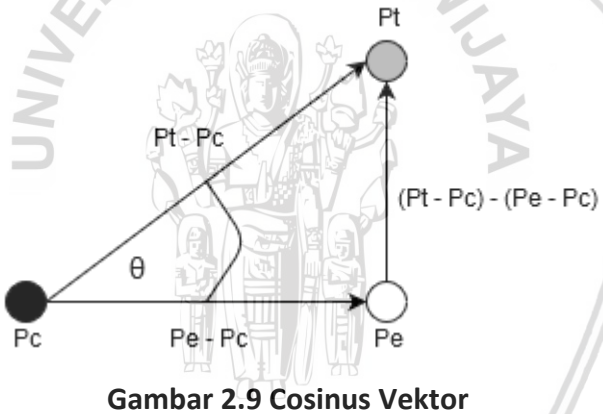


Gambar 2.7 *Prismatic Joint*



Gambar 2.8 Posisi Pc, Pt, Pe

Berdasarkan gambar 2.8, L1 merupakan panjang lengan (*link*) 1, dan diikuti oleh L2 yang merupakan panjang lengan 2. Kemudian, ada Pe yang merupakan koordinat posisi dari *end effector*, Pt merupakan koordinat posisi dari target, dan Pc merupakan koordinat joint dari *link i-th*.



Gambar 2.9 Cosinus Vektor

Untuk mencari θ, maka diperlukan hukum cosinus pada vector. Dengan melihat gambar 2.9, maka persamaannya :

$$\left(\overrightarrow{(Pt - Pc)}.\overrightarrow{(Pe - Pc)}\right) = \left\|\overrightarrow{(Pt - Pc)}\right\| \left\|\overrightarrow{(Pe - Pc)}\right\| \cos\theta \qquad (2.9)$$

Keterangan :

- Pe : titik koordinat dari *end-effector*
- Pc : titik koordinat dari *joint*
- Pt : titik koordinat dari target
- $\overrightarrow{(\boldsymbol{Pe - Pc})}$: vektor dari Pc ke Pe
- $\overrightarrow{(\boldsymbol{Pt - Pc})}$: vektor dari Pc ke Pt
- θ : sudut antara $\overrightarrow{(Pt - Pc)}$ dan $\overrightarrow{(Pe - Pc)}$

Pada persamaan 2.9, vektor Pt – Pc dan Pe – Pc dilakukan perkalian titik (*dot product)* dan hasilnya sama dengan vektor Pt – Pc yang di-*norm*-kan yang

kemudian dikali dengan vektor Pe – Pc yang di-*norm*-kan juga dan kemudian dikalikan dengan cosinus dari θ. Untuk mencari θ, persamaannya adalah :

$$\cos\theta = \frac{\overrightarrow{(Pt-Pc)}}{\left\|\overrightarrow{(Pt-Pc)}\right\|} \cdot \frac{\overrightarrow{(Pe-Pc)}}{\left\|\overrightarrow{(Pe-Pc)}\right\|} \tag{2.11}$$

Keterangan :

- Pe : titik koordinat dari *end-effector*
- Pc : titik koordinat dari *joint*
- Pt : titik koordinat dari target
- $\overrightarrow{(\boldsymbol{Pe}-\boldsymbol{Pc})}$: vektor dari Pc ke Pe
- $\overrightarrow{(\boldsymbol{Pt}-\boldsymbol{Pc})}$: vektor dari Pc ke Pt
- θ : sudut antara $\overrightarrow{(Pt-Pc)}$ dan $\overrightarrow{(Pe-Pc)}$

Pada persamaan 2.11, untuk mencari θ maka cos θ dipindahkan ke sebelah kiri persamaan. Kemudian, vektor Pt – Pc dan Pe – Pc dilakukan perkalian titik (*dot product)* dibagi dengan perkalian dari *norm* masing-masing vektor Pt – Pc dan Pe – Pc.

Setelah mendapatkan sudut, maka perlu dicari arah putarannya. Untuk mencari arah putaran dapat menggunakan *cross product* dari 2 vektor $\overrightarrow{(Pt-Pc)}$ dan $\overrightarrow{(Pe-Pc)}$.

$$\overrightarrow{(Pe-Pc)} \times \overrightarrow{(Pt-Pc)} = \left\|\overrightarrow{(Pe-Pc)}\right\| \left\|\overrightarrow{(Pt-Pc)}\right\| \sin\theta \tag{2.12}$$

Keterangan :

- Pe : titik koordinat dari *end-effector*
- Pc : titik koordinat dari *joint*
- Pt : titik koordinat dari target
- $\overrightarrow{(\boldsymbol{Pe}-\boldsymbol{Pc})}$: vektor dari Pc ke Pe
- $\overrightarrow{(\boldsymbol{Pt}-\boldsymbol{Pc})}$: vektor dari Pc ke Pt
- θ : sudut antara $\overrightarrow{(Pt-Pc)}$ dan $\overrightarrow{(Pe-Pc)}$

Pada persamaan 2.12, vektor Pt – Pc dan Pe – Pc dilakukan perkalian silang (*cross product*) dan hasilnya sama dengan vektor Pt – Pc yang di-*norm*-kan yang kemudian dikali dengan vektor Pe – Pc yang di-*norm*-kan juga dan kemudian dikalikan dengan cosinus dari θ. Besar θ sudah diketahui di persamaan 2.11, tetapi tidak diketahui apakah besarnya minus (berputar ke kiri) atau tidak. Jika θ bernilai minus.

Jika $\overrightarrow{(Pe-Pc)}$ = a dan $\overrightarrow{(Pt-Pc)}$ = b, maka hasil dari *cross product* dalam 3 dimensi adalah :

$$\begin{bmatrix} a_x \\ a_y \\ a_z \end{bmatrix} \times \begin{bmatrix} b_x \\ b_y \\ b_z \end{bmatrix} = \begin{bmatrix} a_y b_z - a_z b_y \\ a_z b_x - b_z a_x \\ a_x b_y - a_y b_x \end{bmatrix} \quad (2.13)$$

Keterangan :

- a_x : komponen x dari vektor a
- a_y : komponen y dari vektor a
- a_z : komponen z dari vektor a
- $\boldsymbol{b_x}$: komponen x dari vektor b
- $\boldsymbol{b_y}$: komponen y dari vektor b
- b_z : komponen z dari vektor b

Persamaan 2.13 merupakan cara pengerjaan dari perkalian vektor a dan b. Dan karena CCD berjalan dalam sistem *planar*, maka *cross product* untuk 2 dimensi :

$$\begin{bmatrix} a_x \\ a_y \end{bmatrix} \times \begin{bmatrix} b_x \\ b_y \end{bmatrix} = \begin{bmatrix} \\ a_x b_y - a_y b_x \end{bmatrix} \quad (2.14)$$

Keterangan :

- a_x : komponen x dari vektor a
- a_y : komponen y dari vektor a
- $\boldsymbol{b_x}$: komponen x dari vektor b
- $\boldsymbol{b_y}$: komponen y dari vektor b

Jika hasil dari *cross product* pada persamaan 2.14 bernilai positif, maka θ akan bernilai positif, dan jika *cross product* bernilai negatif, maka θ akan bernilai negatif.

Setelah mendapatkan besar sudut dan arah putarannya, maka akan dilakukan rotasi menggunakan matriks rotasinya. Persamaan 2.15 merupakan matriks rotasi dalam sistem koordinat 2 dimensi.

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} \quad (2.15)$$

Keterangan :

- x' : titik x hasil rotasi
- y' : titik y hasil rotasi
- $\boldsymbol{x}$: titik x sebelum rotasi
- $\boldsymbol{y}$: titik y sebelum rotasi
- θ : besar sudut rotasi

x dan y pada matriks rotasi menggunakan koordinat lokal (koordinat relatif terhadap *link* sebelumnya) sebagai sistem acuannya. Karena saat rotasi akan menghasilkan koordinat baru, maka harus diubah menjadi koordinat global. sehingga matriks rotasinya akan menjadi persamaan 2.16 :

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix} \left(\begin{bmatrix} x_{i+1} \\ y_{i+1} \end{bmatrix} - \begin{bmatrix} x_i \\ y_i \end{bmatrix} \right) + \begin{bmatrix} x_i \\ y_i \end{bmatrix} \quad (2.16)$$

Keterangan :

- x' : titik x hasil rotasi
- y' : titik y hasil rotasi
- $\boldsymbol{x_i}$: titik x sebelum rotasi dari *link*/lengan $\boldsymbol{i}$
- $\boldsymbol{y_i}$: titik y sebelum rotasi dari *link*/lengan $\boldsymbol{i}$
- x_{i+1} : titik x sebelum rotasi dari *link*/lengan $i + 1$
- y_{i+1} : titik y sebelum rotasi dari *link*/lengan $i + 1$
- θ : besar sudut rotasi

Selain koordinat baru, hasil persamaan 2.10 juga harus diubah menjadi koordinat global. Ini dikarenakan sudut pada θ_i akan mempengaruhi sudut pada link setelahnya (θ_{i+1}) seperti pada persamaan 2.17 di bawah ini :

$$\theta_{i+1} = \theta_{i+1} + \theta_i \quad (2.17)$$

Keterangan :

- θ_{i+1} : besar sudut dari *revolute joint* $i + 1$
- θ_i : besar sudut dari *revolute joint* i

Selain itu, *link* setelahnya (i+2) akan berpengaruh juga karena perpindahan *link* hasil rotasi seperti pada persamaan 2.18 dan 2.19 di bawah ini :

$$x_{(i+2)} = x_{(i+1)} + link_{(i+1)} \quad (2.18)$$

$$y_{(i+2)} = y_{(i+1)} \quad (2.19)$$

Keterangan :

- $x_{(i+2)}$: titik x dari *link*/lengan $i + 2$
- $y_{(i+2)}$: titik y dari *link*/lengan $i + 2$
- $x_{(i+1)}$: titik x dari *link*/lengan $i + 1$
- $y_{(i+1)}$: titik y dari *link*/lengan $i + 1$
- $link_{(i+1)}$: panjang lengan dari lengan/*link* $i + 1$

Untuk X dan Y pada *end effector* tidak perlu penambahan sudut dan koordinat setelahnya karena tidak berpengaruh.

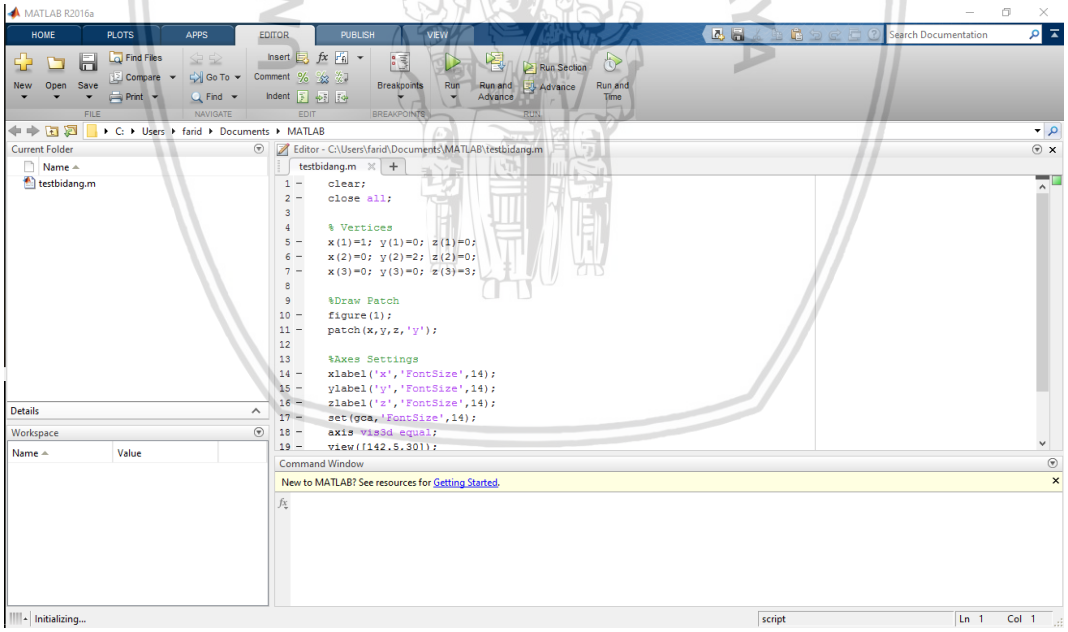
2.2.4 MATLAB

MATLAB adalah platform pemrogramman yang didesain spesifik untuk *engineers* dan ilmuwan. Inti dari MATLAB adalah Bahasa MATLAB, yaitu sebuah bahasa yang berbasis matriks yang mengekspresikan komputasi matematik. (Mathworks, n.d.)

MATLAB merupakan bahasa tingkat tinggi untuk perhitungan teknis. Bahasa ini mengintregasikan komputasi, visualisasi, dan pemrograman yang mudah digunakan dimana masalah dan solusi diekspresikan dalam notasi matematis yang familiar. MATLAB biasanya digunakan untuk :

1. Matematika dan perhitungan
2. Pengembangan algoritma
3. Pemodelan, simulasi, dan pembuatan *prototype*
4. Analisis data, explorasi, dan visualisasi
5. Grafis ilmiah dan keteknikan
6. Pengembangan Aplikasi, termasuk pengembangan GUI

Tampilan pada antarmuka MATLAB terdiri atas jendela *Editor, Command Window, Workspace, dan Current Folder*, seperti pada gambar 2.10.

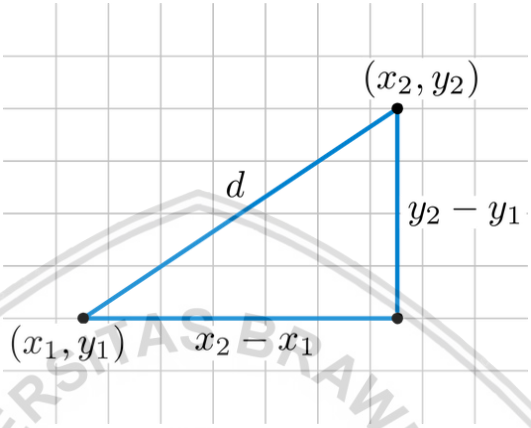


Gambar 2.10 Antarmuka Aplikasi MATLAB

Untuk penggunaan MATLAB pada penelitian ini. Dibutuhkan sebuah *toolbox* yang akan dimasukkan kedalam aplikasi MATLAB. *Toolbox* ini dibutuhkan untuk mendukung penilitian tentang kinematika ini. *Toolbox* dapat diunduh di tautan http://www.petercorke.com/RTB dengan mengisikan nama dan pekerjaan.

2.2.5 Euclidean Distance

Secara geometri, akurasi posisi pada robot dari posisi yang diberikan dapat dicari dengan menghitung jarak antara titik posisi yang diinginkan (Ahmed Joubair, 2014). *Euclidean Distance* adalah metode untuk mengukur panjang sebuah garis lurus antara 2 titik. Metode ini dapat merepresentasikan menghitung jarak antara 2 titik koordinat. Metode ini menggunakan teori segitiga Phytagoras dimana :



Gambar 2.11 Segitiga Phytagoras

Sumber : http://rosalind.info/glossary/euclidean-distance/

Jika titik (x1, y1) dan (x2, y2) pada gambar 2.11 dalam ruang 2 dimensi, maka *Euclidean Distance* antara 2 titik tersebut menjadi persamaan 2.20 :

$$\sqrt{(x2 - x1)^2 + (y2 - y1)^2} \quad (2.20)$$

Dan jika titik (x1, y1, dan z1) dan (x2, y2, dan z2) dalam ruang 3 dimensi, maka *Euclidean Distance* diantara 2 titik tersebut menjadi persamaan 2.21 :

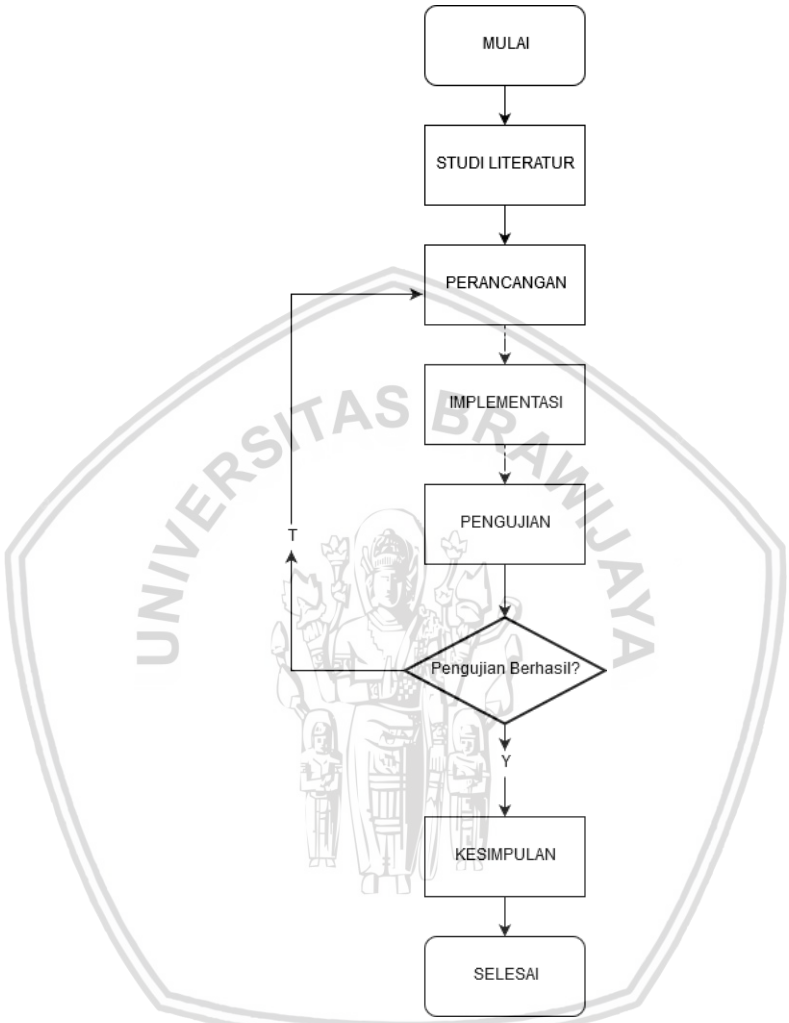
$$\sqrt{(x2 - x1)^2 + (y2 - y1)^2 + (z2 - z1)^2} \quad (2.21)$$

Keterangan :

- $x2$: titik x dari koordinat 2
- $y2$: titik y dari koordinat 2
- $z2$: titik z dari koordinat 2
- $x1$: titik x dari koordinat 1
- $y1$: titik y dari koordinat 2
- $z1$: titik z dari koordinat 1

BAB 3 METODOLOGI

Pada bab ini akan menjelaskan tentang metode yang akan digunakan dalam penelitian, dengan beberapa tahapan yang digambarkan pada diagram alir penelitian dibawah ini .



Gambar 3.1 Diagram Alir Metodologi

3.1 Studi Literatur

Studi literatur merupakan kegiatan persiapan yang dilakukan untuk menentukan objek penelitian yang sesuai dengan tema dan menjadikan sebagai kajian peneliti. Studi literatur menjelaskan tentang beberapa *paper* yang terkait dengan penelitian ini, teori pendukung dapat diperoleh dari jurnal, makalah ilmiah, dan beberapa penelitian sebelumnya yang terkait dengan penelitian yang akan dilakukan oleh peneliti. Studi literatur yang ada pada penelitian ini antara lain mengenai Robot *Manipulator*, *Inverse Kinematics, Iterative Cyclic Coordinate Descent,* MATLAB, dan *Euclidean Distance*.

3.2 Perancangan dan Implementasi

3.2.1 Analisis Kebutuhan

Pada tahapan analisa kebutuhan menunjukan dan menjelaskan kebutuhan yang dibutuhkan oleh penulis dalam melakukan penelitian pada tugas akhir ini. Analisis kebutuhan dibagi menjadi dua kebutuhan yaitu kebutuhan perangkat keras dan perangkat lunak.

3.2.2 Perancangan Simulasi

Perancangan simulasi menjelaskan tentang gambaran umum simulasi secara keseluruhan serta bagaimana perancangan simulasi yang akan dibagun. Perancangan simulasi ini meliputi tentang rancangan bentuk robot dan skenario pengujian yang akan dianalisis. Penjelasan dari bagian perancangan simulasi akan diterangkan pada tahapan tahapan berikut :

a. Rancangan Bentuk Robot

Rancangan bentuk robot dibentuk dari kebutuhan dan batasan dari 2 metode. Agar robot bisa menjalankan metode yang sama, maka robot harus mempunyai kriteria berbentuk *serial chain.* Agar hasil pengujian lebih valid, maka akan ada 2 robot yang diuji. Yaitu, 2 DOF dan 3 DOF.

b. Rancangan Kinematika

Setelah bentuk robot dirancang, kemudian akan dihitung kinematika berdasarkan metode yang dibahas. Pada *Inverse Kinematic* dan *Iterative*, akan dibuat persamaan untuk mencari sudut berdasarkan panjang lengan dan koordinat yang dituju.

c. Rancangan Skenario Pengujian

Penentuan skenario yang akan digunakan untuk pengujian ini terdapat dua skenario dengan menggunakan metode *Inverse Kinematics* dan *Iterative*.

1. Skenario satu, skenario dengan Robot SCARA diberikan 10 koordinat berbeda beda
2. Skenario dua, skenario dengan Robot SCARA diberikan 5 panjang lengan yang berbeda dan koordinat yang berbeda

Penelitian ini fokus terhadap perhitungan akurasi posisi antara *end effector* dengan titik yang diinginkan.

3.2.3 Implementasi

Pada tahap implementasi dijelaskan bagaimana cara mengimplementasi metode hasil rancangan ke dalam MATLAB. Agar bisa menjalankan metode hasil rancangan, diharapkan implementasi dapat menjalankan fungsi memproses persamaan matematika dengan baik hingga mendapatkan keluaran yang sesuai.

3.3 Pengujian dan Analisis

Pada tahapan ini akan dilakukan pengujian sesuai dengan rancangan skenario pengujian yang ada. Pada Pengujian kali ini akan dilihat apakah implementasi berjalan sesuai dengan yang diharapkan atau tidak. Kemudian, hasil pengujian selanjutnya dianalisis apakah sesuai dengan tujuan penelitian

3.4 Kesimpulan

Pengambilan kesimpulan dilakukan ketika perancangan, implementasi, pengujian, dan analisis sudah selesai dilakukan. Kesimpulan didapat dari hasil pengujian dan analisis yang telah dibangun. Setelah menarik kesimpulan kemudian dapat diberikan saran yang dimaksudkan untuk memberi masukan dan pertimbangan terhadap pengembangan kedepannya

BAB 4 PERANCANGAN DAN IMPLEMENTASI

Pada bab ini akan dijelaskan mengenai perancangan dan implementasi dari Robot dengan menerapkan metode *Inverse Kinematic* dan *Iterative*. Perancangan akan dibagi menjadi perancangan Robot dan perancangan Kinematika, kemudian hasil dari perancangan akan dibahas dalam implementasi.

