**TUGAS AKHIR – TE 141599**

# NAVIGASI *MOBILE ROBOT* MENGGUNAKAN *DYNAMIC PATH PLANNING ALGORITHM* BERBASIS *GENETIC ALGORITHM*

Jeffry Susanto
NRP. 2213 100 010

Dosen Pembimbing
Dr. Trihastuti Agustinah, S.T., M.T.
Ir. Rusdhianto Effendi A.K., M.T.

DEPARTEMEN TEKNIK ELEKTRO
Fakultas Teknologi Elektro
Institut Teknologi Sepuluh Nopember
Surabaya 2016

*FINAL PROJECT* – **TE 141599**

# MOBILE ROBOT NAVIGATION USING DYNAMIC PATH PLANNING ALGORITHM BASED ON GENETIC ALGORITHM

Jeffry Susanto
NRP. 2213 100 024

Supervisor
Dr. Trihastuti Agustinah, S.T., M.T.
Ir. Rusdhianto Effendi A.K., M.T.

*DEPARTEMENT OF ELECTRICAL ENGINEERING*
*Faculty of Electrical Technology*
Institut Teknologi Sepuluh Nopember
Surabaya 2016

# PERNYATAAN KEASLIAN TUGAS AKHIR

Dengan ini saya menyatakan bahwa isi sebagian maupun keseluruhan tugas akhir saya dengan judul "**Navigasi *Mobile Robot* Menggunakan *Dynamic Path Planning Algorithm* berbasis *Genetic Algorithm***" adalah benar-benar hasil karya intelektual mandiri, diselesaikan tanpa menggunakan bahan-bahan yang tidak diijinkan dan bukan merupakan karya pihak lain yang saya akui sebagai karya sendiri.

Semua referensi yang dikutip maupun dirujuk telah ditulis secara lengkap pada daftar pustaka.

Apabila ternyata pernyataan ini tidak benar, saya bersedia menerima sanksi sesuai peraturan yang berlaku.

Surabaya, Juni 2017

Jeffry Susanto
NRP. 2213 100 010

*Halaman ini sengaja dikosongkan*

**NAVIGASI *MOBILE ROBOT* MENGGUNAKAN *DYNAMIC PATH PLANNING PATH ALGORITHM* BERBASIS *GENETIC ALGORITHM***

**TUGAS AKHIR**

Diajukan Guna Memenuhi Sebagian Persyaratan Untuk
Memperoleh Gelar Sarjana Teknik Elektro
Pada
Bidang Studi Teknik Sistem Pengaturan
Departemen Teknik Elektro
Fakultas Teknologi Elektro
Institut Teknologi Sepuluh Nopember

**LEMBAR PENGESAHAN**

Menyetujui

| Dosen Pembimbing I, | Dosen Pembimbing II, |
|---|---|
| Dr. Trihastuti Agustinah, S.T., M.T. | Ir. Ruschiyanto Effendi A.K., M.T. |
| NIP. 196808121994032001 | NIP. 195412271981031002 |

**SURABAYA**
**JULI, 2017**



vii

*Halaman ini sengaja dikosongkan*

**NAVIGASI *MOBILE ROBOT* MENGGUNAKAN *DYNAMIC PATH PLANNING ALGORITHM* BERBASIS *GENETIC ALGORITHM***


Nama : Jeffry Susanto
Dosen Pembimbing : Dr. Trihastuti Agustinah, S.T., M.T.
196808121994032001
: Ir. Rusdhianto Effendi A.K., M.T.
195412271981031002


# ABSTRAK


Kemampuan navigasi dalam sebuah *mobile robot* mengacu pada kemampuan robot tersebut untuk mengetahui posisinya sendiri dalam *frame reference*nya dan kemudian merencanakan sebuah *path* yang *feasible* menuju sebuah lokasi tujuan. Dalam tugas akhir ini, akan dibahas bagaimana *Genetic Algorithm* akan digunakan untuk merencanakan *path* yang akan dilalui *mobile robot* untuk mencapai sebuah tujuan di sebuah lingkungan yang dinamis. *Mobile robot* yang digunakan adalah sebuah *differential drive robot*. Untuk keperluan validasi algoritma akan dilakukan simulasi dan implementasi. Untuk simulasi, akan digunakan *software* MATLAB. Sedangkan untuk implementasi akan digunakan sebuah Qbot yang diproduksi oleh Quanser. Algoritma yang diterapkan dalam penelitian ini bisa menghasilkan panjang terpendek dalam sebuah *map* 20x20 dengan *obstacle* statis dan dinamis. Dalam lingkungan statis, *path* terpendek yang ditemukan dalam 20,4914 detik adalah 28,5779 satuan panjang. Untuk lingkungan dinamis, proses *re-planning* yang dilakukan dalam 2,7062 detik mampu menemukan sebuah *path* baru yang tidak menabrak *obstacle*.



**Kata Kunci:** *Path Planning, Differential Drive Robot,* Qbot, Algoritma Genetika, lingkungan statis dan dinamis.

*Halaman ini sengaja dikosongkan*

***MOBILE ROBOT NAVIGATION USING DYNAMIC PATH PLANNING ALGORITHM BASED ON GENETIC ALGORITHM***

*Name* *:* Jeffry Susanto
*Supervisor* : Dr. Trihastuti Agustinah, S.T., M.T.
196808121994032001
: Ir. Rusdhianto Effendi A.K., M.T.
195412271981031002

## *ABSTRACT*

*A mobile robot's ability to navigate usually refers to its capability to know its position in its own frame reference and then plan a feasible path to a target location. In this final project, genetic algorithm will be used to plan a path which will be followed by the mobile robot in order to travel through an environment with dynamic obstacles. The mobile robot which will be used is a differential drive robot. For validation purposes, a set of simulations and implementations will be conducted. For the simulations, the software MATLAB, will be used. While for the implementation, a Qbot, a mobile robot produced by Quanser, will be used. The algorithm used in this research is capable of finding a shortest path in a 20 by 20 map with static and dynamic obstacles. In a static environment, the shortest path which is found in 20.4914 s, is 28.5779 unit length. In a dynamic environment, the re-planning process which is done in 2.7062 s can find a new path which will not collide any obstacle.*

***Keywords:*** *Path Planning, Differential Drive Robot, Qbot, Genetic Algorithm, dynamic and static environment.*

*Halaman ini sengaja dikosongkan*

# KATA PENGANTAR

Puji syukur penulis unjukkan kehadirat Tuhan yang Maha Esa atas berkatNya sehingga penulis dapat menyelesaikan tugas akhir ini yang berjudul **"Navigasi *Mobile Robot* menggunakan *Dynamic Path Planning Algorithm* berbasis *Genetic Algorithm*"** guna memenuhi syarat kelulusan pada Bidang Studi Teknik Sistem Pengaturan Departemen Teknik Elektro Fakulitas Teknologi Elektro Institut Teknologi Sepuluh Nopember (ITS) Surabaya. Penulis juga mengucapkan terimakasih kepada pihak-pihak yang berjasa dalam pengerjaan tugas akhir ini yaitu:

1. Keluarga terutama orang tua, dan anggota keluarga lain yang selalu memberi dukungan, semangat, dan doa untuk keberhasilan penulis.
2. Ibu Dr. Trihastuti Agustinah, S.T., M.T. dan Bapak Ir. Rusdhianto Effendie A.K., M.T. selaku dosen pembimbing yang selalu menyediakan waktu untuk membimbing.
3. Dosen-dosen lain yang telah mengajar penulis selama penulis belajar di ITS.

Laporan Tugas Akhir ini masih jauh dari sempurna. Oleh karena itu, penulis mengharapkan kritik dan saran dari pembaca. Semoga buku laporan Tugas Akhir ini dapat memberikan manfaat bagi pembaca sebagai acuan penelitian selanjutnya.

Surabaya, 4 Juni 2017

Penulis

*Halaman ini sengaja dikosongkan*

# DAFTAR ISI

# DAFTAR GAMBAR

# DAFTAR TABEL

*Halaman ini sengaja dikosongkan*

# BAB 1
# PENDAHULUAN

Pada bab satu akan dibahas mengenai latar belakang, permasalahan, tujuan, metodologi, sistematika, dan relevansi tugas akhir yang dikerjakan.

## 1.1 Latar Belakang

Terdapat banyak tantangan yang mungkin ditemui dalam navigasi *mobile robot* dalam lingkungan yang statis maupun dinamis. Lingkungan yang statis adalah lingkungan dengan rintangan (*obstacle*) yang tidak bergerak, sedangkan lingkungan dinamis adalah lingkungan dengan rintangan (*obstacle*) yang bergerak. Salah satu tantangan yang mungkin ditemui adalah bagaimana mengatur gerak robot dari satu titik ke titik lain tanpa menabrak rintangan. Permasalahan tersebut diharapkan bisa diterapkan di dunia nyata dimana robot dijalankan di sebuah lingkungan urban (*urban environment*). [1]

Telah banyak penelitian yang telah dilakukan untuk menyelesaikan baik permasalahan *path planning* maupun navigasi menggunakan *artificial intelligence*, seperti *Fuzzy Logic*, *Artificial Neural Networks* [2], *Genetic Algorithm,* maupun gabungan dari beberapa metode seperti. Kebanyakan algoritma *path planning* diatas memiliki keterbatasan dimana terkadang terdapat *infeasible path*, hanya mampu menangani *static obstacle*, dan lain-lain yang mampu mempengaruhi waktu komputasi yang akan mempengaruhi kecepatan gerak robot.

Dalam tugas akhir ini, *genetic algorithm* digunakan karena ia memiliki kemampuan optimasi yang baik. *Genetic Algorithm* sendiri merupakan metode yang digunakan untuk menyelesaikan baik *constrained* maupun *unconstrained optimization problems* yang didasarkan pada proses seleksi alam yang terjadi dalam proses evolusi biologi. Algoritma ini bisa diterapkan untuk menyelesaikan permasalahan yang tidak bisa diseleksaikan dengan algoritma optimasi biasa, termasuk *objective function* yang *discontinuous, nondifferentiable,* stokastik, atau sangat *non-linear*.

## 1.2 Perumusan Masalah

Pada tugas akhir ini, masalah yang dibahas adalah perencanaan *path* gerak sebuah *mobile* robot sehingga mampu mencapai tujuan dengan lintasan terpendek melewati *obstacle* statis dan dinamis. Kemudian, algoritma yang digunakan untuk menyelesaikan masalah tersebut akan divalidasi menggunakan simulasi dan implementasi.

## 1.3 Batasan Masalah

Terdapat beberapa batasan masalah dalam tugas akhir ini antara lain:

1. Terdapat beberapa parameter yang harus diketahui terlebih dahulu sebelum menjalankan program, seperti: ukuran populasi, ukuran kromosom yang tetap, ukuran dan posisi *obstacle*, nilai Pc dan Pm, dll
2. Digunakan sebuah map dengan ukuran 20x20, dimana titik yang mungkin menjadi kandidat *waypoint* hanya titik dengan koordinat bilangan bulat.

## 1.4 Tujuan Penelitian

Tujuan dari pelaksanaan tugas akhir ini adalah menghasilkan:

1. Lintasan (*path*) terpendek dari satu titik ke titik lain tanpa menabrak rintangan statis maupun dinamis.
2. Validasi algoritma menggunakan simulasi dan implementasi.

## 1.5 Metodologi

Dalam penelitian tugas akhir ini diperlukan suatu tahapan yang merepresentasikan urutan yang harus dilaksanakan agar sesuai dengan tujuan penelitian. Tahapan tersebut ialah sebagai berikut:

1. Studi Literatur
   Studi literatur dilakukan dengan mencari dan membaca referensi yang berasal dari sumber ilmiah terpercaya, seperti buku materi, jurnal ilmiah, artikel ilmiah, pendapat para ahli, dan hasil penelitian terkait.
2. Identifikasi
   Identifikasi sistem perlu dilakukan untuk memperoleh model matematika plant terkait, serta mencari parameter-parameter

lain yang akan diperlukan dalam perancangan sistem, simulasi dan implementasi.

3. Simulasi
   Algoritma tersebut kemudian disimulasikan menggunakan software MATLAB sehingga bisa dilakukan berbagai analisa performansi sistem.
4. Implementasi
   Tahap implementasi adalah tahap dimana kontroler yang telah dirancang diterapkan pada *robot* sesudah dipastikan bahwa simulasi berhasil. Implementasi dilakukan agar bisa diperoleh data performansi dalam kondisi riil.
5. Penulisan Buku Tugas Akhir
   Tahap yang terakhir ialah penulisan laporan/buku Tugas Akhir. Penulisan dilakukan secara intensif bila proses pengujian telah selesai.

## 1.6 Sistematika Penelitian

Pembahasan Tugas Akhir ini dibagi menjadi lima bab dengan sistematika sebagai berikut :

**BAB I : Pendahuluan**
Bab ini meliputi latar belakang, permasalahan, tujuan penelitian, dan sistematika penulisan.

**BAB II : Teori Penunjang**
Bab ini membahas tinjauan pustaka yang membantu penelitian, diantaranya teori *path planning,* algoritma genetika, *differential drive robot,* persamaan lingkaran, *pure pursuit controller*, dan *polar histogram*.

**BAB III : Perancangaan Sistem**
Bab ini dijelaskan mengenai identifikasi masalah, perancangan algoritma *path planning,* perancangan simulasi, dan perancangan implementasi menggunakan *mobile robot* Qbot.

**BAB IV : Hasil dan Analisa**
Bab ini memuat hasil dan analisa simulasi *path planning* dalam lingkungan statis dan dinamis, dan

implementasi menggunakan Qbot dalam lingkungan statis dan dinamis.

**BAB V : Penutup**

Bab ini berisi kesimpulan dan saran dari hasil penelitian yang telah dilakukan.

## 1.7 Relevansi

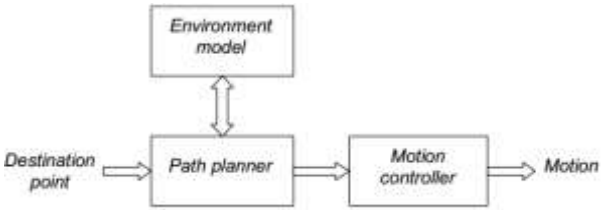
Hasil yang diperoleh dari Tugas Akhir ini diharapkan menjadi referensi pengembangan teknologi *path planning* berbasis algoritma genetika yang tepat untuk pada *mobile robot* di masa depan.

# BAB 2
# TINJAUAN PUSTAKA

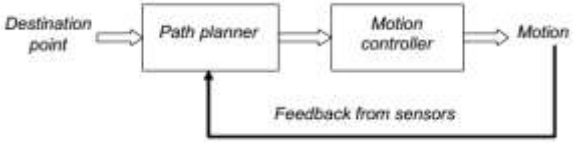
Pada bab ini dibahas mengenai teori dasar yang berkaitan dengan penelitian yang dilakukan yang meliputi: *path planning,* algoritma genetika, *differential drive robot*, persamaan lingkaran, *pure pursuit controller*, dan *polar histogram*.

## 2.1 *Path Planning*

Pada permasalahan navigasi *mobile robot*, *path planning* merupakan salah satu permasalahan yang paling mendasar. *Path planning* membuat *mobile robot* mampu menemukan *path* yang terpendek atau optimal antara dua titik. *Path* yang optimal sendiri belum tentu merupakan *path* yang terpendek, melainkan sesuai kriteria seperti: meminimalkan jumlah *turning*, *braking*, dan lain-lain.



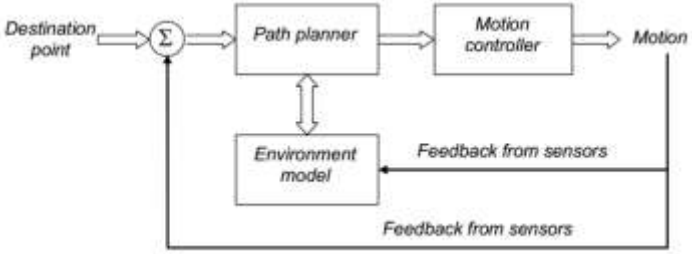

Gambar 2. 1 *Global Path Planning [3]*




Gambar 2. 2 *Local Path Planning [3]*

*Path planning* dapat dikategorikan menjadi dua: *global path planning (offline)* (Gambar 2.1) dan *local path planning (online)* (Gambar 2.2). Untuk menyelesaikan permasalahan *global path planning,* diperlukan informasi berupa *map* yang berisikan informasi mengenai ukuran dan posisi *obstacle.* Akan tetapi, *global path planning* hanya mampu menyelesaikan permasalahan *path planning* dalam lingkungan statis oleh karena tidak ada sensor yang digunakan. Dalam permasalahan

*local path planning,* digunakan hasil pembacaan sensor untuk menentukan gerak dari *mobile robot*. *Dynamic path planning* (Gambar 2.3) sendiri adalah perpaduan dari *global path planning* dan *local path planning* dimana *environment model* akan di-*update* jika terdeteksi *obstacle* lain yang sebelumnya tidak terdeteksi. Selain itu, diperlukan juga kemampuan robot untuk mengetahui posisinya saat ini relatif dengan *environment* agar bisa menjadi salah satu pertimbangan dalam melakukan *path planning.* [3]




Gambar 2. 3 *Dynamic Path Planning Algorithm [3]*

*Dynamic Path Planning Algorithm* yang berbasis *Genetic Algorithm* diharapkan mampu mencapai tujuan tugas akhir ini dalam mencapai tujuan. *Dynamic Path Planning Algorithm* sendiri terdiri atas beberapa subprogram, antara lain : *Goal Heading, Remove Redundant Point, Compute Shortest Path, Path Optimization, Detect Obstacle,* dan sebagainya. Masing-masing subprogram tersebut mempunyai fungsi masing-masing yang jika digabungkan akan menyelesaikan permasalahan *path planning*. Konsep-konsep pada *genetic algorithm* tersebut akan diterapkan dalam penyusunan *dynamic path planning algorithm,* sehingga akan dihasilkan sebuah *path* optimal yang bisa dilalui sebuah *mobile robot* tanpa menabrak *obstacle* baik statis maupun dinamis.