4.1 Analisis Kebutuhan

Pada subbab ini akan menjelaskan tentang apa saja yang dibutuhkan oleh penulis untuk melakukan penelitian ini. Analisis kebutuhan terbagi menjadi dua yaitu kebutuhan perangkat keras dan kebutuhan perangkat lunak.

4.1.1 Perangkat Keras

Kebutuhan perangkat keras yang digunakan dalam pengerjaan tugas akhir ini membutuhkan perangkat keras yang mampu men-*support* berbagai aplikasi dan program yang akan digunakan dalam pengerjaan tugas akhir ini. Berikut ini adalah beberapa spesifikasi perangkat keras yang digunakan dalam pengerjaan tugas akhir :

64-Bit MATLAB, Simulink, and Polyspace Product Families				
Operating Systems	**Processors**	**Disk**	**RAM**	**Graphics**
Windows 10 Windows 8.1 Windows 7 Service Pack 1 Windows Server 2016 Windows Server 2012 R2 Windows Server 2012	**Minimum** Any Intel or AMD x86-64 processor **Recommended** Any Intel or AMD x86-64 processor with four logical cores and AVX2 instruction set support	**Minimum** 2 GB of HDD space for MATLAB only, 4-6 GB for a typical installation **Recommended** An SSD is recommended A full installation of all MathWorks products may take up to 22 GB of disk space	**Minimum** 4 GB **Recommended** 8 GB For Polyspace, 4 GB per core is recommended	No specific graphics card is required. Hardware accelerated graphics card supporting OpenGL 3.3 with 1GB GPU memory is recommended. GPU acceleration using the Parallel Computing Toolbox requires a CUDA GPU. See GPU Computing Support for details.

Gambar 4.1 Kebutuhan perangkat keras

Sumber : https://www.mathworks.com/support/sysreq.html

4.1.2 Perangkat Lunak

Kebutuhan perangkat lunak yang digunakan dalam pengerjaan tugas akhir ini adalah menggunakan aplikasi *simulator* dan komputasi yaitu MATLAB versi

R2016a. MATLAB digunakan untuk memodelkan scenario dan melakukan komputasi pada pengujian dengan menggunakan scenario yang telah dirancang oleh peneliti.

4.1.3 Kebutuhan Fungsional

Kebutuhan fungsional merupakan penjelasan dari fitur atau fungsi yang harus ada dan dapat diperoleh dengan sistem tersebut. Pada kebutuhan sistem meliputi aspek *input* dan *output* sistem, serta fungsi respon sistem terhadap *input* dan *output* dalam proses berjalannya sistem. Kebutuhan fungsional dari sistem adalah sebagai berikut :

1. Menghitung dan memproses persamaan dari *forward kinematics*

   Sistem dapat menghitung dan memproses persamaan dari *forward kinematics.* Untuk dapat memenuhi fungsi ini, sistem harus dapat memproses masukkan berupa variabel *joint* dan memprosesnya menjadi keluaran koordinat menggunakan persamaan seperti yang sudah dijelaskan sebelumnya. Fungsi ini akan digunakan pada fungsi *workspace*, inverse kinematic, dan CCD.

2. Menghitung dan memetakan *workspace* dari robot

   Sistem dapat menghitung dan memetakan area kerja. Untuk dapat memenuhi fungsi ini, sistem harus dapat memproses fungsi *forward kinematics*, kemudian menyimpan variabel *joint* yang masih dalam area limitasi variabel *joint*-nya untuk dimasukkan ke dalam fungsi *forward kinematic*. Sehingga akan muncul keluaran berupa setiap kemungkinan koordinat yang dapat dicapai oleh robot SCARA. Hasil dari fungsi ini akan digunakan untuk memilih data untuk pengujian nantinya.

3. Menghitung besar error menggunakan *Euclidean Distance.*

   Sistem dapat menghitung besar *error* menggunakan *Euclidean Distance*. Untuk memenuhi fungsi ini, sistem harus dapat menerima masukkan berupa 2 titik (x, y, z), kemudian memprosesnya menjadi keluaran berupa besar jarak dari 2 titik tersebut dengan cara memasukkannya ke dalam persamaan *Euclidean Distance*. Hasil dari fungsi ini akan menjadi pembanding dari 2 metode yang akan diuji.

4. Menghitung waktu pemrosesan.

   Sistem dapat menghitung waktu pemrosesan dari 2 metode yang akan diuji. Untuk dapat memenuhi fungsi ini, dibutuhkan fungsi tic; dan toc; dari MATLAB. Fungsi ini akan menjadi pembanding dari 2 metode yang akan diuji.

5. Menghitung dan memproses persamaan dari *inverse kinematics*

   Sistem dapat menghitung dan memproses persamaan dari *inverse kinematics*. Untuk dapat memenuhi fungsi ini, sistem harus dapat memproses masukkan berupa koordinat akhir yang termasuk dalam

workspace dan memprosesnya menjadi keluaran variabel *joint* dari robot menggunakan persamaan seperti yang sudah dijelaskan sebelumnya. Dan kemudian akan diuji keakuratan dan besar waktu pemrosesannya.

6. Menghitung dan memproses persamaan dari *Cyclic Coordinate Descent*.

   Sistem dapat menghitung dan memproses persamaan dari CCD. Untuk dapat memenuhi fungsi ini, sistem harus dapat memproses masukkan berupa koordinat akhir yang termasuk dalam *workspace* dan memprosesnya menjadi keluaran variabel *joint* dari robot menggunakan persamaan seperti yang sudah dijelaskan sebelumnya. Dan kemudian akan diuji keakuratan dan besar waktu pemrosesannya.

7. Mem-*plotting* robot SCARA.

   Sistem dapat mem-*plotting* robot yang sudah dirancang sebelumnya dan bergerak menuju koordinat berdasarkan variabel *joint* dari 2 metode sebelumnya. Untuk dapat memenuhi fungsi ini, sistem membutuhkan "Robotic Toolbox", yaitu *toolbox* buatan Peter Corke untuk memproses sudut dan parameter DH yang sudah dirancang.

4.1.4 Kebutuhan non fungsional

Kebutuhan non fungsional merupakan kebutuhan yang digunakan untuk mendukung fungsi dasar dari sistem yang akan dibuat agar dapat berjalan dengan optimal. Kebutuhan non fungsionalitas dalam penelitian yang dilakukan adalah :

1. MATLAB IDE

   MATLAB IDE sebagai tempat membuat, meng-*compile*, dan mengeksekusi *script* yang akan dibuat

2. Bahasa Pemrograman MATLAB

   Pemograman dengan MATLAB IDE dilakukan dalam bahasa MATLAB

3. *Robotic Toolbox for* MATLAB

   Robotic Toolbox for MATLAB sebagai eksekutor dalam memenuhi fungsi *plotting* robot.

4.2 Perancangan Simulasi

Pada subbab ini akan menjelaskan lebih detail mengenai perancangan model robot dengan metode *Inverse Kinematic dan Iterative*. Perancangan model robot akan dibagi menjadi dua yaitu perancangan robot dan perancangan kinematika.

4.2.1 Perancangan Robot

Penelitian ini menggunakan Robot SCARA 3 DOF AdeptThree XL dengan perubahan ukuran dan diagram kinematika, berikut perbedaannya :

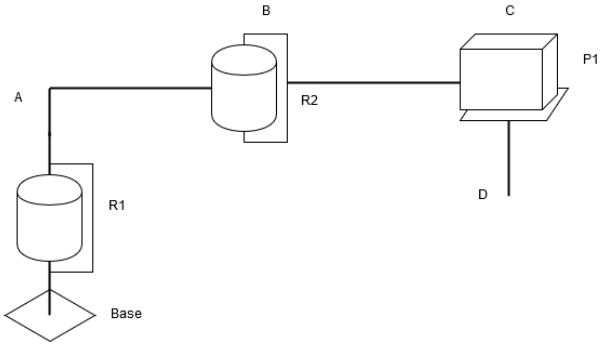
1. Adept Three XL mempunyai konfigurasi 4 DOF. Dikarenakan penelitian ini tidak memakai *end effector* berupa *grip* ataupun yang lainnya.
2. Robot untuk penelitian ini disesuaikan panjang lengannya untuk menjaga area kerja tetap kecil karena hanya untuk simulasi. Berikut perbedaannya pada tabel 4.1 :

Tabel 4.1 Penyesuaian AdeptThree XL terhadap Robot Demo

Robot / Item	AdeptThree XL	Robot SCARA Demo
Base – A (d1)	115,2 cm	8 cm
A – B (L1)	55,9 cm	6 cm
B – C (L2)	50,8 cm	6 cm
C – D (d3)	35,5 cm	3 cm
Batas Sudut R1	$-150^o < \theta < 150^o$	$-30^o < \theta < 30^o$
Batas Sudut R2	$-150^o < \theta < 150^o$	$-30^o < \theta < 30^o$
Batas Translasi P1	$0 < d < 30,5$ cm	$0 < d < 3$ cm

a. Perancangan Robot SCARA

Konfigurasi Robot *Manipulator* SCARA mempunyai 3 Degree Of Freedom (DOF) terdiri dari 2 *revolute joint* yang membentuk *serial chain* dan *prismatic joint.* Diagram kinematika untuk Robot 2 DOF ini dapat dilihat di Gambar 4.1



Gambar 4.2 Diagram Kinematika Robot SCARA

Link antara Base dan A yang disebut d1 mempunyai panjang 8 cm, A dan B yang disebut L1 mempunyai panjang 6 cm, B dan C yang disebut L2 mempunyai panjang 6 cm, dan C dan D yang disebut d3 mempunyai panjang 3 cm.

Robot SCARA mempunyai paramer DH sebagai berikut pada Tabel 4.2 :

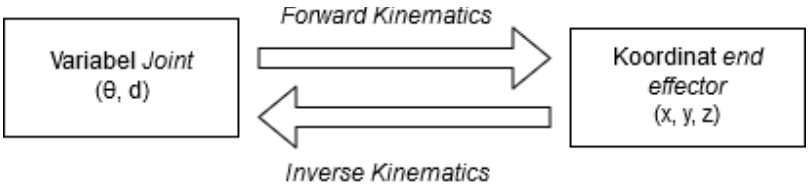
Tabel 4.2 Tabel Parameter DH Robot SCARA

Link	θ (Theta)	d (*Link Offset)*	a (Panjang Lengan)	α (Sudut *Twist*)
1	θ_1	d1	L1	0
2	θ_2	0	L2	π
3	0	d3	0	0

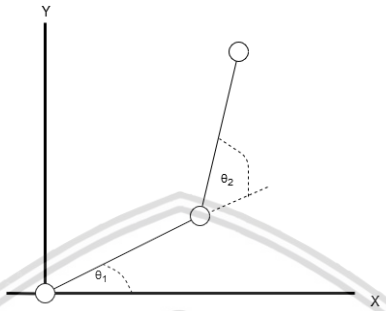
4.2.2 Perancangan Kinematika

A. Perancangan *Forward Kinematics*

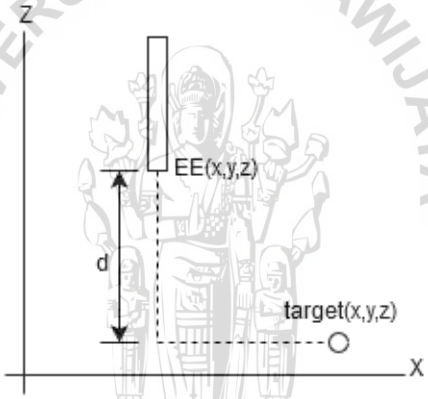
Sebelum masuk ke perhitungan *Inverse*, akan dijelaskan proses penghitungan *forward kinematics*. *Forward kinematics* merupakan proses perubahan sudut menjadi koordinat sehingga hasil koordinat akhir yang dihasilkan sistem dapat dibandingkan dengan titik koordinat yang dimasukkan oleh pengguna. Pada *forward kinematics* diketahui sudut perubahan pada masing-masing *joint* dan akan dicari titik koordinat yang berhasil dicapai berdasarkan sudut pada masing-masing *joint* tersebut. Gambar 4.2 di bawah ini merupakan hubungan antara *Inverse* dan *Forward Kinematics*.



Gambar 4.3 Diagram *Forward* dan *Inverse Kinematics*



Gambar 4.4 Posisi θ_1 dan θ_2



Gambar 4.5 Posisi d

Sesuai dengan perancangan robot, robot ini terdiri dari 2 *revolute joint* dan 1 *prismatic joint*. Pada Gambar 4.3 merupakan variabel *joint* dari 2 *revolute joint* dan Gambar 4.4 merupakan variabel *joint* dari 1 *prismatic joint*. *Revolute joint* pada SCARA akan berpengaruh pada koordinat x dan y sedangkan *prismatic joint* akan berpengaruh pada koordinat z. Maka :

$$X = L1 * \cos(\theta 1) + L2 * \cos(\theta 1 + \theta 2) \quad (4.1)$$

$$Y = L1 * \sin(\theta 1) + L2 * \sin(\theta 1 + \theta 2) \quad (4.2)$$

$$Z = d1 - (d3 + d) \quad (4.3)$$

Keterangan :

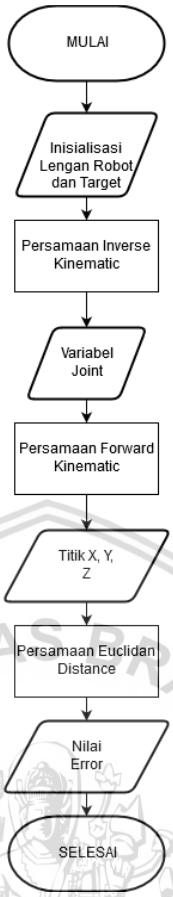
- L1 : panjang dari lengan/*link* 1
- L2 : panjang dari lengan/*link* 2

- θ_2 : perubahan sudut/variabel *joint* dari *revolute joint* 2
- θ_1 : perubahan sudut/variabel *joint* dari *revolute joint* 1
- d1 : panjang dari *link offset* 1
- d3 : panjang dari *link offset* 3
- d : variabel *joint* dari *prismatic joint*
- X : titik koordinat X dari *end-effector*
- Y : titik koordinat Y dari *end-effector*
- Z : titik koordinat Z dari *end-effector*

Persamaan 4.1, 4.2 dan 4.3 merupakan persamaan untuk mencari titik X, Y, dan Z berdasarkan masukkan variabel masing-masing *joint.*

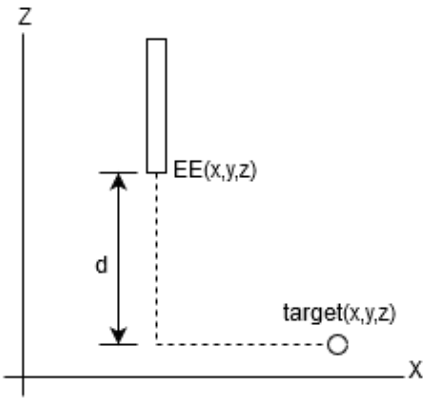
B. Perancangan *Inverse Kinematics*

Pada subbab ini akan dijelaskan tentang proses penghitungan *inverse kinematics* guna mengubah input koordinat dari *user* menjadi besar sudut yang dibutuhkan masing-masing *joint* agar ujung lengan mampu mencapai titik koordinat yang diinginkan. Perancangan *inverse kinematics* selanjutnya akan terbagi ke dalam 2 bagian, *prismatic joint* dan *serial chain*. Untuk memudahkan perhitungan matematis, maka semua perhitungan akan menggunakan 2 dimensi dimana *prismatic joint* menggunakan tampak samping (sumbu X – Z) dan *serial chain* menggunakan tampak atas (sumbu X – Y).



Gambar 4.6 *Flowchart* skrip *Inverse Kinematic*

a. Prismatic Joint



Gambar 4.7 Prismatic Joint

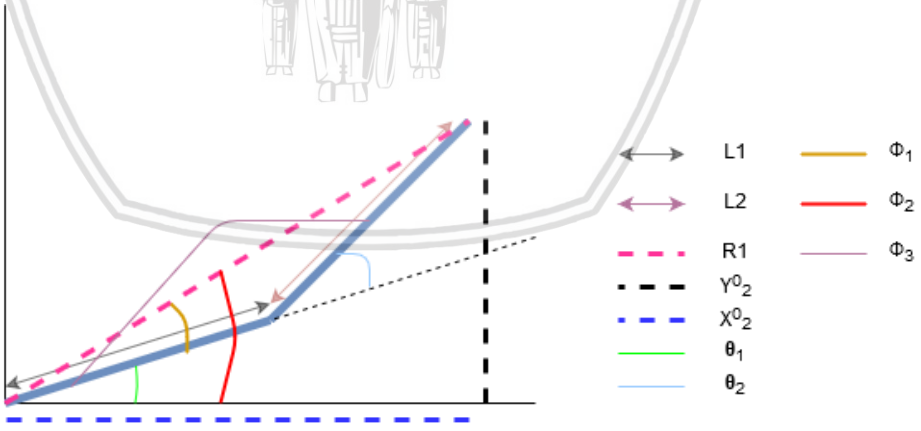
Pada Gambar 4.5, yang harus dicari adalah nilai d, dimana d merupakan panjang yang harus dicapai agar sejajar dengan titik z dari koordinat z target yaitu z'. Untuk mendapatkan nilai d, maka persamaan 4.4 dapat digunakan.

$$d = (d1 - d3) - z \tag{4.4}$$

Keterangan :

- d1 : Panjang lengan d1.
- d3 : Panjang lengan d3.
- Z : Koordinat Z dari *end-effector* (x,y,z).
- d : variable *joint* dari *prismatic joint*

b. Serial Chain



Gambar 4.8 *Serial Chain revolute*

Berdasarkan gambar 4.6, L1 merupakan panjang lengan (*link*) 1, kemudian diikuti oleh L2 yang merupakan panjang lengan 2. Sisanya merupakan garis imajiner yang terdiri dari R1 yang merupakan garis penghubung *end effector* dengan *base*, kemudian ada Y^0_2 dan X^0_2 yang merupakan sumbu Y dan X dari *base* terhadap *end effector*, kemudian ada theta1 dan theta2 (θ_1 dan θ_2) yang

merupakan sudut masing masing *joint*. Untuk phi1, phi2, dan phi3 (Φ_1, Φ_2, dan Φ_3) merupakan sudut bantu yang digunakan untuk mencari theta1 dan theta2.

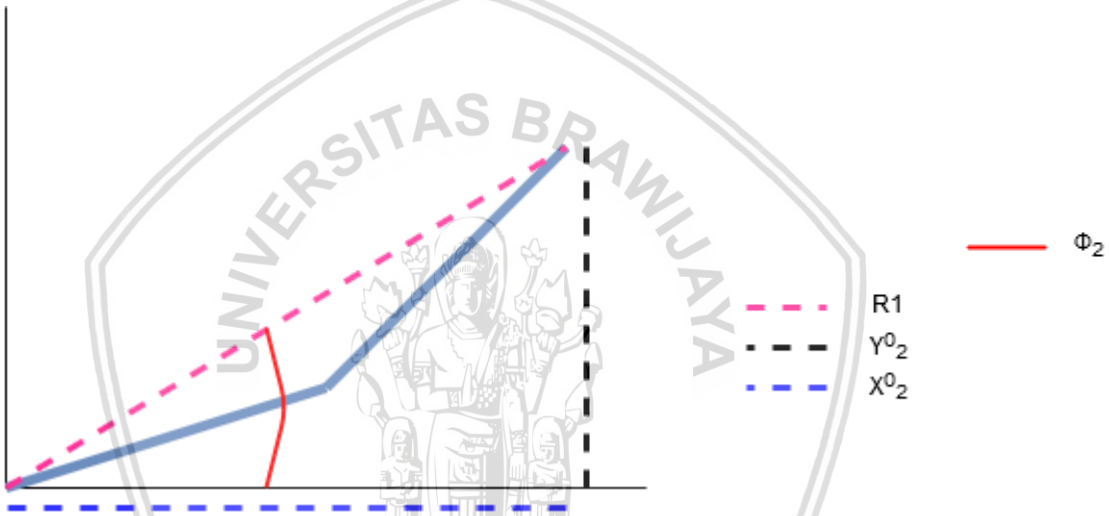
Berdasarkan gambar di atas, maka :

$$\theta_1 = \Phi_2 - \Phi_1 \quad (4.5)$$

Keterangan :

- θ_1 : perubahan sudut/variabel *joint* dari *revolute joint* 1
- Φ_1 : sudut antara L1 dan R1
- Φ_2 : sudut antara R1 dan X_2^0

Untuk mencari θ_1, seperti pada persamaan 4.5 maka harus mencari Φ_2 dan Φ_1 . Untuk mencari Φ_2, dapat digunakan segitiga R1 - Y^0_2 - X^0_2 :



Gambar 4.9 segitiga R1 - Y^0_2 - X^0_2

Berdasarkan gambar 4.7, Φ_2 dapat dicari dengan fungsi tangen. Persamaannya adalah :

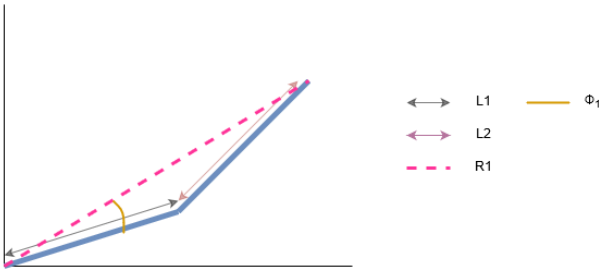
$$\Phi_2 = \tan^{-1} \frac{Y_2^0}{X_2^0} \quad (4.7)$$

Keterangan :

- $\mathbf{X_2^0}$: garis imajiner sumbu X dari koordinat lokal robot
- $\mathbf{Y_2^0}$: garis imajiner sumbu Y dari koordinat lokal robot
- Φ_2 : sudut antara R1 dan X_2^0

Φ_2 dapat dicari dengan menggunakan rumus trigonometri dari tangen, dengan melihat Φ_2 dalam segitiga R1 - Y_2^0 - X_2^0. Maka persamaannya adalah seperti persamaan 4.7.

Setelah mendapatkan Φ_2, kemudian mencari Φ_1. Untuk mencari Φ_1 dapat menggunakan aturan cosinus pada segitiga L1 - L2 - R1.