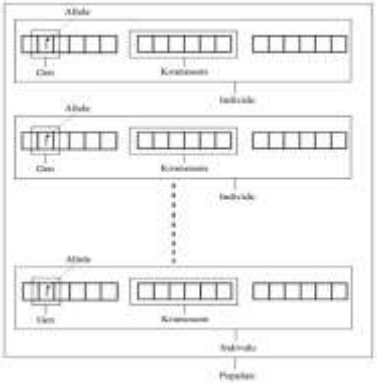
## 2.2 Algoritma Genetika

Genetic algorithm atau algoritma genetika (GA) masuk dalam kelompok *Evolutionary Elgorithm*. *Genetic algorithm* didasarkan pada prinsip-prinsip genetika dan seleksi alam. Elemen-elemen dasar dari genetika alam adalah reproduksi, kawin silang, dan mutasi. Kemudian diadopsi menjadi algoritma komputasi untuk mencari solusi suatu permasalahan optimasi. GA pertama kali dicetuskan oleh John Holland,

Profesor University of Michigan pada tahun 1970 dan dikembangkan oleh Goldberg. [4]

GA yang dikembangkan oleh Goldberg menyatakan bahwa kelangsungan hidup suatu makhluk dipengaruhi aturan "*survival of the fittest*". Darwin juga menyatakan bahwa kelangsungan tersebut dapat dipertahankan melalui proses reproduksi, *crossover*, dan mutasi dan kemudian diadopsi menjadi algoritma komputasi untuk mencari solusi suatu permasalahan dengan mengambil inspirasi dari proses alami tersebut.

Dalam Genetic algorithm kromosom adalah satu hal yang penting, satu kromosom atau individu mewakili satu vektor solusi yang akan diproses dengan prosedur-prosedur algoritma genetika. Terkadang representasi dari kromosom dapat ditulis dengan nilai kontinyu, dalam hal lain juga dapat ditulis dengan angka biner sehingga harus melalui *encoding* atau pengkodean lalu melakukan *decoding* kembali. Hal ini tergantung dengan permasalahan yang dihadapi. Dalam permasalahn optimasi fungsi sering kali menggunakan angka biner, karena dengan angka biner proses kawin silang dan mutasi akan lebih banyak variasinya. Dalam *genetic algorithm* akan dibangkitkan sejumlah kromosom atau individu yang disebut dengan populasi sehingga dengan dibangkitkannya populasi ini, maka akan tersedia banyak pilihan solusi. Populasi yang terjadi pertama kali biasa disebut dengan populasi awal atau inisialisasi populasi, seperti pada Gambar 2.4 [5]



Gambar 2. 4 Susunan Algoritma Genetika *[4]*

Dalam Genetic algorithm dikenal suatu fungsi yang disebut fungsi *fitness*. Dimana fungsi *fitness* ini digunakan untuk mengukur tingkat kebaikan atau kesesuaian suatu solusi yang dicari. Fungsi fitness biasanya berhubungan langsung dengan fungsi tujuan. Dalam kondisi tertentu juga bisa dimodifikasi dari fungsi tujuannya, hal ini menyesuaikan dengan tujuan yang diinginkan. [6]

*Etilisme* merupakan usaha untuk mempertahankan individu atau kromosom terbaik dari populasi yang ada. Sehingga individu terbaik ini tidak terseleksi maupun mengalami kawin silang saat proses genetika selanjutnya. Sehingga individu terbaik ini dapat bertahan di generasi selanjutnya dan sangat mempengaruhi hasil dari *genetic algorithm.* Sebelum dilakukannya kawin silang terlebih dahulu dilakukan proses seleksi orang tua yang akan dikawinkan. Seleksi ini dapat dilakukan dengan beberapa metode antara lain, roda lotere, turnamen, dan *stochastic universal selection*. [6]

Setelah terpilih dua individu yang terpilih menjadi orang tua setelah itu akan dilakukan proses kawin silang yang akan menghasilkan individu baru untuk semua populasi, kecuali individu terbaik. Kromosom baru yang disebut dengan *off spring*. Jumlah kromosom dalam populasi yang mengalami *crossover* ditetukan oleh paramater yang disebut dengan *crossover rate*. Gambar dibawah merupakan ilustrasi dari *crossover* dengan menggunakan metode aritmatik yang sebagian banyak sesuai dengan bilangan kontinyu.

Mekanisme perubahan susunan genetika akibat adanya faktor alam disebut dengan mutasi. Jumlah gen dalam populasi yang mengalami mutasi ditentukan oleh parameter yang dinamakan *mutation rate*. Setelah beberapa generasi akan dihasilkan kromosom yang nilainya konvergen pada suatu nilai, di mana merupakan solusi terbaik yang dihasilkan oleh GA terhadap permasalahan yang ingin diselesaikan.

Sebuah kromosom dibentuk dari komponen-komponen penyusun yang disebut sebagai gen dan nilainya dapat berupa bilangan numerik, biner, simbol, ataupun karakter. Sebagian besar kromosom memiliki nilai yang berupa nilai biner atau kontinyu. Penggunaan bilangan tersebut bergantung pada jenis permasalahan yang dihadapi. Penggunaan jenis bilangan pada kromosom tersebut juga menentukan metode yang sesuai

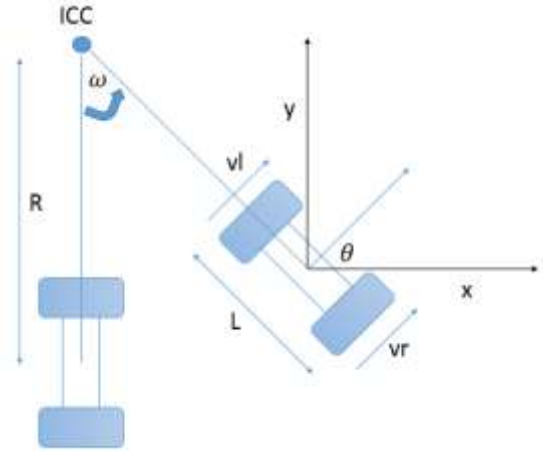
untuk proses seleksi dan *crossover*. Kromosom akan berevolusi secara berkelanjutan yang disebut dengan generasi. Dalam tiap generasi kromosom tersebut dievaluasi tingkat keberhasilan nilai solusinya terhadap masalah yang ingin diselesaikan (fungsi objektif) menggunakan ukuran yang disebut dengan *fitness*. Individu yang memiliki nilai *fitness* tinggi akan memiliki peluang lebih besar untuk terpilih lagi pada generasi selanjutnya. Diagram alir GA dapat dilihat pada Gambar 2.5.


Gambar 2. 5 Diagram Alir Proses Algoritma Genetika *[4]*

## 2.3 Differential Drive Robot

Sebuah *differential drive robot* memiliki dua roda yang bisa bergerak secara independen. Perbedaan atau kesamaan kecepatan kedua roda menentukan apakah robot tersebut akan bergerak rotasi atau lurus.

Ilustrasi mengenai *differential drive robot* dan parameternya tercantum pada Gambar 2.6. [7]




*Gambar 2. 6 Differential Drive Robot*

Dimana L adalah jarak dari pusat antara kedua roda, $vl$ dan $vr$ adalah kecepatan roda (translasi), R merupakan jarak antara titik tengah L (L/2 jika diukur dari salah satu roda) dan *Instaneous Center of Curvature* (ICC) yang merupakan titik dimana robot berotasi. [7]

Jika mengikuti konfigurasi diatas, maka ICC dapat dituliskan sebagai:

$$ICC = [x - Rsin(\theta)\,, y + Rcos(\theta)] \tag{2.1}$$

Saat $vl \neq vr$, *trajectory* robot juga akan berubah (dengan kata lain, melakukan perubahan arah gerak). Oleh karena perubahan rotasi $\omega$ di ICC harus sama antara kedua roda, Persamaan (2.2) dan (2.3) berlaku [7]

$$\omega\left(R + \frac{L}{2}\right) = vr \tag{2.2}$$

$$\omega\left(R - \frac{L}{2}\right) = vl \tag{2.3}$$

Jika Persamaan (2.2) dan (2.3) digabungkan dengan persamaan $v = \omega R$, [7]

$$R = \frac{L}{2}\frac{vr + vl}{vr - vl} \tag{2.4}$$

$$\omega = \frac{vr - vl}{l} \tag{2.5}$$

Dari persamaan-persamaan diatas dapat ditarik beberapa hal, antara lain:

1. Jika $vl = vr$, maka akan diperoleh gerak lurus linear ke depan. $R$ bernilai tak terhingga dan tidak ada gerak rotasi yang mengakibatkan $\omega$ bernilai 0
2. Jika $vl = -vr$, maka $R$ bernilai 0, dan akan diperoleh rotasi pada titik $L/2$ atau berputar di tempat.
3. Jika $vl = 0$, maka akan diperoleh $R = L/2$ dan gerak rotasi dengan poros pada roda kiri. Dan sebaliknya untuk roda kanan. [7]