Gambar 4.10 Segitiga L1 - L2 - R1

Berdasarkan gambar 4.8, persamaan aturan cosinus dari segitiga L1-L2-R1 adalah seperti persamaan 4.8 di bawah ini :

$$(L2)^2 = (L1)^2 + (R1)^2 - 2\ L1\ R1\ \cos(\Phi_1) \quad (4.8)$$

$$R1 = \sqrt{(X_2^0)^2 + (Y_2^0)^2} \quad (4.9)$$

Keterangan :

- X_2^0 : garis imajiner sumbu X dari koordinat lokal robot
- Y_2^0 : garis imajiner sumbu Y dari koordinat lokal robot
- Φ_1 : sudut antara L1 dan R1
- R1 : garis imajiner yang menghubungkan L1 dan L2
- L1 : panjang dari lengan/*link* 1
- L2 : panjang dari lengan/*link* 2

Nilai R1 pada persamaan 4.8 dapat menggunakan rumus Pythagoras pada segitiga R1 – L1 – L2, sehingga persamaannya menjadi seperti pada persamaan 4.9. Karena yang dicari adalah Φ_1, maka persamaan 4.8 dapat diturunkan menjadi persamaan 4.10 di bawah ini :

$$\Phi_1 = \cos^{-1}\left(\frac{L2^2 - L1^2 - R1^2}{-2\ L1\ R1}\right) \quad (4.10)$$

Keterangan :

- Φ_1 : sudut antara L1 dan R1
- R1 : garis imajiner yang menghubungkan L1 dan L2
- L1 : panjang dari lengan/*link* 1
- L2 : panjang dari lengan/*link* 2

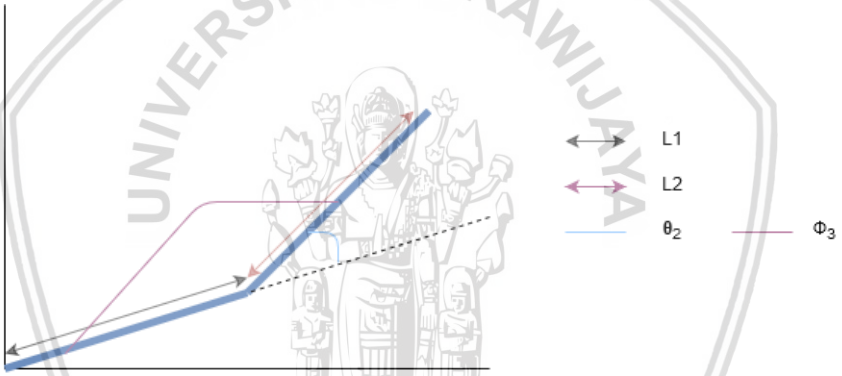
Berdasarkan persamaan 4.7 dan 4.10, maka dapat diketahui perubahan sudut θ_1 dengan digabung menjadi persamaan 4.11 :

$$\theta_1 = (\tan^{-1}(\frac{Y02}{X02})) - (\cos^{-1}(\frac{L2^2 - L1^2 - R1^2}{-2\ L1\ R1})) \quad (4.11)$$

Keterangan :

- $\mathbf{X_2^0}$: garis imajiner sumbu X dari koordinat lokal robot
- $\mathbf{Y_2^0}$: garis imajiner sumbu Y dari koordinat lokal robot
- Φ_1 : sudut antara L1 dan R1
- R1 : garis imajiner yang menghubungkan L1 dan L2
- L1 : panjang dari lengan/*link* 1
- L2 : panjang dari lengan/*link* 2

Untuk θ_2, digunakan bantuan L1 yang dipanjangkan dan sudut Φ_3 seperti gambar berikut :



Gambar 4.11 Sudut $\theta_2 + \Phi_3$

Berdasarkan gambar 4.9, Φ_3 dan θ_2 membentuk sudut 180°, sehingga dapat dibuat persamaan seperti persamaan 4.12 dan 4.13 berikut :

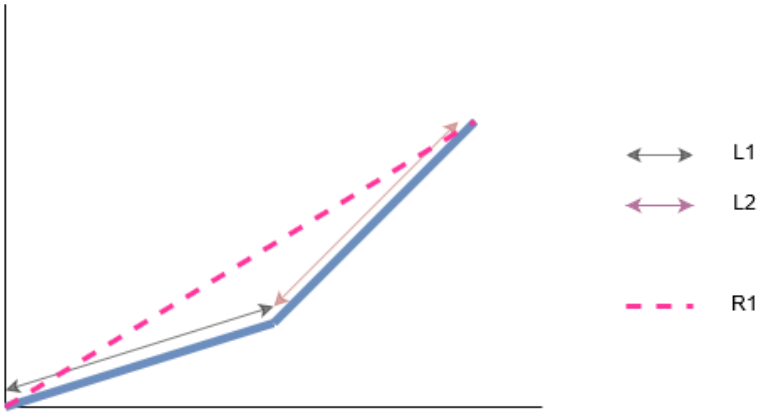
$$180° = \theta_2 + \Phi_3 \quad (4.12)$$

$$\theta_2 = 180° - \Phi_3 \quad (4.13)$$

Keterangan :

- θ_2 : perubahan sudut/variabel *joint* dari *revolute joint* 2
- Φ_3 : sudut antara L1 dan L2

Untuk mencari Φ_3, dapat menggunakan aturan cosinus pada segitiga L1 - L2 - R1 :



Gambar 4.12 Segitiga L1 - L2 - R1

Berdasarkan gambar 4.10, Φ_3 dan θ_2 persamaan aturan cosinus dari segitiga L1-L2-R1 adalah seperti persamaan 4.14 di bawah ini :

$$(R1)^2 = (L1)^2 + (L2)^2 - 2\ L1\ L2\ \cos(\Phi_3) \quad (4.14)$$

Karena yang dicari adalah Φ_3, maka persamaan 4.8 dapat diturunkan menjadi persamaan 4.15 di bawah ini :

$$\Phi_3 = \cos\text{-}1\ \left(\frac{R1^2 - L1^2 - L2^2}{-2\ L1\ L2}\right) \quad (4.15)$$

Setelah didapat persamaan Φ_3, maka perubahan sudut pada θ_2 menjadi pada persamaan 4.16 :

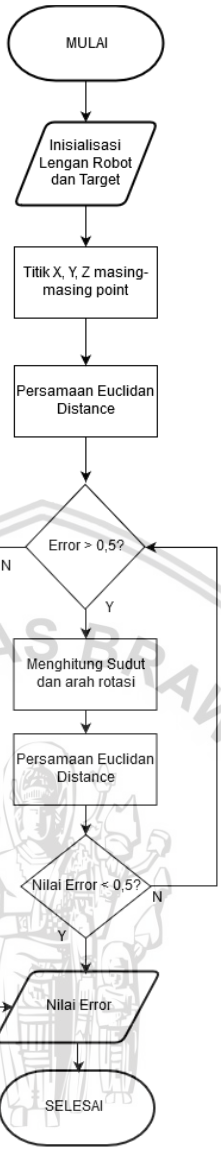
$$\theta_2 = 180° - \cos^{-1}\ \left(\frac{R1^2 - L1^2 - L2^2}{-2\ L1\ L2}\right) \quad (4.16)$$

Keterangan :

- Φ_3 : sudut antara L1 dan L2
- R1 : garis imajiner yang menghubungkan L1 dan L2
- L1 : panjang dari lengan/*link* 1
- L2 : panjang dari lengan/*link* 2

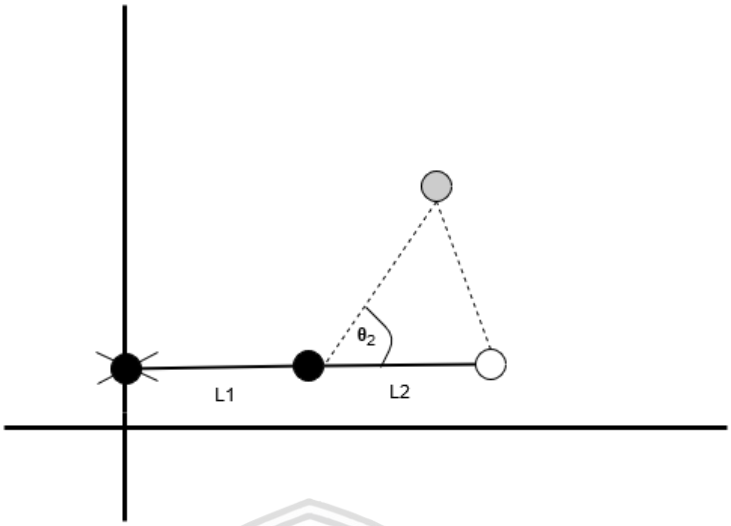
C. Perancangan *Iterative CCD (Cyclic Coordinate Descent)*

Dalam tesisnya, Chris Welman mendeskripsikan metode untuk mengoptimasi *Inverse Kinematics*. Ia menggunakan teknik yang awalnya diperkenalkan oleh Li-Chun Tommy Wang dan Chih Cheng Chen dalam karya tulisnya di *IEEE Transactions on Robotics*. CCD melibatkan minimalisasi kesalahan yang ada pada sistem dengan menyesuaikan sudut pada setiap *joint* satu per satu. CCD dimulai dari *link* pada *end effector*, kemudian *link* selanjutnya hingga *base*.



Gambar 4.13 *Flowchart* skrip *Cyclic Coordinate Descent*

Sama seperti perancangan *inverse*, perancangan CCD selanjutnya akan dibagi 2 berdasarkan jenis *joint*-nya, yaitu *Prismatic Joint* dan *Revolute Joint*. Dua bagian ini akan dirancang dalam bentuk *planar* untuk mempermudah perhitungan.



Gambar 4.14 Visual CCD pada *Revolute Joint*

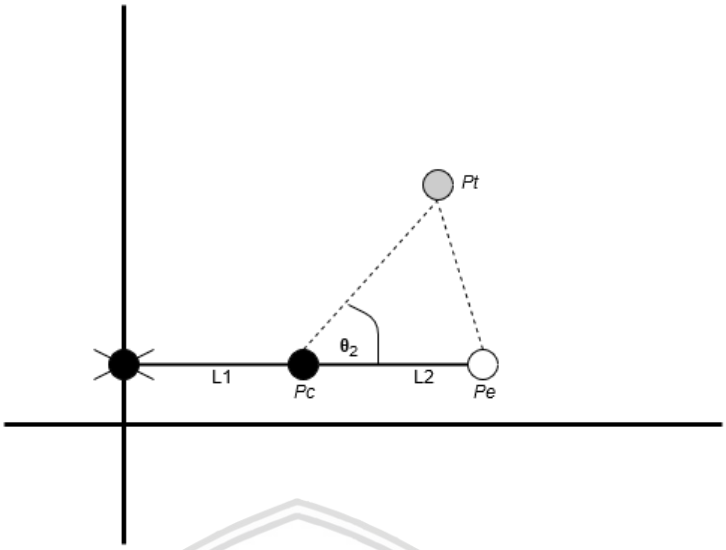
Algoritma CCD menggunakan pendekatan heuristik untuk mencari solusi dengan memutar *link* berulang-ulang kali, sehingga *end effector* berpindah mendekati target. Pada bagian *revolute joint,* setiap iterasi akan memutar *link i-th*, dimulai dari *end effector* hingga *base*. Iterasi dilakukan untuk meminimalisasi sudut θ_i diantara vektor dari *link* menuju *end effector* dan vektor dari *link* menuju target, seperti gambar 4.11.

Pada bagian *prismatic joint* tidak berbeda dengan perhitungan dengan *inverse kinematics*. Dimana yang harus dicari adalah nilai d, dimana d merupakan panjang yang harus dicapai agar sejajar dengan titik Z dari koordinat target. Maka persamaan d-nya seperti persamaan 4.17 di bawah ini :

$$d = (d1 - d3) - z \tag{4.17}$$

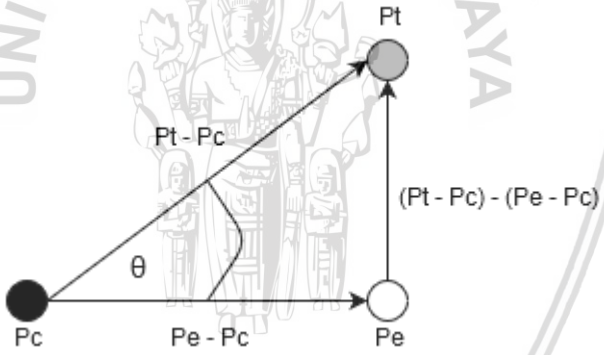
Keterangan :

- d1 : Panjang lengan d1.
- d3 : Panjang lengan d3.
- Z : Koordinat Z dari *end-effector* (x,y,z).
- d : variable *joint* dari *prismatic joint*



Gambar 4.15 Posisi Pc, Pt, Pe

Berdasarkan gambar 4.12, L1 merupakan panjang lengan (*link*) 1, dan diikuti oleh L2 yang merupakan panjang lengan 2. Kemudian, ada Pe yang merupakan koordinat posisi dari *end effector*, Pt merupakan koordinat posisi dari target, dan Pc merupakan koordinat joint dari *link i-th*.



Gambar 4.16 Cosinus Vektor pada CCD

Untuk mencari θ, maka diperlukan hukum cosinus pada vector. Dengan melihat gambar 4.13, maka persamaannya :

$$\left(\overrightarrow{(Pt - Pc)}.\overrightarrow{(Pe - Pc)}\right) = \left\|\overrightarrow{(Pt - Pc)}\right\| \left\|\overrightarrow{(Pe - Pc)}\right\| \cos\theta \qquad (4.18)$$

Keterangan :

- Pe : titik koordinat dari *end-effector*
- Pc : titik koordinat dari *joint*
- Pt : titik koordinat dari target
- $\overrightarrow{(\boldsymbol{Pe - Pc})}$: vektor dari Pc ke Pe
- $\overrightarrow{(\boldsymbol{Pt - Pc})}$: vektor dari Pc ke Pt
- θ : sudut antara $\overrightarrow{(Pt - Pc)}$ dan $\overrightarrow{(Pe - Pc)}$

Pada persamaan 4.18, vektor Pt – Pc dan Pe – Pc dilakukan perkalian titik (*dot product)* dan hasilnya sama dengan vektor Pt – Pc yang di-*norm*-kan yang kemudian dikali dengan vektor Pe – Pc yang di-*norm*-kan juga dan kemudian dikalikan dengan cosinus dari θ. Untuk mencari θ, persamaannya adalah :

$$\cos\theta = \frac{(\overrightarrow{(Pt-Pc)}.\overrightarrow{(Pe-Pc)})}{\left\|\overrightarrow{(Pt-Pc)}\right\| \left\|\overrightarrow{(Pe-Pc)}\right\|} \tag{4.18}$$

$$\theta = \text{acos}^{-1}\left(\frac{\overrightarrow{(Pt-Pc)}}{\left\|\overrightarrow{(Pt-Pc)}\right\|} \cdot \frac{\overrightarrow{(Pe-Pc)}}{\left\|\overrightarrow{(Pe-Pc)}\right\|}\right) \tag{4.20}$$

Keterangan :

- Pe : titik koordinat dari *end-effector*
- Pc : titik koordinat dari *joint*
- Pt : titik koordinat dari target
- $\overrightarrow{(\boldsymbol{Pe}-\boldsymbol{Pc})}$: vektor dari Pc ke Pe
- $\overrightarrow{(\boldsymbol{Pt}-\boldsymbol{Pc})}$: vektor dari Pc ke Pt
- θ : sudut antara $\overrightarrow{(Pt-Pc)}$ dan $\overrightarrow{(Pe-Pc)}$

Pada persamaan 4.19, untuk mencari θ maka cos θ dipindahkan ke sebelah kiri persamaan. Kemudian, vektor Pt – Pc dan Pe – Pc dilakukan perkalian titik (*dot product)* dibagi dengan perkalian dari *norm* masing-masing vektor Pt – Pc dan Pe – Pc, yang kemudian menjadi persamaan 4.20.

Setelah mendapatkan sudut, maka perlu dicari arah putarannya. Untuk mencari arah putaran dapat menggunakan *cross product* dari 2 vektor $\overrightarrow{(Pt-Pc)}$ dan $\overrightarrow{(Pe-Pc)}$.

$$\overrightarrow{(Pe-Pc)} \times \overrightarrow{(Pt-Pc)} = \left\|\overrightarrow{(Pe-Pc)}\right\| \left\|\overrightarrow{(Pt-Pc)}\right\| \sin\theta \tag{4.21}$$

Keterangan :

- Pe : titik koordinat dari *end-effector*
- Pc : titik koordinat dari *joint*
- Pt : titik koordinat dari target
- $\overrightarrow{(\boldsymbol{Pe}-\boldsymbol{Pc})}$: vektor dari Pc ke Pe
- $\overrightarrow{(\boldsymbol{Pt}-\boldsymbol{Pc})}$: vektor dari Pc ke Pt
- θ : sudut antara $\overrightarrow{(Pt-Pc)}$ dan $\overrightarrow{(Pe-Pc)}$

Pada persamaan 4.21, vektor Pt – Pc dan Pe – Pc dilakukan perkalian silang (*cross product*) dan hasilnya sama dengan vektor Pt – Pc yang di-*norm*-kan yang kemudian dikali dengan vektor Pe – Pc yang di-*norm*-kan juga dan kemudian dikalikan dengan cosinus dari θ. Besar θ sudah diketahui di persamaan 4.20,

tetapi tidak diketahui apakah besarnya minus (berputar ke kiri) atau tidak. Jika θ bernilai minus.

Jika $\overrightarrow{(Pe - Pc)}$ = a dan $\overrightarrow{(Pt - Pc)}$ = b, maka hasil dari *cross product* dalam 3 dimensi adalah :

$$\begin{bmatrix} a_x \\ a_y \\ a_z \end{bmatrix} \times \begin{bmatrix} b_x \\ b_y \\ b_z \end{bmatrix} = \begin{bmatrix} a_y b_z - a_z b_y \\ a_z b_x - b_z a_x \\ a_x b_y - a_y b_x \end{bmatrix} \quad (4.22)$$

Keterangan :

- a_x : komponen x dari vektor a
- a_y : komponen y dari vektor a
- a_z : komponen z dari vektor a
- $\boldsymbol{b_x}$: komponen x dari vektor b
- $\boldsymbol{b_y}$: komponen y dari vektor b
- b_z : komponen z dari vektor b

Persamaan 4.22 merupakan cara pengerjaan dari perkalian vektor a dan b. Dan karena CCD berjalan dalam sistem *planar*, maka *cross product* untuk 2 dimensi :

$$\begin{bmatrix} a_x \\ a_y \end{bmatrix} \times \begin{bmatrix} b_x \\ b_y \end{bmatrix} = \begin{bmatrix} \\ a_x b_y - a_y b_x \end{bmatrix} \quad (4.23)$$

Keterangan :

- a_x : komponen x dari vektor a
- a_y : komponen y dari vektor a
- $\boldsymbol{b_x}$: komponen x dari vektor b
- $\boldsymbol{b_y}$: komponen y dari vektor b

Jika hasil dari *cross product* pada persamaan 4.23 bernilai positif, maka θ akan bernilai positif, dan jika *cross product* bernilai negatif, maka θ akan bernilai negatif.

Setelah mendapatkan besar sudut dan arah putarannya, maka akan dilakukan rotasi menggunakan matriks rotasinya. Persamaan 4.24 merupakan matriks rotasi dalam sistem koordinat 2 dimensi.

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} \quad (4.24)$$

Keterangan :

- x' : titik x hasil rotasi

- y' : titik y hasil rotasi
- $\boldsymbol{x}$: titik x sebelum rotasi
- $\boldsymbol{y}$: titik y sebelum rotasi
- θ : besar sudut rotasi

x dan y pada matriks rotasi menggunakan koordinat lokal (koordinat relatif terhadap *link* sebelumnya) sebagai sistem acuannya. Karena saat rotasi akan menghasilkan koordinat baru, maka harus diubah menjadi koordinat global. Sehingga matriks rotasinya akan menjadi persamaan 4.25 :

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix} \left(\begin{bmatrix} x_{i+1} \\ y_{i+1} \end{bmatrix} - \begin{bmatrix} x_i \\ y_i \end{bmatrix} \right) + \begin{bmatrix} x_i \\ y_i \end{bmatrix} \quad (4.25)$$

Keterangan :

- x' : titik x hasil rotasi
- y' : titik y hasil rotasi
- $\boldsymbol{x_i}$: titik x sebelum rotasi dari *link*/lengan $\boldsymbol{i}$
- $\boldsymbol{y_i}$: titik y sebelum rotasi dari *link*/lengan $\boldsymbol{i}$
- x_{i+1} : titik x sebelum rotasi dari *link*/lengan $i+1$
- y_{i+1} : titik y sebelum rotasi dari *link*/lengan $i+1$
- θ : besar sudut rotasi

Selain koordinat baru, hasil persamaan 4.22 juga harus diubah menjadi koordinat global. Ini dikarenakan sudut pada θ_i akan mempengaruhi sudut pada link setelahnya (θ_{i+1}) seperti pada persamaan 4.26 di bawah ini :

$$\theta_{i+1} = \theta_{i+1} + \theta_i \quad (4.26)$$

Keterangan :

- θ_{i+1} : besar sudut dari *revolute joint* $i+1$
- θ_i : besar sudut dari *revolute joint* i

Selain itu, *link* setelahnya (i+2) akan berpengaruh juga karena perpindahan *link* hasil rotasi seperti pada persamaan 4.27 dan 4.28 di bawah ini :

$$x_{(i+2)} = x_{(i+1)} + link_{(i+1)} \quad (4.27)$$

$$y_{(i+2)} = y_{(i+1)} \quad (4.28)$$

Keterangan :

- $x_{(i+2)}$: titik x dari *link*/lengan $i+2$
- $y_{(i+2)}$: titik y dari *link*/lengan $i+2$
- $x_{(i+1)}$: titik x dari *link*/lengan $i+1$

- $y_{(i+1)}$: titik y dari *link*/lengan $i + 1$
- *link*$_{(i+1)}$: panjang lengan dari lengan/*link* $i + 1$

Untuk X dan Y pada *end effector* tidak perlu penambahan sudut dan koordinat setelahnya karena tidak berpengaruh.

Dalam CCD, diperlukan juga batas iterasi dan batas toleransi *error* agar tidak melakukan iterasi terus menerus. Batas iterasi disini ditetapkan hingga 10000 agar tidak terlalu lama, sedangkan batas toleransi *error* dihitung menggunakan *Euclidean Distance.* Batas toleransi ditetapkan sebesar 0.5.

D. Perancangan *Euclidean DIstance*

Euclidean Distance merupakan perhitungan untuk mencari garis lurus antara 2 titik dalam ruang 3 dimensi. Garis lurus ini akan menjadi jarak antara 2 titik tersebut, dimana semakin kecil jaraknya, berarti semakin akurat titik koordinat hasil perhitungan 2 metode dan titik target. Sedangkan nilai maksimum *error* yang dapat ditoleransi sebesar 0.5 (Zhou). Persamaan *Euclidean Distance*-nya adalah seperti pada persamaan 4.29 :

$$\sqrt{(X_2 - X_1)^2 + (Y_2 - Y_1)^2 + (Z_2 - Z_1)^2} \quad (4.29)$$

Keterangan :

- $x2$: titik x dari koordinat 2
- $y2$: titik y dari koordinat 2
- $z2$: titik z dari koordinat 2
- $x1$: titik x dari koordinat 1
- $y1$: titik y dari koordinat 1
- $z1$: titik z dari koordinat 1

E. Area Kerja

Area Kerja sangat penting untuk dihitung untuk mengetahui batas koordinat yang bisa dicapai oleh lengan robot. Untuk menghitung area kerja, cukup memasukkan limitasi sudut per *joint* pada persamaan *forward kinematics* (4.1), (4.2), dan (4.3). Maka hasilnya seperti yang ada pada Tabel 4.3:

Tabel 4.3 Area Kerja Robot SCARA

θ_1	θ_2	d	X	Y	Z
0	0	3	12	0	2
0	30	3	11,1961	3	2
0	-30	3	11,1961	-3	2

30	0	3	10,3923	6	2
30	30	3	8,1961	8	2
30	-30	3	11,1961	3	2
-30	0	3	10,3923	-6	2
-30	30	3	11,1961	-3	2
-30	-30	3	8,1961	-8	2
0	0	0	12	0	5
0	30	0	11,1961	3	5
0	-30	0	11,1961	-3	5
30	0	0	10,3923	6	5
30	30	0	8,1961	8	5
30	-30	0	11,1961	3	5
-30	0	0	10,3923	-6	5
-30	30	0	11,1961	-3	5
-30	-30	0	8,1961	-8,1961	5

4.3 Implementasi

4.3.1 Implementasi Robot

Implementasi ini berfungsi untuk mensimulasikan robot berdasarkan perancangan robot. Robot akan bergerak sesuai dengan sudut yang dihasilkan oleh perhitungan kinematika.