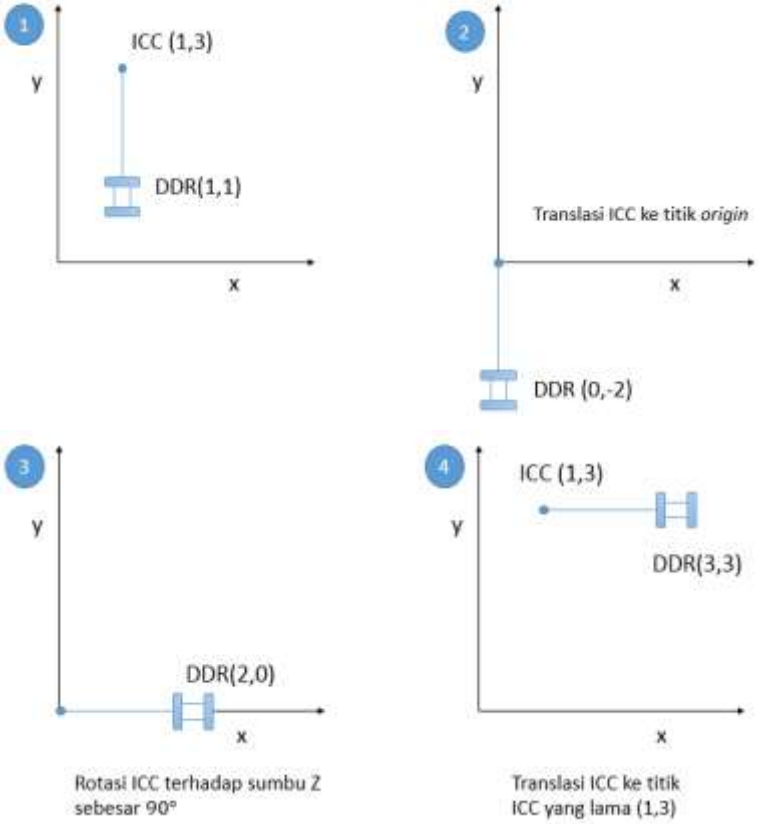
*Differential drive robot* tidak bisa bergerak ke arah sumbu kedua roda atau dengan kata lain, memiliki *non-holonomic constraint*. Selain itu, biasanya *differential drive robot* memiliki sebuah roda tambahan yang pasif sebagai *support wheel* agar lebih stabil terhadap variasi yang mungkin ada di lantai.

### 2.3.1 *Forward Kinematics*

Pada Gambar 2.6, terdapat sebuah robot pada posisi sebarang $(x, y)$, bergerak menuju sebuah titik dengan arah $\theta$ terhadap sumbu $x$. Dengan mengubah nilai variabel kontrol $vl$ dan $vr$, posisi dan orientasi robot bisa diubah. Mengacu pada Persamaan (2.1), maka posisi suatu robot pada $t + \delta t$ adalah : [7]

$$\begin{bmatrix} x' \\ y' \\ \theta' \end{bmatrix} = \begin{bmatrix} \cos(\omega\delta t) & -\sin(\omega\delta t) & 0 \\ \sin(\omega\delta t) & \cos(\omega\delta t) & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x - ICC_x \\ y - ICC_y \\ \theta \end{bmatrix} + \begin{bmatrix} ICC_x \\ ICC_y \\ \omega\delta t \end{bmatrix} \tag{2.6}$$

Persamaan (2.6) diperoleh dari transformasi linear dari vektor posisi robot asal $[x\ y\ \theta]'$ ke vektor posisi robot baru $[x'\ y'\ \theta']'$. Pada Gambar 2.7, diilustrasikan contoh kasus yang bertujuan untuk menunjukkan asal dari Persamaan (2.6) (bagaimana posisi robot jika melakukan gerak rotasi $90^0$). Transformasi linear yang dilakukan adalah: translasi ICC ke titik *origin*, rotasi ICC 90° terhadap sumbu Z, dan translasi ICC ke titik ICC semula.




Gambar 2. 7 Contoh Gerak *Forward Kinematics*

### 2.3.2 *Inverse Kinematics* [7]

Persamaan (2.7), (2.8), dan (2.9) menjelaskan posisi *differential drive robot* bergerak ke arah $\theta$ jika diketahui informasi mengenai $v(t)$.

$$x(t) = \frac{1}{2}\int_0^t \big(vr(t) + vl(t)\big)\cos[\theta(t)]\,dt \tag{2.7}$$

$$y(t) = \frac{1}{2}\int_0^t \big(vr(t) + vl(t)\big)\sin[\theta(t)]\,dt \tag{2.8}$$

$$\theta(t) = \frac{1}{L}\int_0^t \big(vr(t) - vl(t)\big)dt \tag{2.9}$$

Jika diasumsikan bahwa $vl(t) = vl$ ; $vr(t) = vr$ ; $vl \neq vr$, kemudian Persamaan (2.7), (2.8), dan (2.9) disubstitusikan dengan Persamaan (2.4) dan (2.5) agar diperoleh nilai posisi dalam $vl\ dan\ vr$, maka Persamaan (2.7), (2.8), dan (2.9) dapat dituliskan sebagai:

$$x(t) = R\sin(\omega t) = \frac{L}{2}\frac{vr + vl}{vr - vl}\sin(\frac{t}{L}(vr - vl)) \tag{2.10}$$

$$y(t) = -R\cos(\omega t) = -\frac{L}{2}\frac{vr + vl}{vr - vl}\cos(\frac{t}{L}(vr - vl)) \tag{2.11}$$

$$\theta(t) = \omega = \frac{t}{L}(vr - vl) \tag{2.12}$$

### 2.3.3 Mobile Robot Qbot [8]

Qbot *Mobile robot* merupakan sistem inovatif robot darat *autonomous* yang menggabungkan ilmu kendaraan darat dengan *Quanser Controller Module* (QCM). *Mobile robot* ini terdiri dari iRobot Create® *Robotic Platform*, sensor inframerah dan sonar serta sebuah kamera *Logitech Quickcam Pro* 9000 USB seperti terlihat pada Gambar 2.8. QCM yang terpasang pada Qbot yang mana menggunakan komputer Gumstix untuk menjalankan QuaRC, perangkat lunak kontrol Quanser, dan Qbot kartu data akuisisi (DAC).

*Gambar 2. 8 Mobile Robot Qbot*

Berikut ini spesifikasi dan model parameter *mobile robot* Qbot dapat dilihat pada Tabel 2.1.

Tabel 2. 1 Spesifikasi Model *Mobile Robot* Qbot

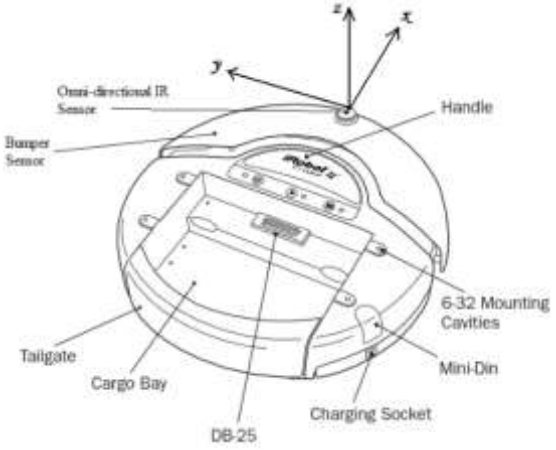
| **Simbol** | **Deskripsi** | ***Value*** | ***Unit*** |
|---|---|---|---|
| *D* | Diameter Qbot *mobile robot* | 0,34 | *m* |
| *H* | Tinggi Qbot *mobile robot* (dengan tambahan kamera) | 0,19 | *m* |
| *Vmax* | Kecepatan maksimum dari Qbot *mobile robot* | 0,5 | *m/s* |
| *M* | Berat total Qbot *mobile robot* | 2,92 | *kg* |

*Interface* untuk QCM adalah MATLAB Simulink dengan QuaRC. Qbot dapat menerima perintah, melalui tiga blok yang berbeda, yaitu blok Roomba untuk menggerakkan Qbot, blok HIL untuk membaca dari sensor dan/atau menulis untuk O*utput* servo dan blok OpenCV untuk mengakses kamera. Kontroler dikembangkan pada Simulink dengan QuaRC dan model ini diunduh dan disusun pada target (Gumstix). Diagram dari konfigurasi ini ditampilkan dalam Gambar 2.9.


Gambar 2. 9 Hierarki Komunikasi Qbot

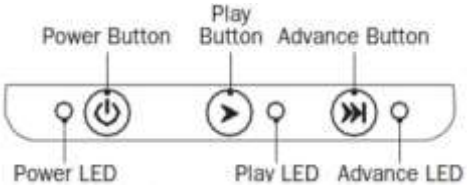
### 2.3.3.1 *Sistem Perangkat Keras Qbot*

*Mobile robot* Qbot menggunakan iRobot Create® seperti pada Gambar 2.10. Qbot mengikuti standar Quanser untuk sumbu rangka tubuh, di mana sumbu x pada arah maju, sumbu y sebelah kiri dan sumbu z ada di atas. Diameter dari rangka ini 34 cm dan tinggi (tanpa tambahan kamera) adalah 7 cm. dan Qbot digerakkan oleh dua roda kemudi yang berbeda. iRobot Create® terdapat bumper sensor dan inframerah. QCN dapat mengakses data dari sensor ini.