1. Parameter DH dan limitasi sudut

Pertama adalah mendefinisikan parameter DH robot. Pada bagian *code* ini berguna untuk memprogram bagaimana bentuk robot pada MATLAB. Parameter yang dimasukkan adalah parameter yang ada di Tabel 4.2.

Pada *Link* pertama, theta-nya adalah 0, karena merupakan *variabel joint* dari *revolute joint*. Kemudian d bernilai 0, L bernilai 6, alpha bernilai 0, dan 0 terakhir adalah penanda *joint* tersebut adalah *revolute.*

Pada *Link* kedua, theta-nya adalah 0, karena merupakan *variabel joint* dari *revolute joint*. Kemudian d bernilai 0, L bernilai 6, alpha bernilai pi sebagai *twist*, dan 0 terakhir adalah penanda *joint* tersebut adalah *revolute.*

Pada *Link* ketiga, theta-nya adalah 0, karena *prismatic* tidak mempunyai *variabel joint* theta. Kemudian d bernilai 0 karena merupakan *variabel joint* dari *prismatic joint*, L bernilai 0, alpha bernilai 0, dan 1 terakhir adalah penanda *joint*

tersebut adalah *prismatic.* Kemudian, definisikan juga limitasi dari setiap *joint* dengan properti *qlim* sesuai dengan Tabel 4.1

Pada Tabel 4.4 dapat dilihat baris 1 sampai 3 merupakan proses untuk mendefinisikan parameter DH dari konfigurasi robot dan baris 5 sampai 7 merupakan proses untuk mendefinisikan limitasi dari setiap *joint* pada robot.

Tabel 4.4 Parameter DH dan limitasi sudut

AnimScara.m	
1	`L(1) = Link([0,0,6,0,0]);`
2	`L(2) = Link([0,0,6,pi,0]);`
3	`L(3) = Link([0,0,0,0,1]);`
4	
5	`L(1).qlim = [-pi/6 pi/6];`
6	`L(2).qlim = [-pi/6 pi/6];`
7	`L(3).qlim = [0 6];`

2. Inisiasi Workspace dan *plotting* robot

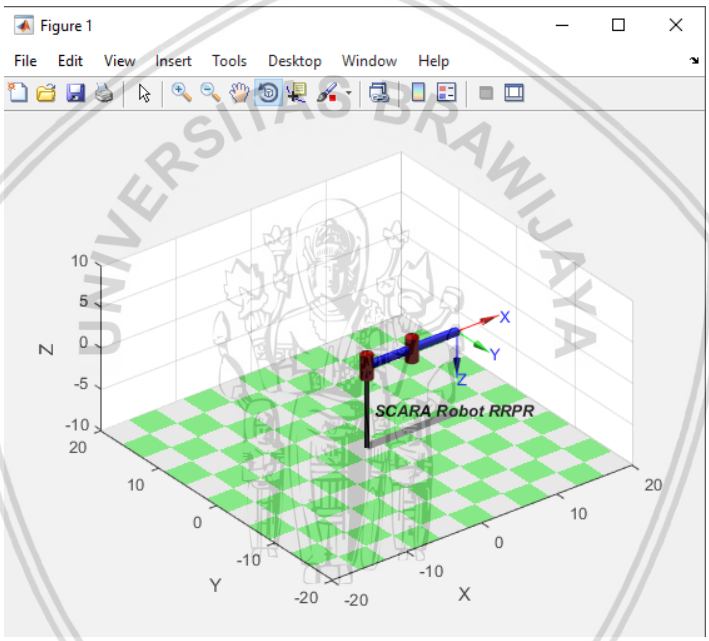
Workspace adalah ruang kerja untuk robot. *Workspace* menjadi syarat wajib jika ingin memakai *prismatic joint* dalam penggunaan *Robotic Toolbox*. *Workspace* dihitung dari total panjang lengan robot dan panjang *prismatic joint* bisa bertranslasi. Pada robot ini, total panjang adalah L1 + L2 = 12 dengan menambah jarak aman maka diambil 20cm, kemudian *prismatic* dapat bertranslasi sampai 3 cm dengan ditambah panjang lengannya maka hasilnya adalah 6 cm. Kemudian tinggi robot adalah 8 cm (d1). Maka *workspace*-nya adalah [-x x -y y -z z] = [-20 20 -20 20 -8 8].

Kemudian hasil perancangan robot dan *workspace* tadi dikemas menjadi 1 robot bernama 'SCARA Robot RRP' dengan nama variabel 'robot_scara'. robot_scara akan di-*plot* dengan sudut yang dimasukkan kedalam q(i), i adalah *sequence* jalannya sudut. Untuk q1 difungsikan sebagai *home position* yang berarti [0 0 0], sedangkan q2 difungsikan sebagai pergerakan selanjutnya. Setelah q selesai didefinisikan, maka akan dikemas ke dalam variabel 'tg' bersama jumlah langkah pergerakannya.

Pada Tabel 4.5 dapat dilihat baris 1 dan 2 merupakan deklarasi variabel 'ws' yang merupakan *workspace*. Kemudian baris 3 merupakan inisiasi variabel 'robot_scara' yang merupakan pemanggilan fungsi "*SerialLink*". Kemudian pada baris 5 sampai 7 mendefinisikan variabel sudut. Dan terakhir pada baris 8 merupakan proses *plotting* yang hasilnya seperti pada Gambar 4.14

Tabel 4.5 *Plotting* robot

AnimScara.m	
1	`ws = [-20 20 -20 20 -10 10];`
2	`plot_options = {'workspace',ws};`
3	`robot_scara = SerialLink(L,'name','SCARA Robot`
4	`RRP','plotopt',plot_options);`
5	`q1=[0 0 0];`
6	`q2=[t1_rad t2_rad d_fix];`
7	`tg=jtraj(q1, q2, 50);`
8	`robot_scara.plot(tg)`



Gambar 4.17 Robot SCARA pada MATLAB

4.3.2 Implementasi Kinematika

A. Forward Kinematics

1. Definisikan Panjang Lengan Robot

Pada Tabel 4.6, bagian *code* ini merupakan hasil dari perancangan model robot yaitu berupa panjang d1, L1, L2, d3.

Tabel 4.6 Implementasi Panjang Lengan Robot

Scara_3dof_FK.m	
1	d1 = 8;
2	l1 = 6;
3	l2 = 6;
4	d3 = 3;

2. Masukkan Variabel *Joint*

Perhitungan *forward kinematics* membutuhkan masukan berupa variabel *joint* yang dibutuhkan, yaitu θ_1, θ_2, dan d yang akan diisi pada bagian *code* seperti yang ada pada Tabel 4.7 ini. Variabel "t1" berisi nilai dari θ_1, "t2" berisi nilai θ_2, dan "d" berisi nilai dari d.

Tabel 4.7 Implementasi Panjang Lengan Robot

Scara_3dof_FK.m	
1	t1 = ;
2	t2 = ;
3	d = ;

3. Persamaan X, Y, dan Z

Pada Tabel 4.8 merupakan bagian *code* Persamaan untuk mencari X, Y, dan Z yang ada di Persamaan 4.1, 4.2, 4.3.

Tabel 4.8 Implementasi persamaan X, Y, dan Z

Scara_3dof_FK.m	
1	X = l1 * cosd(t1) + l2 * cosd(t1 + t2);
2	Y = l1 * sind(t1) + l2 * sind(t1 + t2);
3	Z = d1 - (d3 + d);

B. Implementasi Area Kerja

1. Definisikan Panjang Lengan Robot

Pada bagian *code* yang ada di Tabel 4.9, hasil dari perancangan robot akan didefinisikan, yaitu berupa panjang d1, L1, L2, d3.

Tabel 4.9 Implementasi Panjang Lengan Robot

Possible_coor_scara.m	
1	`d1 = 8;`
2	`l1 = 6;`
3	`l2 = 6;`
4	`d3 = 3;`

2. Implementasi *range* limitasi variabel *joint*

Pada bagian *code* yang ada di Tabel 4.10, limitasi *joint-joint* dibuat dalam deret, sehingga θ_1 = -30° hingga 30° dengan selisih 1, kemudian θ_2 = -30° hingga 30° dengan selisih 1, dan d = 0 hingga d3 dengan selisih 1. Semua variabel joint tersebut dimasukkan ke dalam *array-array*.

Tabel 4.10 Implementasi Panjang Lengan Robot

Possible_coor_scara.m	
1	`theta1 = -(rad2deg((pi/6))):1:(rad2deg((pi/6)));`
2	`theta2 = -(rad2deg((pi/6))):1:(rad2deg((pi/6)));`
3	`d = d3:-1:0;`

3. Implementasi *meshgrid*

Pada bagian code yang ada di Tabel 4.11, *array-array* yang berisi variabel-variabel *joint* tersebut kemudian digabung menjadi 3 *array* baru. Dimana isi *array-array* tersebut berukuran 61x61x4. Untuk memnuhi tugas itu, dibutuhkan fungsi *meshgrid* yang akan menggabungkan banyak variabel.

Tabel 4.11 Implementasi *Meshgrid*

Possible_coor_scara.m	
1	`[THETA1,THETA2,D] = meshgrid(theta1,theta2,d);`

4. Memasukkan persamaan X, Y, dan Z

Kemudian bagian *code* yang ada di Tabel 4.12 akan memasukkan setiap kolom isi dari *array-array* tersebut ke persamaan X, Y, dan Z dari *forward kinematics*. Variabel "THETA1" berisi nilai-nilai dari sudut θ_1, "THETA2" berisi nilai-nilai sudut θ_2, dan "D" berisi nilai-nilai dari panjang d.

Tabel 4.12 Implementasi persamaan X, Y, dan Z

Possible_coor_scara.m	
1	`X = l1 * cosd(THETA1) + l2 * cosd(THETA1 + THETA2);`
2	`Y = l1 * sind(THETA1) + l2 * sind(THETA1 + THETA2);`
3	`Z = d1 - (d3 + D);`

5. Menggabungkan kolom dan baris

Pada bagian *code* yang ada di Tabel 4.13, semua kolom dan baris dari *array-array* X, Y, Z, THETA1, THETA2, dan D akan digabungkan dan ditata menjadi 3 *array* baru, yang berisi X, Y, Z, dan variabel *joint*-nya masing masing.

Tabel 4.13 Menggabungkan kolom dan baris dari X, Y, Z, THETA1, THETA2, dan D

Possible_coor_scara.m	
1	`data1 = [X(:) Y(:) Z(:) THETA1(:)];`
2	`data2 = [X(:) Y(:) Z(:) THETA2(:)];`
3	`data3 = [X(:) Y(:) Z(:) D(:)];`

6. Menggabungkan 3 *array* ke dalam 1 *array* hasil

Terakhir pada bagian code yang ada di dalam Tabel 4.14, 3 array data1, data2, dan data3 akan digabungkan kedalam array x_y_theta123. *Array* ini berisi hasil dari setiap theta1, theta2, dan d jika dimasukkan ke dalam persamaan X, Y, dan Z.

Tabel 4.14 Menggabungkan *array* ke dalam 1 *array* hasil

Possible_coor_scara.m	
1	`x_y_theta123 = [data1,data2,data3];`
2	`x_y_theta123(:, 5) = [];`
3	`x_y_theta123(:, 5) = [];`
4	`x_y_theta123(:, 5) = [];`
5	`x_y_theta123(:, 6) = [];`
6	`x_y_theta123(:, 6) = [];`
7	`x_y_theta123(:, 6) = [];`

Maka *array* x_y_theta123 hasilnya adalah seperti gambar 4.15 dibawah. Kolom 1 merupakan kolom titik X, kolom 2 adalah kolom titik Y, kolom 3 adalah kolom titik Z, kolom 4 adalah kolom sudut θ_1 , kolom 5 adalah kolom sudut θ_1, dan kolom terakhir adalah kolom d.

Tabel 4.7 Tabel Area Kerja

X	Y	Z	Theta1	Theta2	d
8,19615	-8,19615	2	-30	-30	3
8,28638	-8,14300	2	-30	-29	3
8,37566	-8,08828	2	-30	-28	3
8,46398	-8,03202	2	-30	-27	3
8,55130	-7,97422	2	-30	-26	3
8,63761	-7,91491	2	-30	-25	3
8,72286	-7,85410	2	-30	-24	3
8,80704	-7,79181	2	-30	-23	3
8,89012	-7,72806	2	-30	-22	3
8,97207	-7,66287	2	-30	-21	3
9,05287	-7,59626	2	-30	-20	3
9,13250	-7,52825	2	-30	-19	3
9,21093	-7,45886	2	-30	-18	3
9,28814	-7,38812	2	-30	-17	3
9,36410	-7,316038	2	-30	-16	3
9,43879	-7,242640	2	-30	-15	3
9,51219	-7,167950	2	-30	-14	3
9,58427	-7,091990	2	-30	-13	3
9,65502	-7,014783	2	-30	-12	3
9,72440	-6,936354	2	-30	-11	3

C. Implementasi *Inverse Kinematics*

1. Implementasi hitung waktu pemrosesan dan masukkan koordinat target

Mulai dari bagian *code* pada tabel, fungsi tic; akan dipanggil. Fungsi ini berfungsi untuk memulai penghitung waktu hingga fungsi toc; dipanggil terakhir. Kemudian ada masukkan untuk posisi target yang akan dicapai.

Tabel 4.15 Fungsi penghitung waktu dan masukkan target posisi

Scara_3dof_IK.m

```
1  tic;
2
3  x = ;
4  y = ;
5  z = ;
6
7  pt = [x;y;z];
```

2. Definisikan panjang lengan robot

Pada bagian *code* yang ada di Tabel 4.16, hasil dari perancangan robot akan didefinisikan, yaitu berupa panjang d1, L1, L2, d3.

Tabel 4.16 Implementasi Panjang Lengan Robot

Scara_3dof_IK.m

```
1  d1 = 8;
2  l1 = 6;
3  l2 = 6;
4  d3 = 3;
```

3. Persamaan θ_1

Pada bagian *code* yang ada di dalam Tabel 4.17, implementasi mencari θ_1 dimulai dari mencari r1, kemudian phi2, kemudian ab, kemudian ab akan di-*inverse-cosinus*-kan. Jika ab bernilai lebih dari 1 dan -1, maka hasilnya akan 0. Ini dikarenakan hasil dari *inverse cosinus* 1 dan -1 adalah 0. Kemudian phi2 akan dikurangi dengan phi1, hasilnya adalah sudut θ_1, sesuai dengan perancangan *Inverse Kinematics.*

Tabel 4.17 Persamaan θ_1

Scara_3dof_IK.m	
1	r1 = sqrt((x^2)+(y^2));
2	phi2 = atan(y/x);
3	ab = ((l2^2)-(l1^2)-(r1^2))/(-2*l1*r1);
4	if ab < 0.9999 && ab > -0.9999
5	phi1 = acos(ab);
6	else
7	phi1 = 0;
8	end
9	t1 = phi2-phi1;

4. Persamaan θ_2

Pada bagian *code* yang ada di dalam Tabel 4.18, implementasi mencari θ_2 dimulai dari menghitung bc, kemudian bc akan di-*inverse-cosinus*-kan. Jika bc bernilai lebih dari 1 dan -1, maka hasilnya akan 0. Ini dikarenakan hasil dari *inverse cosinus* 1 dan -1 adalah 0. Kemudian karena θ_2 akan membentuk 180° dengan phi3, maka 180° akan dikurangi dengan phi3, hasilnya adalah sudut θ_2, sesuai dengan perancangan *Inverse Kinematics.*

Tabel 4.18 Persamaan θ_2

Scara_3dof_IK.m

```
1  bc = ((r1^2)-(l1^2)-(l2^2))/(-2*l1*l2);
2  if bc < 0.9999 && bc > -0.9999
3      phi3 = acos(bc);
4  else
5      phi3 = 0;
6  end
7
8  if phi3 == 0
9      t2 = 0;
10 else
11     t2 = pi-phi3;
12 end
```

5. Persamaan d

Pada bagian *code* yang ada di dalam Tabel 4.19, implementasi mencari d dilakukan dengan cara mengurangi panjang d1 dengan d3, kemudian dikurangi dengan koordinat z target.

Tabel 4.19 Persamaan d

Scara_3dof_IK.m

```
1  d = (d1 - d3) - z;
```

6. Limitasi variabel *joint*

Pada bagian *code* yang ada di dalam Tabel 4.19, dilakukan pengecekan apakah variabel diluar *range* limitasi dilakukan setelah perhitungan. Dimana θ_1, θ_2, dan d akan dibandingkan dengan limitasi variabel masing masing.

Tabel 4.20 Pengecekan Limitasi *Joint*

Scara_3dof_IK.m

```
1  theta1 = (t1/pi)*180;
2  theta2 = (t2/pi)*180;
3
4  if theta1 > 30
5      theta1 = 30;
6  elseif theta1 < -30
7      theta1 = -30;
8  end
9  if theta2 > 30
10     theta2 = 30;
11 elseif theta2 < -30
12     theta2 = -30;
13 end
14
15 if d > 3
16     d = 3;
17 end
```

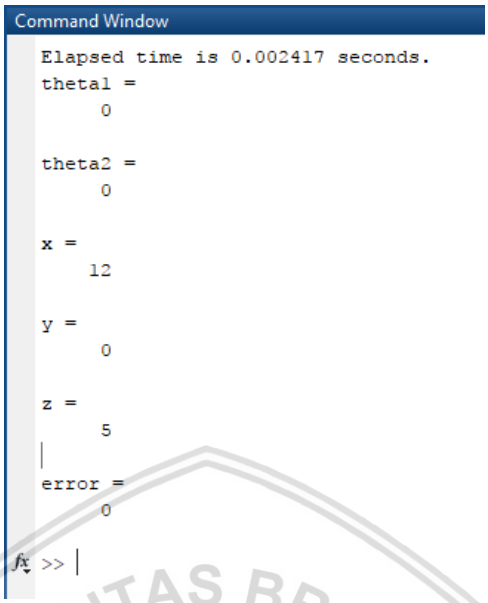
7. Verifikasi dan cek nilai *error*

Pada bagian *code* di dalam Tabel 4.21, setelah selesai pengecekan, maka variabel-variabel tadi akan dimasukkan ke dalam persamaan *forward kinematics* untuk dicek apakah sesuai dengan koordinat yang diinginkan. Kemudian akan dihitung nilai *error* antara hasil perhitungan dan target yang diinginkan. Setelah semuanya selesai, maka akan keluar *output* dari program seperti pada Gambar 4.16. Variabel "theta1" berisi nilai dari θ_1, "theta2" berisi nilai θ_2, dan "d" berisi nilai dari d.

Tabel 4.21 Verifikasi dan hitung nilai *error*

Scara_3dof_IK.m

```
1  pos_x_ee = l1 * cosd(theta1) + l2 * cosd(theta1 +
2  theta2);
3  pos_y_ee = l1 * sind(theta1) + l2 * sind(theta1 +
4  theta2);
5  pos_z_ee = d1 - (d3 + d);
6  error = dist([pos_x_ee pos_y_ee pos_z_ee],pt);
7  toc;
```



Gambar 4.18 Keluaran dari skrip implementasi *Inverse Kinematics*

D. Implementasi *Cyclic Coordinate Descent* (CCD)

1. Fungsi hitung waktu dan masukkan koordinat target

Pada bagian *code* pada Tabel 4.22, fungsi tic; akan dipanggil untuk mulai menghitung waktu pemrosesan dan masukan koordinat target dimasukkan ke dalam vektor pt.

Tabel 4.22 Fungsi tic dan vektor pt

Test_ccd_scara_3dof.m	
1	clc;
2	clear;
3	tic;
4	
5	
6	pt = [x;y;z];

2. Inisiasi variabel

Pada bagian *code* pada Tabel 4.23 ini, variabel-variabel global akan didefinisikan. Seperti num_of_link yang berisi jumlah lengan (L1 dan L2), nilai dari d1, l1, l2, dan d3, kemudian *array* length_arm berisi panjang l1dan l2.

Kemudian ada variabel xdata yang berfungsi sebagai penampung nilai X dari masing masing titik robot dari tampak *planar* (Gambar 4.7), temasuk X dari *end effector*. Kemudian ada variabel "k" sebagai penghitung iterasi dan "kmax" sebagai penentu berapa iterasi maksimal yang dapat ditempuh.

Kemudian ada variabel ydata yang berfungsi sebagai penampung nilai Y dari masing masing titik robot dari tampak *planar* (Gambar 4.6), temasuk Y dari *end effector*. Kemudian ada variabel zdata yang berfungsi sebagai penampung nilai Z dari masing masing titik robot. Dan terakhir variabel angeldata yang berfungsi menampung nilai variabel *joint* dari masing-masing *joint.*

xdata berisi 4 titik, dimana titik pertama bernilai 0 sebagai titik X pada *base* sekaligus *revolute* pertama, kemudian titik kedua yang merupakan ujung dari panjang l1 bernilai 6 sebagai titik X pada *revolute* kedua, kemudian titik ketiga yang merupakan ujung dari l2 bernilai 12 sebagai titik X pada *prismatic*, dan titik keempat akan selalu sama dengan titik ketiga karna sebagai *end effector* yang terus menempel dengan *prismatic.*

ydata berisi 4 titik, dimana semua nilainya bernilai 0 karena ini adalah posisi *home*. zdata berisi 4 titik, dimana titik pertama, kedua, dan ketiga bernilai sama yaitu 8 karena sumbu z pada titik-titik ini dipengaruhi oleh panjang d1. Kemudian zdata titik keempat bernilai 3 yang merupakan panjang d3.

Tabel 4.23 Inisiasi data dan variabel

Test_ccd_scara_3dof.m

```
1  num_of_link = 2;
2  d1 = 8;
3  l1 = 6;
4  l2 = 6;
5  d3 = 3;
6  length_arm = [l1, l2];
7
8
9  xdata(1) = 0;
10 xdata(2) = l1;
11 xdata(3) = l1+l2;
12 xdata(4) = xdata(3);
13
14 k = 0;
15 kmax = 10000;
16
17 for i = 1:num_of_link+2
18     ydata(i) = 0;
19 end
20
21 zdata(1) = d1;
22 zdata(2) = d1;
23 zdata(3) = d1;
24 zdata(4) = d1-d3;
25
26 for i = 1:num_of_link+1
27     angledata(i) = 0;
28 end
```

3. Hitung *error* dan *prismatic joint*

Pada bagian *code* pada Tabel 4.23 ini iterasi CCD dimulai. Pemicu dari iterasi ini adalah nilai *error* yang didapatkan dari perhitungan *Euclidean Distance* antara *end effector* dengan target. Ketika nilai *error* lebih dari 0,5 dan iterasi belum mencapai 0, maka iterasi CCD akan dilakukan. Jika iterasi sudah mencapai 10000 dan/atau nilai *error* kurang dari 0,5 maka iterasi akan berhenti.

Iterasi pertama menghitung selisih jarak antara koordinat Z target dengan koordinat Z dari *end effector*.

Tabel 4.24 Menghitung *error* dan *prismatic joint*

Test_ccd_scara_3dof.m

```
error = dist([xdata(num_of_link+2) ydata(num_of_link+2)
zdata(num_of_link+2)], pt);
while (error > 0.5 && kmax > 0)
    point = num_of_link+1;

        d = zdata(4) - pt(3);
        angledata(3) = d;

        zdata(4) = zdata(4) - d;
```

4. Menghitung sudut antara *joint*, *end effector*, dan koordinat target

Pada bagian *code* dalam Tabel 4.25, selama nilai *point* lebih dari 1, maka penghitungan sudut dan rotasi akan terus dilakukan. *Point* merupakan penanda titik keberapa yang sedang dihitung sudutnya dan dirotasi. "pe" merupakan vektor posisi *end effector* dan pc merupakan vektor posisi *point* yang sedang dirotasi. Selama proses penghitungan sudut ini, persamaan 4.24 berlangsung.

Tabel 4.25 Proses Penghitungan Sudut

Test_ccd_scara_3dof.m

```
while (point > 1)
            pe = [xdata(num_of_link+1);
ydata(num_of_link+1)];
            pc = [xdata(point-1); ydata(point-1)];
            a = (pe - pc)/norm(pe-pc);
            b = ([pt(1);pt(2)] - pc)/norm([pt(1);pt(2)]-
pc);
            ab = dot(a,b);
            if ab < 0.9999
                teta = acosd(ab);
            else
                teta = 0;
            end
```

5. Penentuan arah rotasi dan verifikasi batas sudut

Pada bagian *code* di dalam Tabel 4.26, setelah sudut didapatkan, maka selanjutnya mencari arah putaran dengan menggunakan persamaan 4.23. Kemudian, di bagian ini juga akan dicek apakah sudut yang didapat di luar dari *range* batas sudut yang dapat di tempuh oleh *joint.*

Setelah arah putaran dan sudut didapat, maka akan dimasukkan ke dalam variabel "angledata" menurut *point*-nya. Kemudian, *point* akan dikurangi 1 untuk menghitung titik selanjutnya.

Tabel 4.26 Menentukan arah rotasi dan verifikasi sudut

Test_ccd_scara_3dof.m

```
1           direction = cross([a(1) a(2) 0],[b(1) b(2)
2   0]);
3           if direction(3) < 0
4               teta = -teta;
5           end
6
7
8           if (teta > 30)
9               teta = 30;
10          elseif (teta < -30)
11              teta = -30;
12          end
13
14          angledata(point-1) = teta;
15
16
17          point = point - 1;
```

6. Rotasi *link*

Pada bagian *code* di dalam Tabel 4.27, sudut yang sudah didapat akan langsung dirotasi mulai dari *base* hingga titik sebelum *end effector* yang mempengaruhi posisi dari *end effector* sesuai dengan persamaan 4.24 – 4.28. Setelah rotasi selesai, jika *point* masih > 1 maka akan kembali menghitung sudut untuk *point* setelahnya.

Tabel 4.27 Rotasi *link*

Test_ccd_scara_3dof.m

```
1  for i = 1:num_of_link-1
2                  R = [cosd(angledata(i)) -
3  sind(angledata(i)); sind(angledata(i))
4  cosd(angledata(i))];
5                  temp = R * ([xdata(i+1); ydata(i+1)] -
6  [xdata(i); ydata(i)]) + [xdata(i); ydata(i)];
7                  xdata(i+1) = temp(1);
8                  ydata(i+1) = temp(2);
9                  angledata(i+1) = angledata(i+1) +
10 angledata(i);
11                 xdata(i+2) = xdata(i+1)+length_arm(i+1);
12                 ydata(i+2) = ydata(i+1);
13             end
14
15
16             i = i+1;
17             R = [cosd(angledata(i)) -sind(angledata(i));
18 sind(angledata(i)) cosd(angledata(i))];
19             temp = R * ([xdata(i+1); ydata(i+1)] -
20 [xdata(i); ydata(i)]) + [xdata(i); ydata(i)];
21             xdata(i+1) = temp(1);
22             ydata(i+1) = temp(2);
23             xdata(4) = xdata(3);
24             ydata(4) = ydata(3);
25         end
```

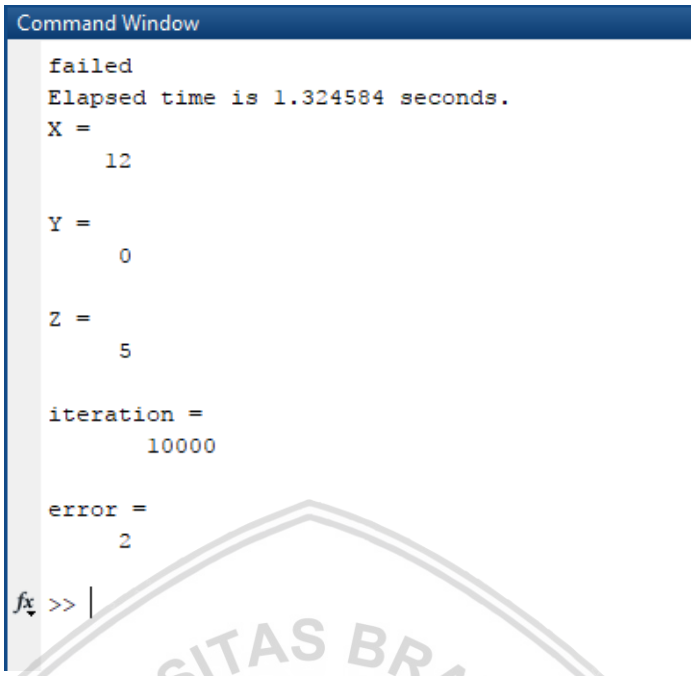
7. Pengecekan *error*

Pada bagian *code* di dalam Tabel 4.28, setelah seluruh point sudah dihitung dan mendapatkan posisi *end effector* yang baru, maka akan dihitung lagi nilai *error*-nya. Jika *error* masih lebih dari 0.5 maka dilihat apakah iterasi sudah mencapai maksimal. Jika dua kondisi tersebut terpenuhi maka akan kembali mengulang iterasi di langkah ketiga.

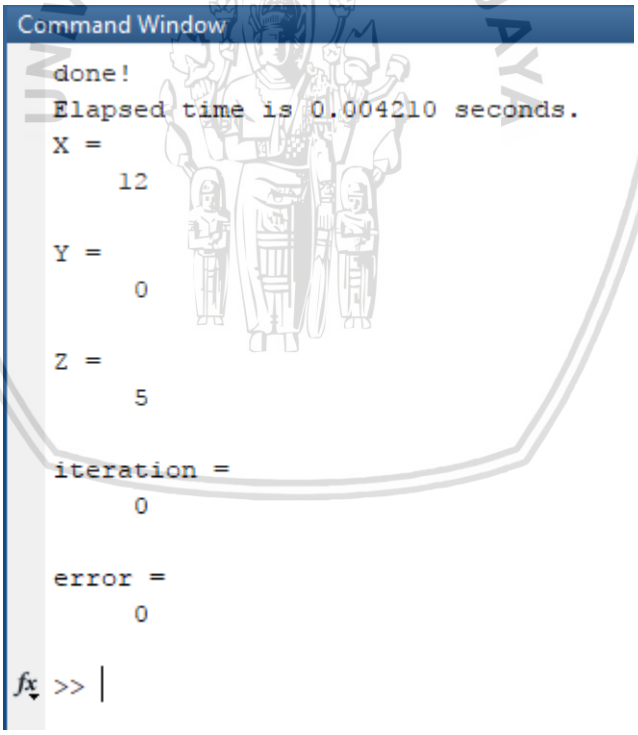
Tabel 4.28 Proses Pengecekan *error* dan Penyelesaian Proses

Test_ccd_scara_3dof.m	
1	kmax = kmax - 1;
2	error = dist([xdata(num_of_link+2)
3	ydata(num_of_link+2) zdata(num_of_link+2)], pt);
4	k = k + 1;
5	
6	end
7	
8	if (error < 0.5)
9	disp('done!');
10	elseif (kmax == 0)
11	disp('failed');
12	end
13	
14	toc;

Jika *error* masuk batas toleransi (kurang dari 0,5), maka proses selesai, muncul teks "Done!" seperti pada Gambar 4.17 dan waktu pemrosesan selesai dihitung dan jika nilai *error* tidak masuk dalam batas toleransi maka akan muncul teks "failed!" seperti pada gambar 4.18.



Gambar 4.19 Keluaran dari skrip implementasi CCD jika nilai *error* masuk toleransi



Gambar 4.20 Keluaran dari skrip implementasi CCD jika nilai *error* masuk toleransi

Untuk membuktikan bahwa implementasi sesuai dengan perhitungan, berikut adalah perhitungan *manual* dari perhitungan untuk salah 1 titik target (11,3,3). Perlu diketahui posisi *end-effector* saat *home* adalah (12,0,5).

A. *Inverse Kinematics*

$$d = (d1 - d3) - z \quad (4.4)$$

$$d = (8 - 3) - 3 \quad (4.30)$$

$$d = 2 \quad (4.31)$$

Pertama, persamaan yang harus diselesaikan adalah translasi dari *prismatic joint*. Seperti yang sudah dijabarkan di bab sebelumnya, persamaan untuk mencari berapa translasi yang harus dicapai untuk mencapai koordinat Z pada titik target menggunakan persamaan 4.4, kemudian dimasukkan panjang lengan d1, d3, dan titik Z target yang harus dituju pada persamaan 4.30 dan hasilnya ada di persamaan 4.31.

$$\Phi_2 = \tan^{-1} \frac{Y_2^0}{X_2^0} \quad (4.7)$$

$$\Phi_2 = \tan^{-1} \frac{3}{11} \quad (4.32)$$

$$\Phi_2 = 15.255 \quad (4.33)$$

Kemudian, persamaan yang harus diselesaikan adalah mencari Φ_2 untuk kemudian dibutuhkan dalam mencari θ_1 . Seperti yang sudah dijabarkan di bab sebelumnya, persamaan untuk mencari berapa Φ_2 menggunakan persamaan 4.7, kemudian dimasukkan titik y dan x target yang harus dituju pada persamaan 4.32 dan hasilnya ada di persamaan 4.33.

$$R1 = \sqrt{(X_2^0)^2 + (Y_2^0)^2} \quad (4.9)$$

$$R1 = \sqrt{(11)^2 + (3)^2} \quad (4.34)$$

$$R1 = 11.4 \quad (4.35)$$

Kemudian, persamaan yang harus diselesaikan adalah mencari R1 untuk kemudian dibutuhkan dalam mencari Φ_1 . Seperti yang sudah dijabarkan di bab sebelumnya, persamaan untuk mencari berapa R1 menggunakan persamaan 4.9, kemudian dimasukkan titik y dan x target yang harus dituju pada persamaan 4.34 dan hasilnya ada di persamaan 4.35.

$$\Phi_1 = \cos^{-1} \left(\frac{L2^2 - L1^2 - R1^2}{-2\ L1\ R1}\right) \quad (4.10)$$

$$\Phi_1 = \cos^{-1} \left(\frac{6^2 - 6^2 - 11{,}4^2}{-2\ 6\ 11{,}4}\right) \quad (4.36)$$

$$\Phi_1 = 18.1680 \quad (4.37)$$

Kemudian, persamaan yang harus diselesaikan adalah mencari Φ_1 untuk kemudian dibutuhkan dalam mencari θ_1 . Seperti yang sudah dijabarkan di bab

sebelumnya, persamaan untuk mencari berapa Φ_1 menggunakan persamaan 4.10, kemudian dimasukkan panjang lengan L2, L1, dan R1 pada persamaan 4.36 dan hasilnya ada di persamaan 4.7.

$$\theta_1 = \Phi_2 - \Phi_1 \quad (4.5)$$
$$\theta_1 = 15.255 - 18.1680 \quad (4.38)$$
$$\theta_1 = -2.913^\circ \quad (4.39)$$

Kemudian, persamaan yang harus diselesaikan adalah mencari θ_1. Seperti yang sudah dijabarkan di bab sebelumnya, persamaan untuk mencari berapa θ_1 menggunakan persamaan 4.5, kemudian dimasukkan Φ_2 dan Φ_1 dan hasilnya ada di persamaan 4.38 dan 4.39.

$$\Phi_3 = \cos\text{-}1 \left(\frac{R1^2 - L1^2 - L2^2}{-2\,L1\,L2}\right) \quad (4.15)$$
$$\Phi_3 = \cos\text{-}1 \left(\frac{11{,}4^2 - 6^2 - 6^2}{-2\,6\,6}\right) \quad (4.40)$$
$$\Phi_3 = 36.3 \quad (4.41)$$

Kemudian, persamaan yang harus diselesaikan adalah mencari Φ_3 untuk kemudian dibutuhkan dalam mencari θ_2 . Seperti yang sudah dijabarkan di bab sebelumnya, persamaan untuk mencari berapa Φ_3 menggunakan persamaan 4.15, kemudian dimasukkan panjang lengan L2, L1, dan R1 pada persamaan 4.40 dan hasilnya ada di persamaan 4.41.

$$\theta_2 = 180^\circ - \Phi_3 \quad (4.13)$$
$$\theta_2 = 180^\circ - 36.3 \quad (4.42)$$
$$\theta_2 = 143.7^\circ \quad (4.43)$$

Kemudian, persamaan yang harus diselesaikan adalah mencari θ_2. Seperti yang sudah dijabarkan di bab sebelumnya, persamaan untuk mencari berapa θ_2 menggunakan persamaan 4.13, kemudian dimasukkan Φ_3 persamaan 4.42 dan hasilnya ada di persamaan 4.43.

Karena sudut maksimal yang dapat ditempuh hanya $-30^\circ < \theta < 30^\circ$, maka θ_2 akan menjadi 30°. Dari sini, didapat θ_1 sebesar -2.913, θ_2 sebesar 30° dan d sebesar 3. Setelah itu θ_1, θ_2, dan *d* akan dimasukkan ke dalam persamaan forward kinematic untuk dihitung dan dilihat koordinat barunya.

$$X = L1 * \cos(\theta1) + L2 * \cos(\theta1 + \theta2) \quad (4.1)$$
$$X = 6 * \cos(-2.913) + 6 * \cos(-2.913 + 30) \quad (4.43)$$
$$X = 4.12822 + 5.3419 \quad (4.44)$$

X = 11.341 (4.45)

Pertama, persamaan yang harus dihitung adalah mencari titik X. Seperti yang sudah dijabarkan di bab sebelumnya, persamaan untuk mencari titik X ada pada persamaan 4.1, kemudian dimasukkan panjang lengan L1, L2, sudut θ_2 dan θ_1 pada persamaan 4.43 dan 4.44. Hasilnya ada di persamaan 4.45.

Y = L1 * sin(θ1) + L2 * sin(θ1 + θ2) (4.2)
Y = 6 * sin(-2.913) + L2 * sin(-2.913 + 30) (4.46)
Y = -0.3 + 2.7 (4.47)
Y = 2.4 (4.48)

Kemudian, persamaan yang harus dihitung adalah mencari titik Y. Seperti yang sudah dijabarkan di bab sebelumnya, persamaan untuk mencari titik Y ada pada persamaan 4.2, kemudian dimasukkan panjang lengan L1, L2, sudut θ_2 dan θ_1 pada persamaan 4.46 dan 4.47. Hasilnya ada di persamaan 4.48.

Z = d1 – (d3 + d) (4.3)
Z = 8 – (3 + 2) (4.49)
Z = 8 – 5 (4.50)
Z = 3 (4.51)

Kemudian, persamaan yang harus dihitung adalah mencari titik Z. Seperti yang sudah dijabarkan di bab sebelumnya, persamaan untuk mencari titik Z ada pada persamaan 4.3, kemudian dimasukkan panjang lengan d1, d2, dan variabel *joint* d pada persamaan 4.49 dan 4.50. Hasilnya ada di persamaan 4.51.

Error = $\sqrt{(X_2 - X_1)^2 + (Y_2 - Y_1)^2 + (Z_2 - Z_1)^2}$ (2.21)
Error = $\sqrt{(11.341 - 11)^2 + (2.4 - 3)^2 + (3 - 3)^2}$ (4.52)
Error = $\sqrt{0.1163 + 0.36 + 0}$ (4.53)
Error = 0.69014 (4.54)

Kemudian, persamaan yang harus dihitung adalah nilai *error* dari akurasi posisi antara *end-effector* hasil perhitungan sebelumnya dan titik target. Seperti yang sudah dijabarkan di bab sebelumnya, persamaan untuk mencari nilai *error* ada pada persamaan 2.21, kemudian dimasukkan titik X, Y, Z hasil persamaan dan X, Y, Z dari titik target pada persamaan 4.52 dan 4.53. Hasilnya ada di persamaan 4.54.

Hasil dari perhitungan manual memberikan hasil X,Y,Z (11.3341;2.4;3). Hasil ini sesuai dengan hasil perhitungan pada MATLAB. Tetapi ada perbedaan hasil perhitungan *error* antara perhitungan *manual* dengan perhitungan pada MATLAB. Hal ini terjadi karena perbedaan penaksiran koma, sehingga akan berbeda hasil juga. Setelah perhitungan menggunakan *inverse kinematic*, selanjutnya adalah perhitungan menggunakan CCD.

$$Error = \sqrt{(X_2 - X_1)^2 + (Y_2 - Y_1)^2 + (Z_2 - Z_1)^2} \quad (2.21)$$

$$Error = \sqrt{(12 - 11)^2 + (0 - 3)^2 + (5 - 3)^2} \quad (4.55)$$

$$Error = \sqrt{1 + 9 + 4} \quad (4.56)$$

$$Error = 3.7417 \quad (4.57)$$

Pertama, persamaan yang harus dihitung adalah nilai *error* dari akurasi posisi antara *end-effector* hasil perhitungan sebelumnya dan titik target. Seperti yang sudah dijabarkan di bab sebelumnya, persamaan untuk mencari nilai *error* ada pada persamaan 2.21, kemudian dimasukkan titik X, Y, Z dari *end-effector* saat ini dan X, Y, Z dari titik target pada persamaan 4.55 dan 4.56. Hasilnya ada di persamaan 4.57.

$$d = (d1 - d3) - z \quad (4.4)$$

$$d = (8 - 3) - 3 \quad (4.58)$$

$$d = 2 \quad (4.59)$$

Kemudian, persamaan yang harus diselesaikan adalah translasi dari *prismatic joint*. Seperti yang sudah dijabarkan di bab sebelumnya, persamaan untuk mencari berapa translasi yang harus dicapai untuk mencapai koordinat Z pada titik target menggunakan persamaan 4.4, kemudian dimasukkan panjang lengan d1, d3, dan titik Z target yang harus dituju pada persamaan 4.58 dan hasilnya ada di persamaan 4.59.

$$Pe = (12;0) \quad (4.60)$$

$$Pc = (6;0) \quad (4.61)$$

$$Pt = (11;3) \quad (4.62)$$

Kemudian menentukan isi dari Pe, Pc, dan Pt. Seperti yang sudah dijabarkan di bab sebelumnya, Pe merupakan vektor yang berisi koordinat dari *end-effector*, Pc merupakan vektor yang berisi koordinat dari titik sebelumnya (bergantung dari lengan mana yang akan dihitung), dan Pt merupakan titik target yang akan dituju.Perhitungan CCD dimulai dari titik sebelum *end-effector*. Maka, Pe berisi (12;0) seperti pada persamaan 4.60, Pc berisi (6;0) seperti pada persamaan 4.61, dan Pt berisi (11;3) seperti pada persamaan 4.62.

$$\theta = \text{acos}^{-1} \left(\frac{\overrightarrow{(Pt - Pc)}}{\left\| \overrightarrow{(Pt - Pc)} \right\|} \cdot \frac{\overrightarrow{(Pe - Pc)}}{\left\| \overrightarrow{(Pe - Pc)} \right\|} \right) \quad (4.20)$$

$$\overrightarrow{(Pt - Pc)} = (11.3) - (6.0) \quad (4.63)$$

$$\overrightarrow{(Pt - Pc)} = (5.3) \quad (4.64)$$

Kemudian, persamaan yang harus diselesaikan adalah mencari $\overrightarrow{(Pt - Pc)}$ untuk kemudian dibutuhkan dalam mencari θ. Seperti yang sudah dijabarkan di bab sebelumnya, persamaan untuk mencari berapa $\overrightarrow{(Pt - Pc)}$ merupakan bagian dari persamaan 4.20, kemudian dimasukkan Pt dan Pc pada persamaan 4.63 dan hasilnya ada di persamaan 4.64.

$$\left\|\overrightarrow{(Pt - Pc)}\right\| = \sqrt{5^2 + 3^2} \quad (4.65)$$

$$\left\|\overrightarrow{(Pt - Pc)}\right\| = \sqrt{34} \quad (4.66)$$

$$\left\|\overrightarrow{(Pt - Pc)}\right\| = 5.8310 \quad (4.67)$$

Kemudian, persamaan yang harus diselesaikan adalah mencari $\left\|\overrightarrow{(Pt - Pc)}\right\|$ untuk kemudian dibutuhkan dalam mencari θ. Seperti yang sudah dijabarkan di bab sebelumnya, persamaan untuk mencari berapa $\left\|\overrightarrow{(Pt - Pc)}\right\|$ merupakan bagian dari persamaan 4.20, kemudian dimasukkan Pt dan Pc pada persamaan 4.65 dan 4.66 dan hasilnya ada di persamaan 4.67.

$$\overrightarrow{(Pe - Pc)} = (12,0) - (6,0) \quad (4.68)$$

$$\overrightarrow{(Pe - Pc)} = (6.0) \quad (4.69)$$

Kemudian, persamaan yang harus diselesaikan adalah mencari $\overrightarrow{(Pe - Pc)}$ untuk kemudian dibutuhkan dalam mencari θ. Seperti yang sudah dijabarkan di bab sebelumnya, persamaan untuk mencari berapa $\overrightarrow{(Pe - Pc)}$ merupakan bagian dari persamaan 4.20, kemudian dimasukkan Pt dan Pc pada persamaan 4.68 dan hasilnya ada di persamaan 4.69.