Gambar 2. 10 Rangka Mobile Robot Qbot

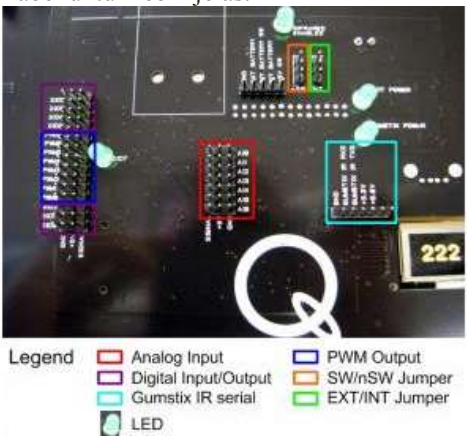
*Mobile robot* Qbot dihidupkan dengan menekan tombol *Power*. Gambar 2.10 menunjukkan tombol yang digunakan untuk mengoperasikan Qbot. Tombol *Play* dan *Advance* untuk menjalankan dalam demo untuk Qbot dan tidak diperlukan untuk tujuan *mobile robot* Qbot.



Gambar 2. 11 Tombol Pada Rangka Qbot

### 2.3.3.2 *Printed Circuit Board (PCB)*

Kabel dan sirkuit untuk Qbot dalam PCB yang terletak dipenutup hitam Qbot atas komputer Gumstix dan DAC. Sensor dan kamera yang juga terpasang pada PCB. Gambar 2.11 menunjukkan pin diakses bagi pengguna. Secara khusus, DIO, *Output* PWM, dan pin *input* analog telah diberi label untuk lebih jelas.



Gambar 2. 12 PCB Mobile Robot Qbot

### 2.3.3.3 *Pin Digital Input/Output (DIO)*

Saluran DIO (0 sampai 6) ditetapkan sebagai *input* secara bawaan. Saluran DIO perlu dikonfigurasi baik sebagai *input* (atau *ouput,*

tapi tidak keduanya) menggunakan blok HIL *Initialize*. Jika *Output* harus dalam keadaan yang dikenal *power up,* dianjurkan bahwa resistor 10K Ω diletakkan dari I/O untuk 5V atau GND sesuai kebutuhan. Ada saluran digital akhir (7) yang merupakan O*utput* tetap, dan itu diwakili oleh LED berlabel DIO7.

#### 2.3.3.4 *Pin Serial Gumstix IR*

Qbot menyediakan *serial* koneksi TTL ke *port* serial Gumstix IR (*port* no 2). *Port serial* terdiri dari *ground* (GND), menerima Gumstix IR RXD, mengirimkan Gumstix IR TXD dan pin daya (+3.3V atau +5.0V). *Port serial* diakses melalui blok QuaRC *Stream* atau *Stream* API.

#### 2.3.3.5 *SW/nSW dan INT/EXT Jumpers*

INT/EXT *jumper switch* Qbot dari internal daya dari baterai iRobot Create (INT) dan *eksterna*l baterai *power supply* (EXT). Tidak ada baterai eksternal yang disertakan pada Qbot, jadi *jumper* ini harus dibiarkan dalam posisi INT untuk daya Qbot tersebut. Saat *jumper power supply* dalam posisi INT, *jumper* SW/ NSW menunjukkan apakah iRobot Create harus diaktifkan (SW) untuk Qbot menerima daya atau apakah Qbot harus selalu membutuhkan daya bahkan ketika iRobot Create dalam keadaan mati (nSW).

#### 2.3.3.6 *Qbot Data Acquisition Card (DAC)*

Qbot DAC adalah kartu data akuisisi, yang mampu menerima *input analog* dan *input* lainnya (untuk sensor sonar). Qbot DAC dapat juga membaca *Output* PWM untuk servo aktuatuor. Qbot DAC terletak di bawah penutup hitam Qbot.

#### 2.3.3.7 *Gumstix*

Gumstix berfungsi penuh pada komputer *open source* dimana Simulink MATLAB secara langsung dapat diunduh, dikompilasi dan dijalankan melalui perangkat lunak yaitu QuaRC. *Motherboard* Gumstix terhubung secara langsung pada Qbot DAC. Pada Gumstix juga terdapat tambahan *Wifi* untuk menyediakan koneksi nirkabel antara target Gumstix dan komputer *host*.

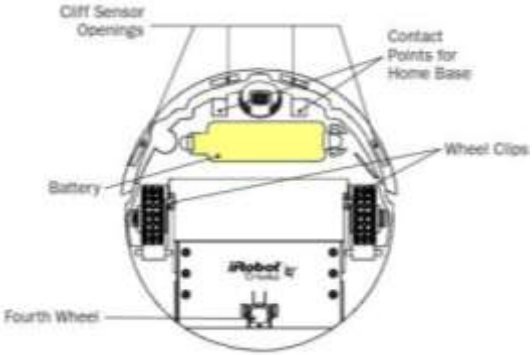
#### 2.3.3.8 *Baterai Mobile Robot Qbot*

Qbot ini didukung oleh baterai *Advance Power System* (APS) yang disediakan oleh iRobot seperti pada Gambar 2.13. Baterai ini

terletak di bawah Qbot dan dapat berlangsung terus menerus selama sekitar dua jam setelah terisi penuh.

Gambar 2. 13 Baterai Qbot

Baterai terletak dibagian bawah seperti pada Gambar 2.14. Lampu daya Qbot mengindikasikan tingkat daya baterai. Lampu hijau menandakan baterai masih terisi penuh, kemudian secara bertahap berubah menjadi merah bila baterai akan habis. Baterai memerlukan waktu kurang dari tiga jam untuk pengisian baterai. Ketika pengisian, lampu daya akan berkedip perlahan dengan warna jingga dan akan berubah menjadi hijau ketika terisi penuh.




Gambar 2. 14 Letak Baterai pada Qbot

### 2.3.3.9 *Sensor Inframerah dan Sonar*

SHARP 2Y0A02 seperti pada Gambar 2.15 merupakan sensor inframerah dengan jarak 20-150 cm. Terdapat lima buah sensor inframerah yang terpasang pada Qbot. Sensor terhubung kesaluran *input analog* dari Qbot DAC, yang kemudian dapat dibaca dengan menggunakn blok HIL *Read Write.*

Gambar 2. 15 Sensor Inframerah SHARP 2Y0A02

Sensor sonar MaxSonar-EZ0 mendeteksi objek dari 0 inci sampai 254 inci dengan resolusi 1 inci. Objek antara 0 inci dan 6 inci berkisar sebagai 6 inci. Terdapat tiga sensor pada Qbot. Sensor yang terhubung ke saluran masukan lain dari Qbot DAC, yang kemudian dapat dibaca dengan menggunakan blok HIL *Read Write*. Gambar 2.16 menunjukkan sensor sonar MaxSonar-EZ0.

Gambar 2. 16 Sensor Sonar MaxSonar-EZ0

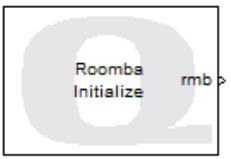
**2.3.3.10 *QUARC***

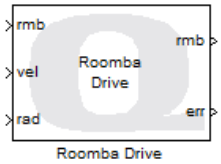
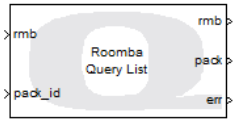
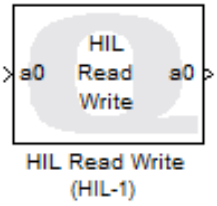
QUARC (*Quanser Real-Time Control*) adalah perangkat lunak yang menyediakan fasilitas untuk mengembangkan dan menguji coba rancangan kontroler pada komputer *host* dengan menggunakan perangkat lunak Simulink MATLAB, dengan model yang dirancang dapat langsung diunduh dan dieksekusi pada Gumstix dengan menggunakan komunikasi *wireless* secara *real time*. Pada waktu yang bersamaan, operator dapat memantau dan mengukur nilai-nilai dari sensor dan mengatur parameter-parameter kontroler langsung dari komputer *host*. Pada MATLAB Simulink terdapat beberapa Blok yang digunakan untuk melakukan komunikasi dengan *mobile robot* Qbot seperti yang terdapat pada Tabel 2.2 dan Tabel 2.3.