$$\left\|\overrightarrow{(Pe - Pc)}\right\| = \sqrt{6^2 + 0^2} \quad (4.70)$$

$$\left\|\overrightarrow{(Pe - Pc)}\right\| = \sqrt{36} \quad (4.71)$$

$$\left\|\overrightarrow{(Pe - Pc)}\right\| = 6 \quad (4.72)$$

Kemudian, persamaan yang harus diselesaikan adalah mencari $\left\|\overrightarrow{(Pe - Pc)}\right\|$ untuk kemudian dibutuhkan dalam mencari θ. Seperti yang sudah dijabarkan di bab sebelumnya, persamaan untuk mencari berapa $\left\|\overrightarrow{(Pe - Pc)}\right\|$ merupakan bagian dari persamaan 4.20, kemudian dimasukkan Pe dan Pc pada persamaan 4.70 dan 4.71 dan hasilnya ada di persamaan 4.72.

$$a = \frac{\overrightarrow{(Pt-Pc)}}{\left\|\overrightarrow{(Pt-Pc)}\right\|} \quad (4.73)$$

$$a = \frac{(5.3)}{5.8310} \quad (4.74)$$

$$a = (0.8575;\ 0.5145) \quad (4.75)$$

Kemudian, persamaan yang harus diselesaikan adalah mencari a untuk kemudian dibutuhkan dalam mencari θ. Variabel a merupakan variabel bantu untuk menyelesaikan persamaan. Seperti yang sudah dijabarkan di bab sebelumnya, persamaan untuk mencari berapa a ada pada persamaan 4.73, kemudian dimasukkan $\overrightarrow{(Pt-Pc)}$ dan $\left\|\overrightarrow{(Pt-Pc)}\right\|$ pada persamaan 4.74 dan hasilnya ada di persamaan 4.75.

$$b = \frac{\overrightarrow{(Pe-Pc)}}{\left\|\overrightarrow{(Pe-Pc)}\right\|} \quad (4.76)$$

$$b = \frac{(6;0)}{6} \quad (4.77)$$

$$b = (1;\ 0) \quad (4.78)$$

Kemudian, persamaan yang harus diselesaikan adalah mencari b untuk kemudian dibutuhkan dalam mencari θ. Variabel b merupakan variabel bantu untuk menyelesaikan persamaan. Seperti yang sudah dijabarkan di bab sebelumnya, persamaan untuk mencari berapa a ada pada persamaan 4.76, kemudian dimasukkan $\overrightarrow{(Pe-Pc)}$ dan $\left\|\overrightarrow{(Pe-Pc)}\right\|$ pada persamaan 4.77 dan hasilnya ada di persamaan 4.78.

$$\theta = \text{acos}^{-1}\left(\frac{\overrightarrow{(Pt-Pc)}}{\left\|\overrightarrow{(Pt-Pc)}\right\|} \cdot \frac{\overrightarrow{(Pe-Pc)}}{\left\|\overrightarrow{(Pe-Pc)}\right\|}\right) \quad (4.20)$$

$$\theta = \text{acos}^{-1}\left(\frac{(5;3)}{(5.8310)} \cdot \frac{(6;0)}{(6)}\right) \quad (4.79)$$

$$\theta = \text{acos}^{-1}\left(0.8575\right) \quad (4.80)$$

$$\theta = 30.96° \quad (4.81)$$

Kemudian, persamaan yang harus diselesaikan adalah mencari θ. Seperti yang sudah dijabarkan di bab sebelumnya, persamaan untuk mencari berapa θ ada pada persamaan 4.20, kemudian dimasukkan a dan b pada persamaan 4.79 & 4.80 dan hasilnya ada di persamaan 4.81. Setelah mendapatkan θ, maka yang perlu dicari kemudian adalah arah rotasinya (+/-)

$$\begin{bmatrix} a_x \\ a_y \end{bmatrix} \times \begin{bmatrix} b_x \\ b_y \end{bmatrix} = \begin{bmatrix} \\ a_x b_y - a_y b_x \end{bmatrix} \quad (4.23)$$

$$\begin{bmatrix} 1 \\ 0 \end{bmatrix} \times \begin{bmatrix} 0.8575 \\ 0.5145 \end{bmatrix} = \begin{bmatrix} \\ 1 * 0.5145 - 0 * 0.8575 \end{bmatrix} \quad (4.82)$$

$$\begin{bmatrix} 1 \\ 0 \end{bmatrix} \times \begin{bmatrix} 0.8575 \\ 0.5145 \end{bmatrix} = 0.5145 \quad (4.83)$$

$$0.5145 > 0 \longrightarrow \theta = +30° \quad (4.84)$$

Untuk mencari arah putar, maka dibutuhkan perhitungan matriks seperti pada persamaan 4.23, kemudian komponen dari variabel a dan b dimasukkan pada persamaan 4.82 dan 4.83, dan hasilnya ada pada 4.84. Karena hasilnya kurang dari 0, maka arah putarnya ke arah kanan (-)

$$R = \begin{bmatrix} \cos(\theta) & -\sin(\theta) \\ \sin(\theta) & \cos(\theta) \end{bmatrix} \quad (4.85)$$

$$R = \begin{bmatrix} \cos(30) & -\sin(30) \\ \sin(30) & \cos(30) \end{bmatrix} \quad (4.86)$$

$$R = \begin{bmatrix} 0.866 & -0.5 \\ 0.5 & 0.866 \end{bmatrix} \quad (4.87)$$

Setelah mendapatkan arah putar, maka dilakukan rotasi terhadap sudut dan arahnya. Persamaan matriks rotasi seperti pada 4.85, kemudian sudutnya dimasukkan ke dalam persamaan 4.86, dan hasilnya pada 4.87

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = R * \left(\begin{bmatrix} x_{i+1} \\ y_{i+1} \end{bmatrix} - \begin{bmatrix} x_i \\ y_i \end{bmatrix} \right) + \begin{bmatrix} x_i \\ y_i \end{bmatrix} \quad (4.25)$$

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = R * \left(\begin{bmatrix} x_{1+1} \\ y_{1+1} \end{bmatrix} - \begin{bmatrix} x_1 \\ y_1 \end{bmatrix} \right) + \begin{bmatrix} x_1 \\ y_1 \end{bmatrix} \quad (4.88)$$

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = R * \left(\begin{bmatrix} 12 \\ 0 \end{bmatrix} - \begin{bmatrix} 6 \\ 0 \end{bmatrix} \right) + \begin{bmatrix} 6 \\ 0 \end{bmatrix} \quad (4.89)$$

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \left(R * \begin{bmatrix} 6 \\ 0 \end{bmatrix} \right) + \begin{bmatrix} 6 \\ 0 \end{bmatrix} \quad (4.90)$$

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} 5.196 \\ 3 \end{bmatrix} + \begin{bmatrix} 6 \\ 0 \end{bmatrix} \quad (4.91)$$

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} 11.196 \\ 3 \end{bmatrix} \quad (4.92)$$

Setelah mendapatkan matriks rotasinya, matriks rotasi tadi dikalikan dengan koordinat sebelumnya untuk mendapatkan koordinat baru setelah dirotasi, seperti pada persamaan 4.25, kemudian sesuai dengan pembahasan pada perancangan, matriks rotasi akan diubah menjadi koordinat global sehingga akan menjadi persamaan 4.88 dan 4.89, kemudian diturunkan ke persamaan 4.90 dan 4.91, sehingga hasil koordinat barunya seperti pada persamaan 4.92. Setelah koordinat baru telah didapat, maka koordinat baru tersebut adalah titik *end-effector* (Pe) yang baru.

Pe = (11.1962;3) (4.93)
Pc = (0;0) (4.94)
Pt = (11;3) (4.95)

Kemudian, lanjut ke titik setelahnya (titik 0;0) akan menjadi titik Pc. Sehingga, Pe dan Pc sekarang akan diperbaharui seperti pada persamaan 4.93 dan 4.94. Sedangkan, Pt tetap seperti pada persamaan 4.95

$\overrightarrow{(Pt - Pc)}$ = (11;3) – (0;0) (4.96)
$\overrightarrow{(Pt - Pc)}$ = (11;3) (4.97)

Kemudian, persamaan yang harus diselesaikan adalah mencari $\overrightarrow{(Pt - Pc)}$ untuk kemudian dibutuhkan dalam mencari θ. Seperti yang sudah dijabarkan di bab sebelumnya, persamaan untuk mencari berapa $\overrightarrow{(Pt - Pc)}$ merupakan bagian dari persamaan 4.20, kemudian dimasukkan Pt dan Pc pada persamaan 4.96 dan hasilnya ada di persamaan 4.97.

$\left\|\overrightarrow{(Pt - Pc)}\right\| = \sqrt{11^2 + 3^2}$ (4.98)
$\left\|\overrightarrow{(Pt - Pc)}\right\| = \sqrt{121 + 9}$ (4.99)
$\left\|\overrightarrow{(Pt - Pc)}\right\|$ = 11.4 (4.100)

Kemudian, persamaan yang harus diselesaikan adalah mencari $\left\|\overrightarrow{(Pt - Pc)}\right\|$ untuk kemudian dibutuhkan dalam mencari θ. Seperti yang sudah dijabarkan di bab sebelumnya, persamaan untuk mencari berapa $\left\|\overrightarrow{(Pt - Pc)}\right\|$ merupakan bagian dari persamaan 4.20, kemudian dimasukkan Pt dan Pc pada persamaan 4.98 dan 4.99 dan hasilnya ada di persamaan 4.100.

$\overrightarrow{(Pe - Pc)}$ = (11.1962;3) – (0;0) (4.101)
$\overrightarrow{(Pe - Pc)}$ = (11.1962;3) (4.102)

Kemudian, persamaan yang harus diselesaikan adalah mencari $\overrightarrow{(Pe - Pc)}$ untuk kemudian dibutuhkan dalam mencari θ. Seperti yang sudah dijabarkan di bab sebelumnya, persamaan untuk mencari berapa $\overrightarrow{(Pe - Pc)}$ merupakan bagian dari persamaan 4.20, kemudian dimasukkan Pt dan Pc pada persamaan 4.101 dan hasilnya ada di persamaan 4.102.

$\left\|\overrightarrow{(Pe - Pc)}\right\| = \sqrt{11.1962^2 + 3^2}$ (4.103)
$\left\|\overrightarrow{(Pe - Pc)}\right\| = \sqrt{125.355 + 9}$ (4.104)
$\left\|\overrightarrow{(Pe - Pc)}\right\|$ = 11.59 (4.105)

Kemudian, persamaan yang harus diselesaikan adalah mencari $\left\|\overrightarrow{(Pe-Pc)}\right\|$ untuk kemudian dibutuhkan dalam mencari θ. Seperti yang sudah dijabarkan di bab sebelumnya, persamaan untuk mencari berapa $\left\|\overrightarrow{(Pe-Pc)}\right\|$ merupakan bagian dari persamaan 4.20, kemudian dimasukkan Pt dan Pc pada persamaan 4.103 dan 4.104 dan hasilnya ada di persamaan 4.105.

$$a = \frac{\overrightarrow{(Pt-Pc)}}{\left\|\overrightarrow{(Pt-Pc)}\right\|} \quad (4.106)$$

$$a = \frac{(11;3)}{(11.4)} \quad (4.107)$$

$$a = (0.965;\ 0.263) \quad (4.108)$$

Kemudian, persamaan yang harus diselesaikan adalah mencari a untuk kemudian dibutuhkan dalam mencari θ. Variabel a merupakan variabel bantu untuk menyelesaikan persamaan. Seperti yang sudah dijabarkan di bab sebelumnya, persamaan untuk mencari berapa a ada pada persamaan 4.106, kemudian dimasukkan $\overrightarrow{(Pt-Pc)}$ dan $\left\|\overrightarrow{(Pt-Pc)}\right\|$ pada persamaan 4.107 dan hasilnya ada di persamaan 4.108.

$$b = \frac{\overrightarrow{(Pe-Pc)}}{\left\|\overrightarrow{(Pe-Pc)}\right\|} \quad (4.109)$$

$$b = \frac{(11.1962;3)}{(11.59)} \quad (5.82)$$

$$b = (0.966; 0.2588) \quad (5.83)$$

Kemudian, persamaan yang harus diselesaikan adalah mencari b untuk kemudian dibutuhkan dalam mencari θ. Variabel b merupakan variabel bantu untuk menyelesaikan persamaan. Seperti yang sudah dijabarkan di bab sebelumnya, persamaan untuk mencari berapa a ada pada persamaan 4.109, kemudian dimasukkan $\overrightarrow{(Pe-Pc)}$ dan $\left\|\overrightarrow{(Pe-Pc)}\right\|$ pada persamaan 5.82 dan hasilnya ada di persamaan 5.83.

$$\theta = \text{acos}^{-1}\left(\frac{\overrightarrow{(Pt-Pc)}}{\left\|\overrightarrow{(Pt-Pc)}\right\|} \cdot \frac{\overrightarrow{(Pe-Pc)}}{\left\|\overrightarrow{(Pe-Pc)}\right\|}\right) \quad (4.20)$$

$$\theta = \text{acos}^{-1}\left(\frac{(11;3)}{(11.4)} \cdot \frac{(11.1962;3)}{(11.59)}\right) \quad (4.110)$$

$$\theta = \text{acos}^{-1}(1) \quad (4.111)$$

$$\theta = 0° \quad (4.112)$$

Kemudian, persamaan yang harus diselesaikan adalah mencari θ. Seperti yang sudah dijabarkan di bab sebelumnya, persamaan untuk mencari berapa θ ada

pada persamaan 4.20, kemudian dimasukkan a dan b pada persamaan 4.110 & 4.111 dan hasilnya ada di persamaan 4.112. Setelah mendapatkan θ, maka yang perlu dicari kemudian adalah arah rotasinya (+/-). Tetapi karena hasilnya adalah 0° maka tidak perlu rotasi.

$$Error = \sqrt{(X_2 - X_1)^2 + (Y_2 - Y_1)^2 + (Z_2 - Z_1)^2} \quad (2.21)$$

$$Error = \sqrt{(11{,}1962 - 11)^2 + (3-3)^2 + (3-3)^2} \quad (4.113)$$

$$Error = \sqrt{0{,}0385 + 0 + 0} \quad (4.114)$$

$$Error = 0{,}1962 \quad (4.115)$$

Kemudian, persamaan yang harus dihitung adalah nilai *error* dari akurasi posisi antara *end-effector* hasil perhitungan sebelumnya dan titik target. Seperti yang sudah dijabarkan di bab sebelumnya, persamaan untuk mencari nilai *error* ada pada persamaan 2.21, kemudian dimasukkan titik X, Y, Z hasil persamaan dan X, Y, Z dari titik target pada persamaan 4.113 dan 4.114, maka hasilnya ada pada persamaan 4.115.

Hasil dari perhitungan manual memberikan hasil X,Y,Z (12.9248;-3.9767;5). Hasil ini sesuai dengan hasil perhitungan pada MATLAB. Tetapi ada perbedaan hasil perhitungan *error* antara perhitungan *manual* dengan perhitungan pada MATLAB. Hal ini terjadi karena perbedaan penaksiran koma, sehingga akan berbeda hasil juga.

Untuk pengujian lainnya, berikut adalah perhitungan *manual* salah 1 titik target (13,-4,5). Perlu diketahui posisi *end-effector* saat *home* adalah (14,0,5).

A. Inverse Kinematics

$$d = (d1 - d3) - z \quad (4.4)$$

$$d = (8-3) - 5 \quad (4.116)$$

$$d = 0 \quad (4.117)$$

Pertama, persamaan yang harus diselesaikan adalah translasi dari *prismatic joint*. Seperti yang sudah dijabarkan di bab sebelumnya, persamaan untuk mencari berapa translasi yang harus dicapai untuk mencapai koordinat Z pada titik target menggunakan persamaan 4.4, kemudian dimasukkan panjang lengan d1, d3, dan titik Z target yang harus dituju pada persamaan 4.116 dan hasilnya ada di persamaan 4.117.

$$\Phi_2 = \tan^{-1}\frac{Y_2^0}{X_2^0} \quad (4.7)$$

$$\Phi_2 = \tan^{-1}\frac{-4}{13} \quad (4.118)$$

$$\Phi_2 = \text{-}17.1027 \quad (4.119)$$

Kemudian, persamaan yang harus diselesaikan adalah mencari Φ_2 untuk kemudian dibutuhkan dalam mencari θ_1 . Seperti yang sudah dijabarkan di bab

sebelumnya, persamaan untuk mencari berapa Φ_2 menggunakan persamaan 4.7, kemudian dimasukkan titik y dan x target yang harus dituju pada persamaan 4.118 dan hasilnya ada di persamaan 4.119.

$$R1 = \sqrt{(X_2^0)^2 + (Y_2^0)^2} \quad (4.9)$$

$$R1 = \sqrt{(13)^2 + (-4)^2} \quad (4.120)$$

$$R1 = 13.6 \quad (4.121)$$

Kemudian, persamaan yang harus diselesaikan adalah mencari R1 untuk kemudian dibutuhkan dalam mencari Φ_1 . Seperti yang sudah dijabarkan di bab sebelumnya, persamaan untuk mencari berapa R1 menggunakan persamaan 4.9, kemudian dimasukkan titik y dan x target yang harus dituju pada persamaan 4.120 dan hasilnya ada di persamaan 4.121.

$$\Phi_1 = \cos^{-1}\left(\frac{L2^2 - L1^2 - R1^2}{-2\,L1\,R1}\right) \quad (4.10)$$

$$\Phi_1 = \cos^{-1}\left(\frac{7^2 - 7^2 - 13{,}6^2}{-2\,7\,13.6}\right) \quad (4.122)$$

$$\Phi_1 = 13.4 \quad (4.123)$$

Kemudian, persamaan yang harus diselesaikan adalah mencari Φ_1 untuk kemudian dibutuhkan dalam mencari θ_1 . Seperti yang sudah dijabarkan di bab sebelumnya, persamaan untuk mencari berapa Φ_1 menggunakan persamaan 4.10, kemudian dimasukkan panjang lengan L2, L1, dan R1 pada persamaan 4.122 dan hasilnya ada di persamaan 4.123.

$$\theta_1 = \Phi_2 - \Phi_1 \quad (4.5)$$

$$\theta_1 = -30.5027° \quad (4.124)$$

Kemudian, persamaan yang harus diselesaikan adalah mencari θ_1. Seperti yang sudah dijabarkan di bab sebelumnya, persamaan untuk mencari berapa θ_1 menggunakan persamaan 4.5, kemudian dimasukkan Φ_2 dan Φ_1 dan hasilnya ada di persamaan 4.124. Hasil dari persamaan menunjukkan hasil yang lebih dari sudut yang bisa dicapai oleh *Joint 1* yaitu $-30 < \theta_1 < 30$. Oleh karena itu, θ_1 menjadi -30°

$$\Phi_3 = \cos\text{-}1\left(\frac{R1^2 - L1^2 - L2^2}{-2\,L1\,L2}\right) \quad (4.15)$$

$$\Phi_3 = \cos\text{-}1\left(\frac{13{,}6^2 - 7^2 - 7^2}{-2\,7\,7}\right) \quad (4.125)$$

$$\Phi_3 = 151.6424 \quad (4.126)$$

Kemudian, persamaan yang harus diselesaikan adalah mencari Φ_3 untuk kemudian dibutuhkan dalam mencari θ_2 . Seperti yang sudah dijabarkan di bab sebelumnya, persamaan untuk mencari berapa Φ_3 menggunakan persamaan 4.15, kemudian dimasukkan panjang lengan L2, L1, dan R1 pada persamaan 4.125 dan hasilnya ada di persamaan 4.126.

$\theta_2 = 180^o - \Phi_3$ (4.13)

$\theta_2 = 180^o - 151.6424$ (4.127)

$\theta_2 = 28.35^o$ (4.128)

Kemudian, persamaan yang harus diselesaikan adalah mencari θ_2. Seperti yang sudah dijabarkan di bab sebelumnya, persamaan untuk mencari berapa θ_2 menggunakan persamaan 4.13, kemudian dimasukkan Φ_3 persamaan 4.127 dan hasilnya ada di persamaan 4.128.

Setelah semua variabel *joint* sudah terpenuhi, maka selanjutnya adalah memasukkannya ke persamaan *forward kinematic* untuk dihitung akurasi posisinya.

$X = L1 * \cos(\theta1) + L2 * \cos(\theta1 + \theta2)$ (4.1)

$X = 7 * \cos(-30) + 7 * \cos(-30 + 28.35)$ (4.129)

$X = 6.062 + 6.9965$ (4.130)

$X = 13.0585$ (4.131)

Pertama, persamaan yang harus dihitung adalah mencari titik X. Seperti yang sudah dijabarkan di bab sebelumnya, persamaan untuk mencari titik X ada pada persamaan 4.1, kemudian dimasukkan panjang lengan L1, L2, sudut θ_2 dan θ_1 pada persamaan 4.129 dan 4.130. Hasilnya ada di persamaan 4.131.

$Y = L1 * \sin(\theta1) + L2 * \sin(\theta1 + \theta2)$ (4.2)

$Y = 7 * \sin(-30) + 7 * \sin(-30 + 28,35)$ (4.132)

$Y = -3.5 + -0.2010$ (4.133)

$Y = -3.7016$ (4.134)

Kemudian, persamaan yang harus dihitung adalah mencari titik Y. Seperti yang sudah dijabarkan di bab sebelumnya, persamaan untuk mencari titik Y ada pada persamaan 4.2, kemudian dimasukkan panjang lengan L1, L2, sudut θ_2 dan θ_1 pada persamaan 4.132 dan 4.133. Hasilnya ada di persamaan 4.134.

$Z = d1 – (d3 + d)$ (4.3)

$Z = 8 – (3 + 0)$ (4.135)

$Z = 8 – 3$ (4.136)

$Z = 5$ (4.137)

Kemudian, persamaan yang harus dihitung adalah mencari titik Z. Seperti yang sudah dijabarkan di bab sebelumnya, persamaan untuk mencari titik Z ada pada persamaan 4.3, kemudian dimasukkan panjang lengan d1, d2, dan variabel *joint* d pada persamaan 4.135 dan 4.136. Hasilnya ada di persamaan 4.137.

$$Error = \sqrt{(X_2 - X_1)^2 + (Y_2 - Y_1)^2 + (Z_2 - Z_1)^2} \quad (2.21)$$
$$Error = \sqrt{(13.0585 - 13)^2 + (-3.7016 - (-4))^2 + (5 - 5)^2} \quad (4.138)$$
$$Error = \sqrt{0.0034 + 0.0890 + 0} \quad (4.139)$$
$$Error = 0.303 \quad (4.140)$$

Kemudian, persamaan yang harus dihitung adalah nilai *error* dari akurasi posisi antara *end-effector* hasil perhitungan sebelumnya dan titik target. Seperti yang sudah dijabarkan di bab sebelumnya, persamaan untuk mencari nilai *error* ada pada persamaan 2.21, kemudian dimasukkan titik X, Y, Z hasil persamaan dan X, Y, Z dari titik target pada persamaan 4.138 dan 4.139. Hasilnya ada di persamaan 4.140.