Tabel 2. 2 Deskripsi Blok Set untuk *Mobile Robot* Qbot

| ***Blok Set*** | **Deskripsi** |
|---|---|
| ***Interface*** | Blok set ini mengimplementasikan dasar *Aplication Program Interfaces* (API) yang disediakan oleh *iRobot Create*®. API dapat dikategorikan berdasarkan fungsional berikut: Sambungkan konfigurasi serial antara kontroler tingkat tinggi (misalnya, PC atau Gumstix) dan Qbot. Pengaturan *mode* operasi Qbot, mengakses informasi sensorik Qbot dan konfigurasi *hardware* lainnya yang sesuai (misalnya, pengaturan *port* I/O *digital* dan *analog*, dan mengubah warna LED) |
| **Aplikasi** | Blok set ini memungkinkan pengguna untuk menerapkan algoritma navigasi menggunakan informasi sensorik yang tersedia. Blok ini hanya menggunakan data *encoder* roda dan *bump sensors* untuk navigasi dan menghindari rintangan. |
| ***Image Processing*** | Blok set ini mengimplementasikan akuisisi citra dan pengolahan fungsi menggunakan *Library Open CV*. Blok set pengolahan citra memungkinkan untuk menangkap gambar dari kamera USB, untuk proses *online* dan menyimpannya ke *disk* untuk analisis lebih lanjut. |

Tabel 2. 3 Deskripsi Blok QUARC pada Qbot

| **Blok** | **Deskripsi** |
|---|---|
|  | Blok ini diperlukan untuk membuat sambungan serial ke Qbot. Qbot diidentifikasi oleh *Universal Resource Identifier* (URI), seperti *serial://localhost:1baud=57600,word=8, parity=none,stop=1*, di mana *port* komunikasi 1 dari target terhubung ke *port serial* Qbot dengan parameter yang ditentukan. Catatan: *Output* "rmb" dari blok ini harus terhubung ke *input* "rmb" dari blok di blok Roomba yang ditetapkan dalam rangka untuk mengakses Roomba. |

| Blok | Deskripsi |
|---|---|
|  Roomba Drive | Blok ini menggerakkan Qbot dengan dua *input*, yaitu kecepatan dan radius. Untuk menggerakkan lurus, *input* radius mengambil 32768 atau 32767. |
|  Roomba Query List | Blok ini mengambil berbagai data *sensor* dari Qbot. *Input pack_id* berkisar 7-42, dan setiap *input* akan menampilkan nilai dari *output pack*. Lihat halaman Help MATLAB untuk deskripsi lengkap untuk masing-masing paket ID.<br>Nilai *pack_id* yang penting :<br>8 = Dinding (0 = tidak ada dinding, 1 = dinding terlihat)<br>19 = Jarak (Jarak yang Qbot telah melakukan perjalanan dalam milimeter)<br>20 = Sudut (Sudut dalam derajat Qbot berubah)<br>22 = Tegangan (Tegangan dari baterai Qbot dalam milivolt)<br>23 = Arus (Arus dalam mili ampere mengalir atau keluar dari baterai Qbot) |
|  | Blok inisialisasi yang diperlukan dalam semua model QuaRC menggunakan HIL *Card*. Hanya satu blok HIL *Initialize* yang dibutuhkan dalam model untuk mengatur satu kendaraan. Blok HIL *Read Write* harus referensi blok ini untuk menentukan *board* yang digunakan. |
|  | Versi saat ini dari Qbot DAC yang dapat membaca dari saluran *analog* (untuk sensor infra merah) dan saluran sensor sonar. Saluran PWM yang didukung untuk operasi *write.*<br>. |

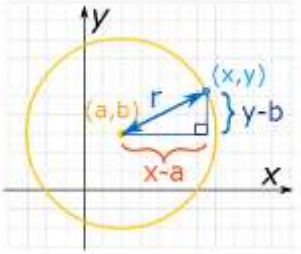
## 2.4 Persamaan Lingkaran

Oleh karena semua *obstacle* berbentuk lingkaran, maka diperlukan cara untuk menyatakan suatu lingkaran yang terletak pada sumbu *Cartesian*. Dalam koordinat *Cartesian*, persamaan lingkaran yang memiliki jari-jari $r$, dengan pusat di $(a, b)$ adalah: [9]

$$(x-a)^2 + (y-b)^2 = r^2 \quad (2.13)$$

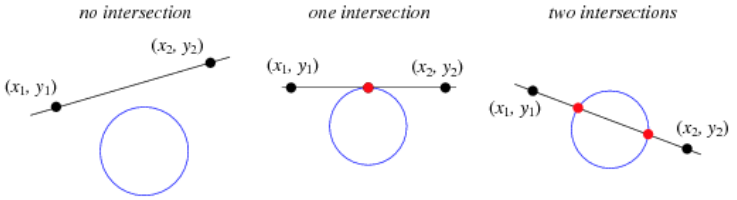
Persamaan (2.13) juga bisa dituliskan dalam bentuk umum

$$x^2 + y^2 + Ax + By + C = 0 \quad (2.14)$$




Gambar 2. 17 Lingkaran dalam Koordinat Kartesius *[9]*

Persamaan garis singgung digunakan dalam penelitian ini untuk menentukan apakah *path* yang dihasilkan akan menabrak *obstacle*. Pada Gambar 2.18, terdapat 3 kemungkinan antara hubungan antara garis yang menghubungkan titik $(x_1, y_1)$ dan $(x_2, y_2)$ mungkin akan bercsinggungan dengan sebuah lingkaran dengan jari-jari $r$ dan pusat di $(a, b)$. Di bagian kiri, terjadi *intersection* di 2 titik imajiner (tidak bersinggunan). Di bagian tengah, terjadi *intersection* di satu titik saja, sedangkan di kanan terjadi *intersection* di dua titik. [10]




Gambar 2. 18 Kemungkinan Garis Menyinggung Lingkaran *[10]*

Untuk membedakan ketiga hal tersebut, dibutuhkan persamaan dari garis lurus dan persamaan lingkaran. Berikut persamaan garis lurus

$$y = mx + n \tag{2.15}$$

Persamaan (2.15) kemudian di substitusi ke Persamaan (2.13) menjadi:

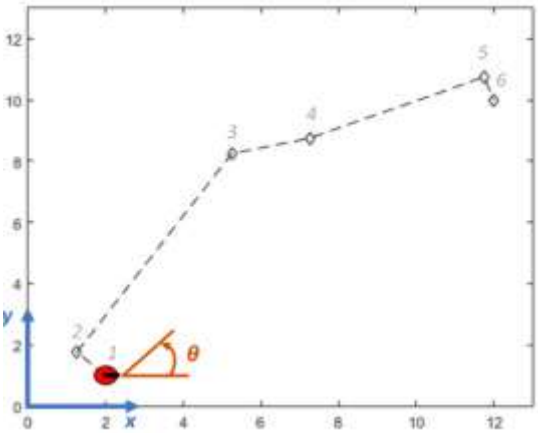
$$x^2 + (mx + n) + Ax + B(mx + n) + C = 0 \tag{2.16}$$

Diskriminan (D) Persamaan (2.16) kemudian menjadi penentu apakah garis tersebut menyinggung lingkaran di 0,1 atau 2 titik. Jika D lebih kecil dari 0, maka garis tidak memotong lingkaran. Jika D sama dengan 0, maka garis menyinggung lingkaran di satu titik. Jika D lebih besar dari 0, maka garis menyingguna lingkaran di dua titik atau dengan kata lain, memotong lingkaran. Teori mengenai garis dan lingkaran ini berguna untuk menentukan apakah *path* yang telah dirancang menyinggung suatu *obstacle* yang dalam kasus ini, berbentuk lingkaran. Dengan kata lain, teori ini diperlukan untuk menguji *feasibility* suatu *path.* [10]

## 2.5 *Pure Pursuit Controller* [11]

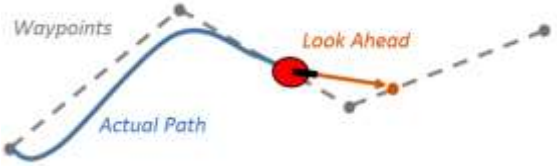
*Pure Pursuit* merupakan sebuah algoritma *path following*. Cara kerja algoritma ini adalah dengan menghitung kecepatan anguler yang menggerakkan robot dari posisi-nya sekarang menuju sebuah titik *look-ahead* yang ada di depan robot. Kecepatan linear robot diasumsikan konstan, sehingga kecepatan linear tersebut bisa diubah kapanpun. Bila dikatakan dengan lebih eksplisit, bisa dibilang robot ini mengejar titik tersebut. Parameter yang bisa diubah adalah seberapa jauh titik look-ahead tersebut.

Gambar 2.19 menunjukkan koordinat *reference* sistem. Input *waypoint* berada di dalam koordinat [x,y], yang kemudian digunakan untuk menghitung kecepatan robot. *Pose* awal robot juga dimasukkan sebagai $\theta$ yang diukur dalam radian mulai 0° berlawanan arah jarum jam.