Hasil dari perhitungan manual memberikan hasil X,Y,Z (13,06;-3,7;5). Hasil ini berbeda sedikit dari hasil MATLAB karena terjadi perbedaan saat penaksiran angka. Setelah perhitungan menggunakan *inverse kinematic*, selanjutnya adalah perhitungan menggunakan CCD.

B. Cyclic Coordinate Descent

$$Error = \sqrt{(X_2 - X_1)^2 + (Y_2 - Y_1)^2 + (Z_2 - Z_1)^2} \quad (2.21)$$
$$Error = \sqrt{(14 - 13)^2 + (0 - (-4))^2 + (5 - 5)^2} \quad (4.141)$$
$$Error = \sqrt{1 + 16 + 0} \quad (4.142)$$
$$Error = 4.1231 \quad (4.143)$$

Pertama, persamaan yang harus dihitung adalah nilai *error* dari akurasi posisi antara *end-effector* hasil perhitungan sebelumnya dan titik target. Seperti yang sudah dijabarkan di bab sebelumnya, persamaan untuk mencari nilai *error* ada pada persamaan 2.21, kemudian dimasukkan titik X, Y, Z dari *end-effector* saat ini dan X, Y, Z dari titik target pada persamaan 4.141 dan 4.142. Hasilnya ada di persamaan 4.143.

$$d = (d1 - d3) - z \quad (4.4)$$
$$d = (8 - 3) - 5 \quad (4.144)$$
$$d = 0 \quad (4.145)$$

Kemudian, persamaan yang harus diselesaikan adalah translasi dari *prismatic joint*. Seperti yang sudah dijabarkan di bab sebelumnya, persamaan untuk mencari berapa translasi yang harus dicapai untuk mencapai koordinat Z pada titik target menggunakan persamaan 4.4, kemudian dimasukkan panjang lengan d1, d3, dan titik Z target yang harus dituju pada persamaan 4.144 dan hasilnya ada di persamaan 4.145.

$$Pe = (14;0) \quad (4.146)$$
$$Pc = (7;0) \quad (4.147)$$

Pt = (13;-4) (4.148)

Kemudian menentukan isi dari Pe, Pc, dan Pt. Seperti yang sudah dijabarkan di bab sebelumnya, Pe merupakan vektor yang berisi koordinat dari *end-effector*, Pc merupakan vektor yang berisi koordinat dari titik sebelumnya (bergantung dari lengan mana yang akan dihitung), dan Pt merupakan titik target yang akan dituju.Perhitungan CCD dimulai dari titik sebelum *end-effector*. Maka, Pe berisi (14;0) seperti pada persamaan 4.146, Pc berisi (7;0) seperti pada persamaan 4.147, dan Pt berisi (13;-4) seperti pada persamaan 4.148.

$$\theta = \text{acos}^{-1}\left(\frac{\overrightarrow{(Pt-Pc)}}{\left\|\overrightarrow{(Pt-Pc)}\right\|} \cdot \frac{\overrightarrow{(Pe-Pc)}}{\left\|\overrightarrow{(Pe-Pc)}\right\|}\right) \quad (4.20)$$

$\overrightarrow{(Pt-Pc)}$ = (13;-4) – (7;0) (4.149)

$\overrightarrow{(Pt-Pc)}$ = (6;-4) (4.150)

Kemudian, persamaan yang harus diselesaikan adalah mencari $\overrightarrow{(Pt-Pc)}$ untuk kemudian dibutuhkan dalam mencari θ. Seperti yang sudah dijabarkan di bab sebelumnya, persamaan untuk mencari berapa $\overrightarrow{(Pt-Pc)}$ merupakan bagian dari persamaan 4.20, kemudian dimasukkan Pt dan Pc pada persamaan 4.149 dan hasilnya ada di persamaan 4.150.

$\left\|\overrightarrow{(Pt-Pc)}\right\| = \sqrt{6^2 + -4^2}$ (4.151)

$\left\|\overrightarrow{(Pt-Pc)}\right\| = \sqrt{52}$ (4.152)

$\left\|\overrightarrow{(Pt-Pc)}\right\|$ = 7.2111 (4.153)

Kemudian, persamaan yang harus diselesaikan adalah mencari $\left\|\overrightarrow{(Pt-Pc)}\right\|$ untuk kemudian dibutuhkan dalam mencari θ. Seperti yang sudah dijabarkan di bab sebelumnya, persamaan untuk mencari berapa $\left\|\overrightarrow{(Pt-Pc)}\right\|$ merupakan bagian dari persamaan 4.20, kemudian dimasukkan Pt dan Pc pada persamaan 4.151 dan 4.152 dan hasilnya ada di persamaan 4.153.

$\overrightarrow{(Pe-Pc)}$ = (14;0) – (7;0) (4.154)

$\overrightarrow{(Pe-Pc)}$ = (7;0) (4.155)

Kemudian, persamaan yang harus diselesaikan adalah mencari $\overrightarrow{(Pe-Pc)}$ untuk kemudian dibutuhkan dalam mencari θ. Seperti yang sudah dijabarkan di bab sebelumnya, persamaan untuk mencari berapa $\overrightarrow{(Pe-Pc)}$ merupakan bagian dari persamaan 4.20, kemudian dimasukkan Pt dan Pc pada persamaan 4.154 dan hasilnya ada di persamaan 4.155.

$\left\|\overrightarrow{(Pe-Pc)}\right\| = \sqrt{7^2 + 0^2}$ (4.156)

$\left\|\overrightarrow{(Pe-Pc)}\right\| = \sqrt{49}$ (4.157)

$$\left\|\overrightarrow{(Pe - Pc)}\right\| = 7 \quad (4.158)$$

Kemudian, persamaan yang harus diselesaikan adalah mencari $\left\|\overrightarrow{(Pe - Pc)}\right\|$ untuk kemudian dibutuhkan dalam mencari θ. Seperti yang sudah dijabarkan di bab sebelumnya, persamaan untuk mencari berapa $\left\|\overrightarrow{(Pe - Pc)}\right\|$ merupakan bagian dari persamaan 4.20, kemudian dimasukkan Pe dan Pc pada persamaan 4.156 dan 4.157 dan hasilnya ada di persamaan 4.158.

$$a = \frac{\overrightarrow{(Pt-Pc)}}{\left\|\overrightarrow{(Pt-Pc)}\right\|} \quad (4.159)$$

$$a = \frac{(6;-4)}{7.2111} \quad (4.160)$$

$$a = (0.8321;\ -0.5547) \quad (4.161)$$

Kemudian, persamaan yang harus diselesaikan adalah mencari a untuk kemudian dibutuhkan dalam mencari θ. Variabel a merupakan variabel bantu untuk menyelesaikan persamaan. Seperti yang sudah dijabarkan di bab sebelumnya, persamaan untuk mencari berapa a ada pada persamaan 4.159, kemudian dimasukkan $\overrightarrow{(Pt - Pc)}$ dan $\left\|\overrightarrow{(Pt - Pc)}\right\|$ pada persamaan 4.160 dan hasilnya ada di persamaan 4.161.

$$b = \frac{\overrightarrow{(Pe-Pc)}}{\left\|\overrightarrow{(Pe-Pc)}\right\|} \quad (4.162)$$

$$b = \frac{(7;0)}{7} \quad (4.163)$$

$$b = (1;\ 0) \quad (4.164)$$

Kemudian, persamaan yang harus diselesaikan adalah mencari b untuk kemudian dibutuhkan dalam mencari θ. Variabel b merupakan variabel bantu untuk menyelesaikan persamaan. Seperti yang sudah dijabarkan di bab sebelumnya, persamaan untuk mencari berapa a ada pada persamaan 4.162, kemudian dimasukkan $\overrightarrow{(Pe - Pc)}$ dan $\left\|\overrightarrow{(Pe - Pc)}\right\|$ pada persamaan 4.163 dan hasilnya ada di persamaan 4.164.

$$\theta = \text{acos}^{-1}\left(\frac{\overrightarrow{(Pt-Pc)}}{\left\|\overrightarrow{(Pt-Pc)}\right\|} \cdot \frac{\overrightarrow{(Pe-Pc)}}{\left\|\overrightarrow{(Pe-Pc)}\right\|}\right) \quad (4.20)$$

$$\theta = \text{acos}^{-1}\left((0.8321; -0.5547) \cdot (1; 0)\right) \quad (4.165)$$

$$\theta = \text{acos}^{-1}\left(0.8321\right) \quad (4.166)$$

$$\theta = 33.60° \quad (4.167)$$

Kemudian, persamaan yang harus diselesaikan adalah mencari θ. Seperti yang sudah dijabarkan di bab sebelumnya, persamaan untuk mencari berapa θ ada pada persamaan 4.20, kemudian dimasukkan a dan b pada persamaan 4.165 &

4.166 dan hasilnya ada di persamaan 4.167. Setelah mendapatkan θ, maka yang perlu dicari kemudian adalah arah rotasinya (+/-)

$$\begin{bmatrix} a_x \\ a_y \end{bmatrix} \times \begin{bmatrix} b_x \\ b_y \end{bmatrix} = \begin{bmatrix} \\ a_x b_y - a_y b_x \end{bmatrix} \quad (4.23)$$

$$\begin{bmatrix} 1 \\ 0 \end{bmatrix} \times \begin{bmatrix} 0.8321 \\ -0.5547 \end{bmatrix} = \begin{bmatrix} \\ 1 * -0.5547 - 0 * 0.8321 \end{bmatrix} \quad (4.168)$$

$$\begin{bmatrix} 1 \\ 0 \end{bmatrix} \times \begin{bmatrix} 0.8321 \\ -0.5547 \end{bmatrix} = \text{-0.5547} \quad (4.169)$$

$$\text{-0.5547} < 0 \longrightarrow \theta = -30° \quad (4.170)$$

Untuk mencari arah putar, maka dibutuhkan perhitungan matriks seperti pada persamaan 4.23, kemudian komponen dari variabel a dan b dimasukkan pada persamaan 4.168 dan 4.169. dan hasilnya ada pada 4.170. Karena hasilnya kurang dari 0, maka arah putarnya ke arah kanan (-)

$$R = \begin{bmatrix} \cos(\theta) & -\sin(\theta) \\ \sin(\theta) & \cos(\theta) \end{bmatrix} \quad (4.171)$$

$$R = \begin{bmatrix} \cos(-30) & -\sin(-30) \\ \sin(-30) & \cos(-30) \end{bmatrix} \quad (4.172)$$

$$R = \begin{bmatrix} 0.8660 & 0.5 \\ -0.5 & 0.8660 \end{bmatrix} \quad (4.173)$$

Setelah mendapatkan arah putar, maka dilakukan rotasi terhadap sudut dan arahnya. Persamaan matriks rotasi seperti pada 4.171, kemudian sudutnya dimasukkan ke dalam persamaan 4.172, dan hasilnya pada 4.173

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = R * \left(\begin{bmatrix} x_{i+1} \\ y_{i+1} \end{bmatrix} - \begin{bmatrix} x_i \\ y_i \end{bmatrix} \right) + \begin{bmatrix} x_i \\ y_i \end{bmatrix} \quad (4.25)$$

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = R * \left(\begin{bmatrix} x_{1+1} \\ y_{1+1} \end{bmatrix} - \begin{bmatrix} x_1 \\ y_1 \end{bmatrix} \right) + \begin{bmatrix} x_1 \\ y_1 \end{bmatrix} \quad (4.174)$$

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = R * \left(\begin{bmatrix} 14 \\ 0 \end{bmatrix} - \begin{bmatrix} 7 \\ 0 \end{bmatrix} \right) + \begin{bmatrix} 7 \\ 0 \end{bmatrix} \quad (4.175)$$

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \left(R * \begin{bmatrix} 7 \\ 0 \end{bmatrix} \right) + \begin{bmatrix} 7 \\ 0 \end{bmatrix} \quad (4.176)$$

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} 6{,}062 \\ -3{,}5 \end{bmatrix} + \begin{bmatrix} 7 \\ 0 \end{bmatrix} \quad (4.177)$$

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} 13{,}062 \\ -3.5 \end{bmatrix} \quad (4.178)$$

Setelah mendapatkan matriks rotasinya, matriks rotasi tadi dikalikan dengan koordinat sebelumnya untuk mendapatkan koordinat baru setelah dirotasi, seperti pada persamaan 4.25, kemudian sesuai dengan pembahasan pada perancangan, matriks rotasi akan diubah menjadi koordinat global sehingga akan menjadi persamaan 4.174 dan 4.175, kemudian diturunkan ke persamaan 4.176 dan 4.177, sehingga hasil koordinat barunya seperti pada persamaan 4.178. Setelah koordinat baru telah didapat, maka koordinat baru tersebut adalah titik *end-effector* (Pe) yang baru.

Pe = (13.062;-3.5) (4.179)
Pc = (0;0) (4.180)
Pt = (13;-4) (4.181)

Kemudian, lanjut ke titik setelahnya (titik 0;0) akan menjadi titik Pc. Sehingga, Pe dan Pc sekarang akan diperbaharui seperti pada persamaan 4.179 dan 4.180. Sedangkan, Pt tetap seperti pada persamaan 4.181

$\overrightarrow{(Pt - Pc)}$ = (13;-4) – (0;0) (4.182)
$\overrightarrow{(Pt - Pc)}$ = (13;-4) (4.183)

Kemudian, persamaan yang harus diselesaikan adalah mencari $\overrightarrow{(Pt - Pc)}$ untuk kemudian dibutuhkan dalam mencari θ. Seperti yang sudah dijabarkan di bab sebelumnya, persamaan untuk mencari berapa $\overrightarrow{(Pt - Pc)}$ merupakan bagian dari persamaan 4.20, kemudian dimasukkan Pt dan Pc pada persamaan 4.182 dan hasilnya ada di persamaan 4.183.

$\left\|\overrightarrow{(Pt - Pc)}\right\| = \sqrt{13^2 + -4^2}$ (4.184)
$\left\|\overrightarrow{(Pt - Pc)}\right\| = \sqrt{169 + 16}$ (4.185)
$\left\|\overrightarrow{(Pt - Pc)}\right\|$ = 13.6014 (4.186)

Kemudian, persamaan yang harus diselesaikan adalah mencari $\left\|\overrightarrow{(Pt - Pc)}\right\|$ untuk kemudian dibutuhkan dalam mencari θ. Seperti yang sudah dijabarkan di bab sebelumnya, persamaan untuk mencari berapa $\left\|\overrightarrow{(Pt - Pc)}\right\|$ merupakan bagian dari persamaan 4.20, kemudian dimasukkan Pt dan Pc pada persamaan 4.184 dan 4.185 dan hasilnya ada di persamaan 4.186.

$\overrightarrow{(Pe - Pc)}$ = (13.062;-3.5) – (0;0) (4.187)

$\overrightarrow{(Pe - Pc)}$ = (13.062;-3.5) (4.188)

Kemudian, persamaan yang harus diselesaikan adalah mencari $\overrightarrow{(Pe - Pc)}$ untuk kemudian dibutuhkan dalam mencari θ. Seperti yang sudah dijabarkan di bab sebelumnya, persamaan untuk mencari berapa $\overrightarrow{(Pe - Pc)}$ merupakan bagian dari persamaan 4.20, kemudian dimasukkan Pt dan Pc pada persamaan 4.187 dan hasilnya ada di persamaan 4.188.

$\left\|\overrightarrow{(Pe - Pc)}\right\| = \sqrt{13.062^2 + -3.5^2}$ (4.189)

$\left\|\overrightarrow{(Pe - Pc)}\right\| = \sqrt{170{,}6158 + 12.25}$ (4.190)

$\left\|\overrightarrow{(Pe - Pc)}\right\|$ = 13.5228 (4.191)

Kemudian, persamaan yang harus diselesaikan adalah mencari $\left\|\overrightarrow{(Pe - Pc)}\right\|$ untuk kemudian dibutuhkan dalam mencari θ. Seperti yang sudah dijabarkan di bab sebelumnya, persamaan untuk mencari berapa $\left\|\overrightarrow{(Pe - Pc)}\right\|$ merupakan bagian dari persamaan 4.20, kemudian dimasukkan Pt dan Pc pada persamaan 4.189 dan 4.190 dan hasilnya ada di persamaan 4.191.

$$a = \frac{\overrightarrow{(Pe-Pc)}}{\left\|\overrightarrow{(Pe-Pc)}\right\|} \quad (4.192)$$

$$a = \frac{(13.062;-3.5)}{(13.5228)} \quad (4.193)$$

$$a = (0.9659;-0.2588) \quad (4.194)$$

Kemudian, persamaan yang harus diselesaikan adalah mencari a untuk kemudian dibutuhkan dalam mencari θ. Variabel a merupakan variabel bantu untuk menyelesaikan persamaan. Seperti yang sudah dijabarkan di bab sebelumnya, persamaan untuk mencari berapa a ada pada persamaan 4.192, kemudian dimasukkan $\overrightarrow{(Pt - Pc)}$ dan $\left\|\overrightarrow{(Pt - Pc)}\right\|$ pada persamaan 4.193 dan hasilnya ada di persamaan 4.194.

$$b = \frac{\overrightarrow{(Pt-Pc)}}{\left\|\overrightarrow{(Pt-Pc)}\right\|} \quad (4.195)$$

$$b = \frac{(13;4)}{(13.6014)} \quad (4.196)$$

$$b = (0.9558;-0.2941) \quad (4.197)$$

Kemudian, persamaan yang harus diselesaikan adalah mencari b untuk kemudian dibutuhkan dalam mencari θ. Variabel b merupakan variabel bantu untuk menyelesaikan persamaan. Seperti yang sudah dijabarkan di bab sebelumnya, persamaan untuk mencari berapa a ada pada persamaan 4.195, kemudian dimasukkan $\overrightarrow{(Pe - Pc)}$ dan $\left\|\overrightarrow{(Pe - Pc)}\right\|$ pada persamaan 4.196 dan hasilnya ada di persamaan 4.197.

$$\theta = \text{acos}^{-1}\left(\frac{\overrightarrow{(Pt-Pc)}}{\left\|\overrightarrow{(Pt-Pc)}\right\|} \cdot \frac{\overrightarrow{(Pe-Pc)}}{\left\|\overrightarrow{(Pe-Pc)}\right\|}\right) \quad (4.20)$$

$$\theta = \text{acos}^{-1}\left(\frac{(13;-4)}{(13.6014)} \cdot \frac{(13.062;-3.5)}{(13.5228)}\right) \quad (4.198)$$

$$\theta = \text{acos}^{-1}(0.9993) \quad (4.199)$$

$$\theta = 2.1027° \quad (4.200)$$

Kemudian, persamaan yang harus diselesaikan adalah mencari θ. Seperti yang sudah dijabarkan di bab sebelumnya, persamaan untuk mencari berapa θ ada pada persamaan 4.20, kemudian dimasukkan a dan b pada persamaan 4.198 & 4.199 dan hasilnya ada di persamaan 4.200. Setelah mendapatkan θ, maka yang perlu dicari kemudian adalah arah rotasinya (+/-)

$$\begin{bmatrix} a_x \\ a_y \end{bmatrix} \times \begin{bmatrix} b_x \\ b_y \end{bmatrix} = \begin{bmatrix} \\ a_x b_y - a_y b_x \end{bmatrix} \quad (4.23)$$

$$\begin{bmatrix} 0.9659 \\ -0.2588 \end{bmatrix} \times \begin{bmatrix} 0.9588 \\ -0.2941 \end{bmatrix} = \begin{bmatrix} \\ (0.9659 * -0.2941) - (-0.2588 * 0.9558) \end{bmatrix} \quad (4.201)$$

$$\begin{bmatrix} 0.9659 \\ -0.2588 \end{bmatrix} \times \begin{bmatrix} 0.9588 \\ -0.2941 \end{bmatrix} = -0.0367 \quad (4.202)$$

$$-0.0367 < 0 \longrightarrow \theta = -2.1027° \quad (4.203)$$

Untuk mencari arah putar, maka dibutuhkan perhitungan matriks seperti pada persamaan 4.23, kemudian komponen dari variabel a dan b dimasukkan pada persamaan 4.201 dan 4.202. dan hasilnya ada pada 4.203. Karena hasilnya kurang dari 0, maka arah putarnya ke arah kanan (-)

$$R = \begin{bmatrix} \cos(\theta) & -\sin(\theta) \\ \sin(\theta) & \cos(\theta) \end{bmatrix} \quad (4.204)$$

$$R = \begin{bmatrix} \cos(2.1027) & -\sin(2.1027) \\ \sin(2.1027) & \cos(2.1027) \end{bmatrix} \quad (4.205)$$

$$R = \begin{bmatrix} 0.9993 & 0.0367 \\ -0.0367 & 0.9993 \end{bmatrix} \quad (4.206)$$

Setelah mendapatkan arah putar, maka dilakukan rotasi terhadap sudut dan arahnya. Persamaan matriks rotasi seperti pada 4.171, kemudian sudutnya dimasukkan ke dalam persamaan 4.172, dan hasilnya pada 4.173

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = R * \left(\begin{bmatrix} x_{i+1} \\ y_{i+1} \end{bmatrix} - \begin{bmatrix} x_i \\ y_i \end{bmatrix} \right) + \begin{bmatrix} x_i \\ y_i \end{bmatrix} \quad (4.25)$$

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = R * \left(\begin{bmatrix} x_{1+1} \\ y_{1+1} \end{bmatrix} - \begin{bmatrix} x_1 \\ y_1 \end{bmatrix} \right) + \begin{bmatrix} x_1 \\ y_1 \end{bmatrix} \quad (4.207)$$

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = R * \left(\begin{bmatrix} 7 \\ 0 \end{bmatrix} - \begin{bmatrix} 0 \\ 0 \end{bmatrix} \right) + \begin{bmatrix} 0 \\ 0 \end{bmatrix} \quad (4.208)$$

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \left(R * \begin{bmatrix} 7 \\ 0 \end{bmatrix} \right) \quad (4.209)$$

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} 6.9551 \\ -0.0367 \end{bmatrix} \quad (4.210)$$

Setelah mendapatkan matriks rotasinya, matriks rotasi tadi dikalikan dengan koordinat sebelumnya untuk mendapatkan koordinat baru setelah dirotasi, seperti pada persamaan 4.25, kemudian sesuai dengan pembahasan pada perancangan, matriks rotasi akan diubah menjadi koordinat global sehingga akan menjadi persamaan 4.207 dan 4.208, kemudian diturunkan ke persamaan 4.209, sehingga hasil koordinat barunya seperti pada persamaan 4.210.

Setelah koordinat baru telah didapat, maka koordinat baru tersebut adalah titik ujung dari L1 yang baru. Setelah lengan L1 diputar dan mendapatkan titik koordinat baru, maka L2 akan ikut berputar karena terpengaruh putaran lengan L1. Untuk menghitung titik koordinat baru dari *end-effector*, dibutuhkan sudut baru karena perubahan dari rotasi L1.

$$\theta' = \theta_{i-1} + \theta_i \quad (4.211)$$
$$\theta' = -2.1027 + -30 \quad (4.212)$$
$$\theta' = -32.1027^\circ \quad (4.213)$$

Sudut yang baru didapat dari penjumlahan antara sudut yang lama, dengan sudut dari L1 seperti pada persamaan 4.211. Kemudian nilai sudut-sudut tersebut dimasukkan seperti pada persamaan 4.212, kemudian hasilnya seperti pada persamaan 4.213.

Selain sudut yang berubah, koordinat baru dari end-effector juga berubah akibat rotasi L1 :

$$x' = x_2 + L2 \quad (4.214)$$
$$x' = 6.9551 + 7 \quad (4.215)$$
$$x' = 13.9551 \quad (4.216)$$

$$y' = y_2 \quad (4.217)$$

$$y' = -0.2569 \quad (4.218)$$

Perubahan koordinat seperti persamaan di atas terjadi karena untuk me-reset posisi *end-effector* untuk iterasi berikutnya. Titik *end-effector* X yang baru merupakan hasil penjumlahan titik X dengan panjang lengan L2 seperti pada persamaan 4.214, kemudian titik X dan panjang L2 dimasukkan ke dalam persamaan seperti pada 4.215 dan hasilnya adalah seperti pada persamaan 4.216. Titik *end-effector* Y yang baru merupakan garis lurus dari titik Y dari titik sebelumnya seperti pada persamaan 4.217, kemudian hasilnya ada pada persamaan 4.218

$$R = \begin{bmatrix} \cos(\theta) & -\sin(\theta) \\ \sin(\theta) & \cos(\theta) \end{bmatrix} \quad (4.219)$$

$$R = \begin{bmatrix} \cos(-32.1027) & -\sin(-32.1027) \\ \sin(-32.1027) & \cos(-32.1027) \end{bmatrix} \quad (4.220)$$

$$R = \begin{bmatrix} 0.8471 & 0.5314 \\ -0.5314 & 0.8471 \end{bmatrix} \quad (4.221)$$

Setelah mendapatkan sudut baru, maka dilakukan rotasi terhadap sudut dan arahnya. Persamaan matriks rotasi seperti pada 4.219, kemudian sudutnya dimasukkan ke dalam persamaan 4.220, dan hasilnya pada 4.221

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = R * \left(\begin{bmatrix} x_{i+1} \\ y_{i+1} \end{bmatrix} - \begin{bmatrix} x_i \\ y_i \end{bmatrix} \right) + \begin{bmatrix} x_i \\ y_i \end{bmatrix} \quad (4.25)$$

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = R * \left(\begin{bmatrix} x_{2+1} \\ y_{2+1} \end{bmatrix} - \begin{bmatrix} x_2 \\ y_2 \end{bmatrix} \right) + \begin{bmatrix} x_2 \\ y_2 \end{bmatrix} \quad (4.222)$$

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = R * \left(\begin{bmatrix} 13.9951 \\ -0.2569 \end{bmatrix} - \begin{bmatrix} 6.9951 \\ -0.2569 \end{bmatrix} \right) + \begin{bmatrix} 6.9951 \\ -0.2569 \end{bmatrix} \quad (4.223)$$

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \left(R * \begin{bmatrix} 7 \\ 0 \end{bmatrix} \right) + \begin{bmatrix} 6.9951 \\ -0.2569 \end{bmatrix} \quad (4.224)$$

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} 12.9248 \\ -3.9767 \end{bmatrix} \quad (4.225)$$

Setelah mendapatkan matriks rotasinya, matriks rotasi tadi dikalikan dengan koordinat sebelumnya untuk mendapatkan koordinat baru setelah dirotasi, seperti pada persamaan 4.25, kemudian sesuai dengan pembahasan pada perancangan, matriks rotasi akan diubah menjadi koordinat global sehingga akan

menjadi persamaan 4.222 dan 4.223, kemudian diturunkan ke persamaan 4.224, sehingga hasil koordinat barunya seperti pada persamaan 4.225.

$$Error = \sqrt{(X_2 - X_1)^2 + (Y_2 - Y_1)^2 + (Z_2 - Z_1)^2} \quad (2.21)$$

$$Error = \sqrt{(12.9248 - 13)^2 + ((-3.9767) - (-4))^2 + (5 - 5)^2} \quad (4.226)$$

$$Error = \sqrt{0.0057 + 0.0005 + 0} \quad (4.227)$$

$$Error = 0.0787 \quad (4.228)$$

Kemudian, persamaan yang harus dihitung adalah nilai *error* dari akurasi posisi antara *end-effector* hasil perhitungan sebelumnya dan titik target. Seperti yang sudah dijabarkan di bab sebelumnya, persamaan untuk mencari nilai *error* ada pada persamaan 2.21, kemudian dimasukkan titik X, Y, Z hasil persamaan dan X, Y, Z dari titik target pada persamaan 4.226 dan 4.227. Hasilnya ada di persamaan 4.228.

Hasil dari perhitungan manual memberikan hasil X,Y,Z (12.9248;-3.9767;5). Hasil ini sesuai dengan hasil perhitungan pada MATLAB. Tetapi ada perbedaan hasil perhitungan *error* antara perhitungan *manual* dengan perhitungan pada MATLAB. Hal ini terjadi karena perbedaan penaksiran koma, sehingga akan berbeda hasil juga.

BAB 5 PENGUJIAN DAN ANALISIS

Pada bab ini akan menjelaskan terkait dengan pengujian dimulai dari skenario pengujian, proses pengujian serta analisis terhadap data hasil dari pengujian yang dilakukan. Proses pengujian dilakukan untuk mengetahui apakah sistem yang sudah dibuat sesuai dengan analisis kebutuhan yang diinginkan. Pengujian terhadap sistem dilakukan dalam beberapa tahapan yang sesuai dengan kebutuhan yang diinginkan, sedangkan analisis dilakukan agar dapat ditarik kesimpulan dari penelitian yang sudah dilakukan.

5.1 Pengujian Besar Kesalahan pada Posisi

Pengujian tingkat akurasi posisi dilakukan untuk mengukur seberapa dekat titik koordinat posisi *end effector* dengan titik koordinat dari target. Pengujian dilakukan dengan 10 koordinat berbeda yang akan dijadikan masukan dalam perhitungan *Inverse* dan CCD, dimana 10 koordinat tersebut tidak boleh diluar area kerja dari robot.

Metode yang digunakan adalah perhitungan nilai kesalahan dengan menggunakan *Euclidean Distance. Euclidean Distance* merupakan perhitungan untuk mencari garis lurus antara 2 titik dalam ruang 3 dimensi. Garis lurus ini akan menjadi jarak antara 2 titik tersebut, dimana semakin kecil jaraknya, berarti semakin akurat titik koordinat hasil perhitungan 2 metode dan titik target.

Kemudian perbandingan lainnya ada lah pada waktu pemrosesan. Waktu pemrosesan mulai diukur pada saat kode dieksekusi hingga keluar hasil perhitungannya yaitu sudut dan posisi. Semakin cepat waktu pemrosesan maka semakin baik kinerja proses. Waktu pemrosesan diukur menggunakan fungsi *tic* dan *toc* pada MATLAB. Dimana *tic* ditaruh di awal skrip dan *toc* ditaruh di akhir skrip.

5.1.1 Tujuan Pengujian

Hasil dari perhitungan *distance* dan waktu pemrosesan dari metode kinematika *inverse* akan dibandingkan dengan hasil dari perhitungan *distance* dan waktu pemrosesan dari metode kinematika CCD. Dari hasil pengujian dapat diketahui tingkat akurasi dan cepatnya dari 2 metode tersebut.

5.1.2 Prosedur Pengujian

Tahap pengujian akurasi posisi sebagai berikut :

1. Masukkan data robot berupa d1, l1, l2, dan d3 ke dalam skrip area kerja.
2. Eksekusi skrip area kerja untuk memilih koordinat yang akan diuji pada *array* x_y_theta123.
3. Pilih dan catat 10 koordinat yang akan diuji.
4. Masukkan data koordinat ke dalam skrip *Inverse* dan CCD.

5. Eksekusi masing-masing skrip.
6. Catat koordinat hasil skrip, nilai *error* dan waktu pemrosesannnya
7. Masukkan hasil pengujian pada tabel
8. Untuk 1 koordinat uji, lakukan 10 kali pengujian. Sehingga akan menghasilkan 1 nilai rata-rata

5.1.3 Hasil Pengujian

Hasil perhitungan nilai kesalahan akurasi posisi dapat dilihat pada Tabel 5.1 dan 5.2:

Tabel 5.1 Pengujian Nilai Kesalahan Posisi dan Waktu menggunakan perhitungan *Inverse*

Koordinat Uji			Koordinat Praktik			*Error*	Waktu Pemrosesan (s)
X	Y	Z	X’	Y’	Z’		
11	3	3	11.3341	2.4272	3	0.6632	0.000889
8	7	5	10.1824	5.5382	5	2.6268	0.014748
10	-5	2	11.1962	-3	2	2.3304	0.004935
8	-8	2	11.1962	-3	2	5.9343	0.003369
11	1	5	11.5771	-0.5691	5	1.6718	0.003198
11	-4	2	11.1774	-3.4734	2	0.5557	0.005437
9	-7	2	11.1962	-3	2	4.5632	0.003604
10	6	4	10	6	4	0	0.003259
9	7	2	9.5288	6.5997	2	0.6632	0.000467
10	5	4	10.8736	4.0149	4	1.3167	0.00418
Rata-rata *Error*						2.03253	0.00441

Tabel 5.2 Pengujian Nilai Kesalahan Posisi dan Waktu menggunakan perhitungan CCD

Koordinat Uji			Koordinat Praktik			Iterasi	*Error*	Waktu Pemrosesan (s)
X	Y	Z	X’	Y’	Z’			
11	3	3	11.1962	3	3	1	0.1962	0.00666
8	7	5	8.0479	7.0419	5	3	0.0636	0.03162
10	-5	2	10.3674	-5.1837	2	1	0.4108	0.05382
8	-8	2	8.1962	-8.1962	2	1	0.2774	0.03582
11	1	5	11.8792	1.0799	5	10000	0.8828	1.45153

11	-4	2	10.8932	-3.9612	2	1	0.1136	0.04540
9	-7	2	9.1495	-7.1163	2	1	0.1894	0.04984
10	6	4	9.9393	5.9636	4	1	0.0708	0.03887
9	7	2	9.1495	7.1163	2	1	0.1894	0.03281
10	5	4	10.3674	5.1837	4	1	0.4108	0.00192
Rata-rata *Error*							0.2660	0.17483

5.1.4 Analisis Pengujian

Berdasarkan hasil pengujian dapat dikatakan sistem telah berjalan sesuai dengan yang diharapkan. Meskipun hasil pengujian pada CCD tetap menemui *error* namun dalam nilai yang cukup kecil. Pada saat menggunakan perhitungan kinematika *Inverse* rata-rata kesalahan sebesar 2.03253 dan ketika menggunakan perhitungan kinematika CCD rata-rata kesalahan sebesar 0.2660. Kesalahan yang terjadi disebabkan oleh limitasi sudut yang dapat dicapai oleh masing-masing penggerak. Namun, dilihat dari waktu pemrosesan, kinematika *Inverse* lebih cepat karena hanya memproses 1 tahap/iterasi. Berbeda dengan CCD yang melakukan iterasi untuk mendapatkan posisi yang lebih akurat.

Pada metode CCD, terjadi perhitungan berulang hingga mendapat nilai *error* di bawah 0.5 seperti yang sudah dijelaskan pada Perancangan sehingga tercapai posisi yang lebih akurat. Sedangkan pada metode *inverse* mempunyai waktu pemrosesan rata-rata tercepat dikarenakan hanya mengalami 1 kali perhitungan dibanding CCD yang mengalami perhitungan berulang.

Menurut hasil pengujian, dapat disimpulkan bahwa yang pengaruhi akurasi adalah panjang lengan. Kesimpulan ini dapat dilihat ketika pengujian dengan metode *Inverse Kinematic*, besar *error* terbesar terjadi pada koordinat (8;-8;2). Dimana semakin pendek panjang lengan, semakin akurat akurasinya. Ini sesuai dengan pernyataan "Salah satu sebab *error* pada akurasi adalah panjang lengan" (Zhou).

5.2 Pengujian Besar Kesalahan Posisi dengan Panjang Lengan yang Berbeda

Pengujian ini dilakukan untuk mengukur seberapa dekat titik koordinat posisi *end effector* dengan titik koordinat dari target jika panjang lengan diubah. Pengujian dilakukan dengan pasangan 5 koordinat dan 5 panjang lengan L1 + L2 + D3 berbeda yang akan dijadikan masukan dalam perhitungan *Inverse* dan CCD, dimana 10 koordinat tersebut tidak boleh diluar area kerja dari robot.

Metode yang digunakan adalah perhitungan nilai kesalahan dengan menggunakan *Euclidean Distance. Euclidean Distance* merupakan perhitungan untuk mencari garis lurus antara 2 titik dalam ruang 3 dimensi. Garis lurus ini akan menjadi jarak antara 2 titik tersebut, dimana semakin kecil jaraknya, berarti semakin akurat titik koordinat hasil perhitungan 2 metode dan titik target.

Kemudian perbandingan lainnya ada lah pada waktu pemrosesan. Waktu pemrosesan mulai diukur pada saat kode dieksekusi hingga keluar hasil perhitungannya yaitu sudut dan posisi. Semakin cepat waktu pemrosesan maka semakin baik kinerja proses. Waktu pemrosesan diukur menggunakan fungsi *tic* dan *toc* pada MATLAB. Dimana *tic* ditaruh di awal skrip dan *toc* ditaruh di akhir skrip.

5.2.1 Tujuan Pengujian

Hasil dari perhitungan *distance* dan waktu pemrosesan dari metode kinematika *inverse* akan dibandingkan dengan hasil dari perhitungan *distance* dan waktu pemrosesan dari metode kinematika CCD. Dari hasil pengujian dapat diketahui tingkat akurasi dan cepatnya dari 2 metode tersebut.

Selain itu, akan dilihat juga apakah dengan mempunyai panjang lengan yang berbeda akan mempengaruhi akurasi dan cepatnya waktu pemrosesan dari masing-masing kinematika.

5.2.2 Prosedur Pengujian

Tahap pengujian akurasi posisi sebagai berikut :

1. Masukkan data robot berupa d1, l1, l2, dan d3 ke dalam skrip area kerja.
2. Eksekusi skrip area kerja untuk memilih koordinat yang akan diuji pada *array* x_y_theta123.
3. Pilih dan catat 10 koordinat yang akan diuji.
4. Masukkan data koordinat ke dalam skrip *Inverse* dan CCD.
5. Eksekusi masing-masing skrip.
6. Catat koordinat hasil skrip, nilai *error* dan waktu pemrosesannnya
7. Masukkan hasil pengujian pada tabel
8. Untuk 1 koordinat uji, lakukan 10 kali pengujian. Sehingga akan menghasilkan 1 nilai rata-rata

5.2.3 Hasil Pengujian

Hasil perhitungan nilai kesalahan akurasi posisi dapat dilihat pada Tabel 5.3 dan 5.4 :

Tabel 5.3 Pengujian Nilai Kesalahan Posisi dan Waktu menggunakan perhitungan *Inverse*

Nomor	Panjang Lengan			Koordinat Uji			Koordinat Praktik			*Error*	Waktu Pemrosesan (s)
	L1	L2	D3	X	Y	Z	X'	Y'	Z'		
1	5	6	3	10	1	2	10.5982	-0.8	2	1.8968	0.00505
2	5	4	4	6	5	3	7.765	3.9174	3	2.0706	0.00502
3	7	8	2	14	3	2	14.2909	2.4008	3	1.2015	0.00556
4	6	8	4	12	5	2	13.103	3.3839	2	1.9566	0.00528
5	7	7	3	13	-4	5	13.4613	-3.8461	5	0.1914	0.00376
Rata-rata										1.7814	0.00493

Tabel 5.4 Pengujian Nilai Kesalahan Posisi dan Waktu menggunakan perhitungan CCD

Panjang Lengan			Koordinat Uji			Koordinat Praktik			Iterasi	*Error*	Waktu Pemrosesan (s)
L1	L2	D3	X	Y	Z	X'	Y'	Z'			
5	6	3	10	1	2	10.8517	1.0852	2	10000	0.8559	1.46021
5	4	4	6	5	3	6.1795	5.1496	3	3	0.2337	0.00573
7	8	2	14	3	2	14.4762	3.1020	2	4	0.4870	0.03357
6	8	4	12	5	2	12.2328	5.097	2	4	0.2522	0.03890
7	7	3	13	-4	5	12.925	-3.9769	5	1	0.0785	0.05265
Rata-rata										0.3551	0.31998

5.2.4 Analisis Pengujian

Berdasarkan hasil pengujian dapat dikatakan sistem telah berjalan sesuai dengan yang diharapkan. Meskipun hasil pengujian pada CCD tetap menemui *error* namun dalam nilai yang cukup kecil. Pada saat menggunakan perhitungan kinematika *Inverse* rata-rata kesalahan sebesar 1.7814 dan ketika menggunakan perhitungan kinematika CCD rata-rata kesalahan sebesar 0.3552. Kesalahan yang terjadi disebabkan oleh limitasi sudut yang dapat dicapai oleh masing-masing penggerak.

Namun, dilihat dari waktu pemrosesan, kinematika *Inverse* lebih cepat karena hanya memproses 1 tahap/iterasi. Berbeda dengan CCD yang melakukan iterasi untuk mendapatkan posisi yang lebih akurat. Penutup

Bagian ini memuat kesimpulan dan saran terhadap skripsi. Kesimpulan dan saran disajikan secara terpisah, dengan penjelasan sebagai berikut:

BAB 6 KESIMPULAN

Bagian ini memuat kesimpulan dan saran terhadap skripsi. Kesimpulan dan saran disajikan secara terpisah, dengan penjelasan sebagai berikut:

6.1 Kesimpulan

Pada bab ini akan menjawab pertanyaan dari rumusan masalah yang digunakan sebagai kesimpulan yang diambil pada penulisan tugas akhir ini. Skenario yang digunakan dalam pengujian metode kinematika *inverse kinematics* dengan *cyclic coordinate descent* adalah dengan memasukkan 10 koordinat berbeda dan 5 panjang lengan yang berbeda dalam persamaannya. Pada pengujian menggunakan 2 metode kinematika yang telah diuji pada skenario tersebut dapat ditunjukkan sebagai berikut :

1. Dalam penelitian ini, menerapkan metode *Inverse Kinematics* dan *Iterative* dilakukan melalui beberapa tahap. Pertama merancang robot, menganalisis cara kerja kinematika, dan membangun persamaan kinematika untuk mendapatkan variabel *joint*. *Inverse Kinematic* dan *Cyclic Coordinate Descent* bekerja dengan memasukkan *input* berupa panjang lengan dan titik target ke dalam persamaan dan *output*-nya adalah variabel *joint* untuk masing-masing sendi (*joint*)

2. Untuk mengetahui pengaruh dari koordinat akhir dan panjang lengan, perlu dilakukan pengujian. Dalam penelitian ini, pengujian dilakukan dengan memasukkan parameter pengujian ke dalam persamaan yang sudah diimplementasikan sebelumnya. Setelah data parameter diproses, kemudian akan diukur keakuratan posisi dan kecepatan waktu pemrosesannya.

3. Dari hasil pengujian, terlihat metode *Iterative* lebih akurat dengan nilai kesalahan rata-rata 0,24274 dan 0,3815 dari 2 pengujian dengan parameter pengujian yang berbeda, dibandingkan dengan metode *Inverse* mempunyai nilai kesalahan rata-rata 2,03253 dan 1,46338 dari 2 pengujian dengan parameter pengujian yang berbeda. Tetapi, metode *inverse* terlihat lebih cepat dalam pengujian waktu pemrosesan, dengan waktu rata-rata 0,00441 dan 0,00493 detik dari 2 pengujian yang berbeda. Dibandingkan dengan metode *iterative*, waktu pemrosesan rata-rata 0,17483 dan 0,31998 detik dari 2 pengujian yang berbeda.

6.2 Saran

Penulisan tugas akhir ini tentunya masih ada beberapa kekurangan yang masih belum dapat dikerjakan dengan baik. Saran yang dapat dilakukan pada penelitian selanjutnya adalah sebagai berikut :

a. Skenario pengujian dapat dilakukan dengan konfigurasi robot lainnya agar mendapatkan data lebih valid.
b. Implementasi pada Robot secara fisik, bukan hanya simulasi.
c. Selain CCD, *inverse kinematic* dapat diuji dengan kinematika lainnya seperti *triangle* dan *circular kinematics*.

DAFTAR PUSTAKA

Adept Technology, I. (December, 1997). *AdeptThree-XL Robot Instruction Handbook.*

Ahmed Joubair, E. P. (2014, October 7). *What are Accuracy and Repeatability in Industrial Robots?* Retrieved from Robotiq: https://blog.robotiq.com/bid/72766/What-are-Accuracy-and-Repeatability-in-Industrial-Robots

Amien, M., & Nurhayati, S. (2012). *Mahir Matematika SMA Cara Bimbel.* Surabaya: Lingua Kata.

Baharin, I., & Hasan, M. (2002). Identification of manipulator kinematics parameters through iterative method. 1-6. doi:10.1109/ICSMC.1994.400018

Baxter, B. (2001, February 21). *Fast Numerical Methods for Inverse Kinematics*. Retrieved from Fast Numerical Methods for Inverse Kinematics: http://billbaxter.com/courses/290/html/

Brumson, B. (2001, September 27). *Scara vs. Cartesian Robots: Selecting the Right Type for Your Applications*. Retrieved from Robotics Industries Association: https://www.robotics.org/content-detail.cfm/Industrial-Robotics-Industry-Insights/Scara-vs-Cartesian-Robots-Selecting-the-Right-Type-for-Your-Applications/content_id/1001

Corke, P. (2017). *Robotics Toolbox 10.2 for MATLAB.*

Juckett, R. (2009, February 11). *Cyclic Coordinate Descent in 2D*. Retrieved from http://www.ryanjuckett.com/programming/cyclic-coordinate-descent-in-2d/

Lander, J. (1998). *Making Kine More Flexible.* Retrieved from http://www.cs.cmu.edu/~15464-s13/lectures/lecture6/jlander_gamedev_nov98.pdf

Mathworks. (n.d.). *What is MATLAB?* Retrieved 2018, from https://www.mathworks.com/discovery/what-is-matlab.html

Rehiara, A. B. (2011). *Kinematics of AdeptThree Robot Arm.* (P. S. Goto, Ed.) InTech. doi:10.5772/17732

Rosalind. (diakses pada 08-01-2018). *Euclidean distance*. Retrieved from Rosalind: http://rosalind.info/glossary/euclidean-distance/

Setiawan, E. (2015). *DIKTAT KULIAH ROBOTIKA.*

Wang, L.-C. T., & Chen, C. C. (1991). A combined optimization method for solving the inverse kinematics problems of mechanical manipulators. *IEEE Transactions on Robotics and Automation, 7*(4), 489 - 499.

What Is MATLAB. (1997). Retrieved Maret 16, 2017, from http://cimss.ssec.wisc.edu/wxwise/class/aos340/spr00/whatismatlab.htm

Zhou, C. (n.d.). Accuracy. In C. Zhou. Georgia Tech. Retrieved July 02, 2018, from www.isye.gatech.edu/people/faculty/Chen_Zhou/Accuracy.PDF