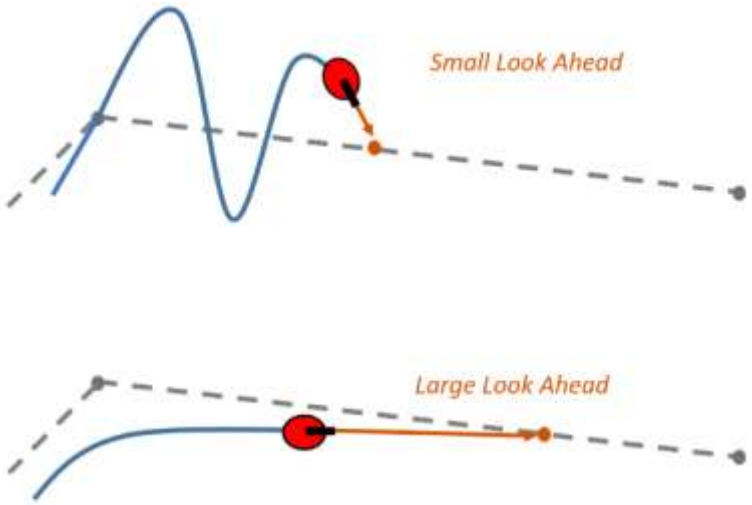

Gambar 2. 19 *Reference Coordinate Frame*

*Look Ahead distance* merupakan seberapa jauh pada *path* robot harus melihat ke depan untuk menghitung kecepatan anguler yang harus digunakan. Gambar 2.20 menunjukkan bagaimana *path* yang dilalui robot belum tentu sama dengan *path* yang direncanakan.




Gambar 2. 20 *Look ahead distance*

Nilai dari *look ahead* sendiri tidak boleh terlalu kecil atau terlalu besar. Nilai yang terlalu kecil akan membuat *path* menjadi terlalu berosilasi dan nilai yang terlalu besar akan membuat *path* nya menjadi terlalu menyimpang.



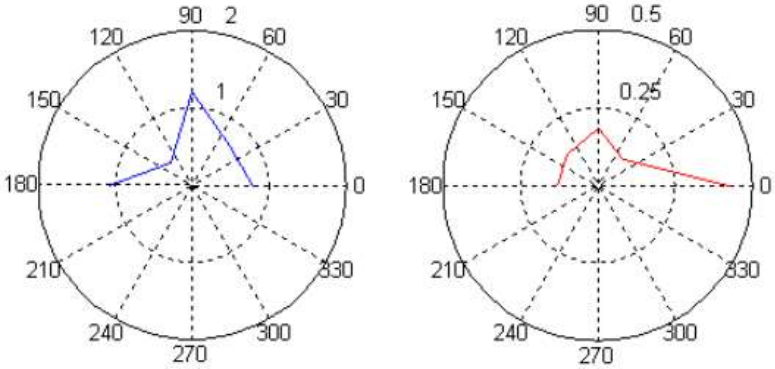

Gambar 2. 21 Pengaruh Nilai *Look ahead*

## 2.6 *Polar Histogram* [12]

*Polar histogram* masuk dalam kategori *reactive path planning* karena ia memanfaatkan data dari sensor yang diperoleh secara *real time* untuk merencanakan *path* yang hendak ditempuh. *Polar histogram* mencari arah yang paling aman untuk dilewati robot dengan cara memetakan *polar density* dari *obstacles*. *Obstacle polar density* bisa diperoleh dari mengambil nilai *weighted average* dari *inverse* data jarak (dengan kata lain jarak ke *obstacle*) pada masing-masing orientasi sensor. Kelima sensor yang ada pada robot yang digunakan dalam implementasi mampu mendeteksi *obstacle* yang berada 20 hingga 150 cm dari robot. Gambar 2.22 bagian kiri menunjukan data jarak yang diperoleh sensor *infrared* yang menghadap arah $0°, 45°, 90°, 135°, 180°$. Gambar 2.22 bagian kanan menunjukkan *polar obstacle density* (POD) dari data jarak pada Gambar 2.22 kiri. Adapun POD tersebut diperoleh dengan rumus berikut.

$$POD_i = \frac{1}{6} f_{norm}^2(d_{i-1}) + \frac{2}{3} f_{norm}^2(d_i) + \frac{1}{6} f_{norm}^2(d_{i+1}) \quad (2.17)$$

$$f_{norm}^{2}(d_i) = 1 - \frac{\min(d_{th}, d_i)}{d_{th}} \tag{2.18}$$




Gambar 2. 22 *Polar Plot* dari *Range Data* (kiri) dan *Normalised Polar Obstacle Density* (kanan)

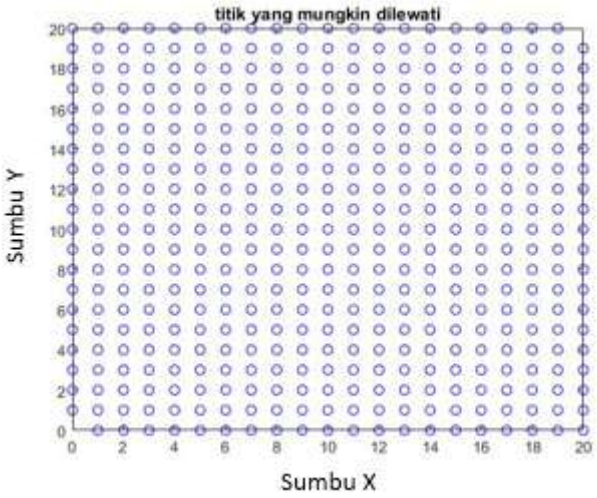
Dimana $i = 1, \dots, 5$ menandakan sensor ke-i, yang menghadap ke arah $45 \times (i - 1)$, dan $d_i$ adalah jarak *obstacle* yang diperoleh sensor ke-i. Jarak maksimum dari sensor dinyatakan dengan simbol $d_{th}$ yang memiliki nilai $150\ cm$. Selain itu, pada Persamaan (2.17), $d_6 = d_0 = 0$. Dengan demikian, dapat disimpulkan bahwa $45°$ merupakan arah yang memiliki *polar obstacle density* paling rendah sehingga merupakan arah gerak robot.

# BAB 3
# PERANCANGAN SISTEM

Pada bab ini akan dibahas tahapan dalam merancang alur algoritma *path planning* sebuah mobile robot secara global dan lokal berbasis *genetic algorithm*, yang meliputi identifikasi masalah, perancangan algoritma genetika, perancangan simulasi dan implementasi dalam lingkungan statis dan dinamis.

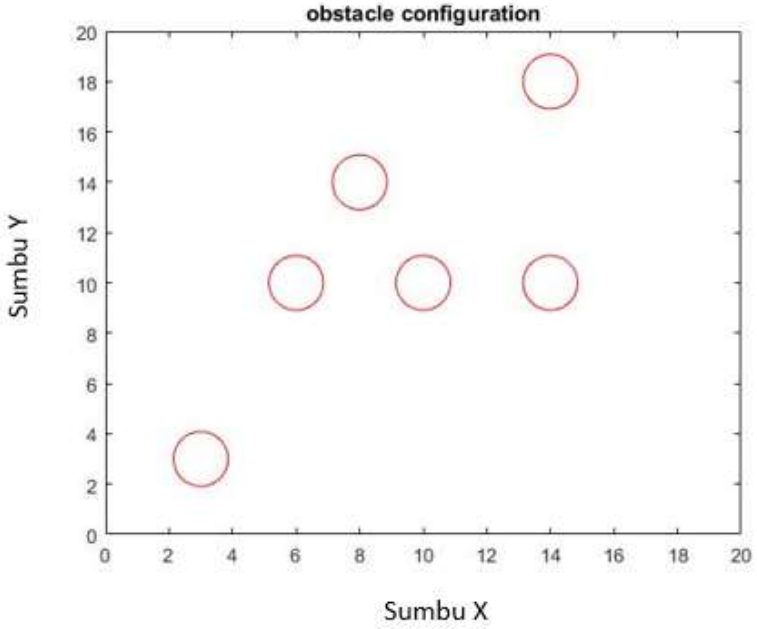
## 3.1 Identifikasi Masalah

Pada tugas akhir ini, permasalahan yang akan dibahas adalah permasalahan *path planning* yang bertujuan menemukan sebuah *path* terpendek antara dua titik yanerada dalam sebuah *map* yang merupakan representasi. *Path planning* sendiri merupakan salah satu permasalahan yang paling mendasar dalam permasalahan navigasi *mobile robot*. *Dynamic path planning* berbasis *genetic algorithm* akan diterapkan dalam tugas akhir ini untuk menyelesaikan permasalahan navigasi dalam lingkungan yang memiliki *moving obstacles*.




Gambar 3. 1 Titik yang bisa dilewati

*Map* yang digunakan sebagai representasi area kerja robot memiliki *grid* yang seragam dengan ukuran 20x20 *grid*. *Grid* tersebut membatasi titik yang mungkin menjadi *waypoint,* sehingga *waypoint* selalu merupakan bilangan bulat. Titik awal gerak robot (*starting point*) diletakkan di ujung kiri bawah, sedangkan titik tujuannya (*end point*) diletakkan di ujung kanan atas. *Obstacle* yang digunakan selalu berbentuk lingkaran dengan radius yang seragam namun tersebar pada titik sebarang didalam *map*. Gambar 3.1 menggambarkan contoh konfigurasi *obstacle* yang mungkin terjadi. *Obstacle* dalam kasus ini melambangkan sebuah area yang tidak boleh dilewati oleh robot, dengan kata lain, koordinat tempat obstacle berada tidak mungkin terdaftar sebagai titik yang mungkin menjadi kandidat *waypoint*.




Gambar 3. 2 Salah Satu Konfigurasi *Obstacle*

Akan tetapi, seperti yang telah dicantumkan dalam batasan masalah, dalam tugas ini tidak dibahas mengenai permasalahan estimasi tabrakan *obstacle* dinamis, dengan demikian, untuk menunjukkan kemampuan sistem untuk mencari *path* baru, tabrakan tersebut akan disimulasikan. Simulasi yang dimaksud adalah setelah mencapai *waypoint* pertama, akan dicari sebuah rute baru karena akan terjadi tabrakan. Setelah itu, rute baru tersebut akan langsung diterapkan. Hal ini menimbulkan masalah baru, dimana pencarian rute baru harus ditemukan dalam waktu yang cepat agar bisa ditemukan *waypoint* dari *path* yang bebas dari *obstacle*.

*Mobile robot* yang akan digunakan dalam penelitian ini adalah robot beroda, karena akan dioperasikan dalam sebuah bidang dua dimensi. Selain itu, robot beroda memiliki keunggulan dalam kemudahan untuk memodelkan dibandingkan *mobile robot* yang lain. Dalam dinamika kendaraan di dunia nyata, sebuah kendaraan perlu menggunakan sistem suspensi untuk mengkompensasi gerakan vertikal yang timbul setelah kendaraan bergerak dengan kecepatan yang cukup tinggi.

Robot beroda yang digunakan merupakan *differential drive robot*, yakni robot yang memiliki dua *drive wheels* yang bisa bergerak secara independen. Gerak roda yang independen tersebut memungkinkan robot untuk melakukan gerak melingkar (rotasi) dengan mengatur kecepatan roda sehingga roda berputar dengan arah berlawanan. Kemudian, berdasarkan model matematika *differential drive robot,* dirancang simulasi dan implementasi untuk membuat mengikuti *(path following) waypoint* yang telah ditemukan menggunakan *path planning* berbasis *genetic algorithm*.

## 3.2 Perancangan Algoritma *Path Planning* berbasis GA

Algoritma genetika (*Genetic algorithm)* merupakan teknik pencarian (*search technique*) yang meniru operator genetika natural. Langkah awal yang dilakukan untuk merancang algoritma *path planning* berbasis algoritma genetika ini diawali dengan inisialisasi populasi. Ukuran populasi yang dibangkitkan dalam proses pencarian *path* menentukan kecepatan program dan banyaknya kromosom. Setiap kromosom merupakan *path* yang mungkin menjadi jawaban dari permasalahan *path planning* (tidak menabrak *obstacle*). Oleh karena itu, diperlukan informasi mengenai letak *obstacle* secara global agar bisa

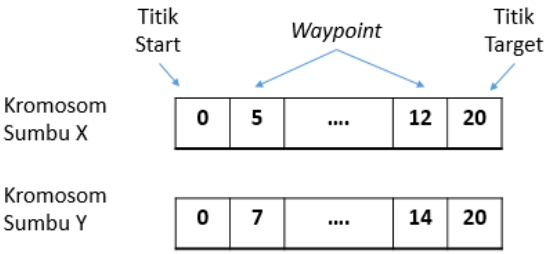
memastikan bahwa kromosom yang dibangkitkan telah memenuhi kriteria tersebut. Setelah selesai membangkitkan populasi awal hingga sesuai nilai populasi yang telah ditetapkan, setiap kromosom akan dievaluasi menggunakan *fitness function* untuk mengetahui kromosom yang memiliki jarak terpendek pada generasi tersebut. Setelah itu, dilakukan proses seleksi menggunakan seleksi *roulette wheel* untuk memilih kromosom yang akan dikawin-silangkan (*crossover*). Setelah dilakukan *crossover,* proses berikutnya adalah mutasi. Lalu, setelah diperoleh sebuah kromosom baru yang masih harus dicek terlebih dahulu apakah ia memenuhi syarat bebas *obstacle.* Demikian penjelasan mengenai proses *path planning* dalam lingkungan yang statis.

Untuk permasalahan *path planning* dalam sebuah lingkungan dengan *obstacle dinamis,* timbul masalah baru dimana proses pencarian *path* baru harus dilakukan dengan waktu yang cukup cepat. Untuk mengatasi masalah tersebut, kromosom terbaik yang diperoleh untuk mencari *path* pada permasalahan *path planning* dalam lingkungan statis digunakan dalam proses pembangkitan populasi awal sehingga bisa memotong waktu komputasi. Alur Kerja *dynamic path planning* berbasis *genetic algorithm* terlampir pada subbab-subbab berikutnya.

### 3.2.1 Insialisasi Populasi

Inisialisasi populasi adalah proses dimana kromosom yang mewakili *path* yang perlu dilalui untuk mencapai titik tujuan pada map yang telah didefinisikan. Di sebuah *map* yang memiliki dua dimensi, untuk memperoleh posisi sebuah titik dalam *map* tersebut, maka diperlukan pengetahuan akan posisinya terhadap sumbu x dan sumbu y. Oleh karena itu, setiap kali proses pembuatan kromosom, diperlukan dua kromosom yang mewakili posisi terhadap sumbu x dan sumbu y.

Di dalam masing-masing kromosom, terdapat gen yang mewakili *waypoint* dalam suatu *path*. Dalam kasus ini, gen tetap menggunakan pengkodean desimal. Pada Gambar 3.3, terdapat ilustrasi mengenai pengkodean gen yang digunakan di tugas akhir ini.




Gambar 3. 3 Konfigurasi Kromosom

Dalam proses inisialisasi, terdapat beberapa parameter yang harus diketahui seperti: ukuran populasi, ukuran kromosom (panjang *waypoint*), dan posisi setiap *obstacle* di dalam *map*. Parameter-parameter tersebut harus diketahui karena dalam setiap proses pembuatan kromosom (*chromosome generation*) dibangkitkan kromosom (dengan ukuran tertentu) sejumlah ukuran populasi. Setelah itu, kromosom yang dibangkitkan harus dipastikan untuk tidak melewati *obstacle.* Jika kromosom yang baru ternyata melewati *obstacle* yang telah ditentukan, maka akan dicari lagi sebuah kromosom hingga akhirnya ditemukan yang bebas dari halangan. Pada Gambar 3.4 terdapat contoh sebuah populasi dengan ukuran populasi 10.

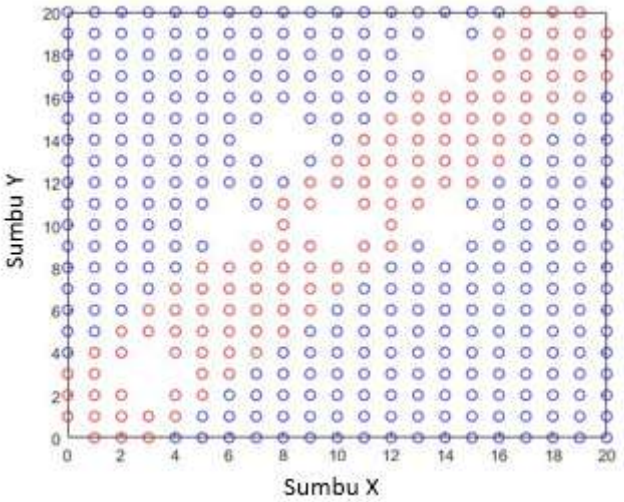
```
ych =                          xch =

     0     5     8    20          0    15    20    20
     0     4     1    20          0    15    14    20
     0     1     0    20          0    19    15    20
     0     9    20    20          0    20     8    20
     0     3    14    20          0    14    19    20
     0     0     9    20          0    10    20    20
     0     8     8    20          0    19    15    20
     0     4     7    20          0     9    15    20
     0    17    20    20          0     6    16    20
     0    11    19    20          0     1     9    20
```

Gambar 3. 4 Contoh Populasi dengan Ukuran 10

Dalam proses pemilihan titik yang mungkin terpilih menjadi sebuah kromosom, digunakan *weighted sampling*. *Weighted sampling*

adalah metode sampling yang memberi suatu nilai (*weight*) pada setiap titik yang mungkin, sehingga terdapat daerah tertentu yang memiliki probabilitas yang lebih besar dibanding daerah lain. Pada titik awal dan titik akhir yang telah ditetapkan [(0,0) dan (20,20)], jika tidak terdapat *obstacle*, maka kira-kira *waypoint* yang dihasilkan akan berupa garis lurus dari titik awal ke akhir. Dengan demikian, untuk penentuan *weight*, pada daerah yang dekat (*radius obstacle* + 2) diukur dari garis $x = y$, akan diberi probabilitas 0.55 dibagi dengan jumlah titik yang ada dalam daerah tersebut. Sedangkan daerah yang lain akan memiliki probabilitas sebesar 0.45 dibagi dengan jumlah titik yang ada dalam daerahya. Distribusi probabilitas yang digunakan dalam sampling diilustrasikan lebih lanjut pada Gambar 3.5.




Gambar 3. 5 *Weighted Sampling*

### 3.2.2 *Fitness Function* dan Seleksi

*Fitness function* adalah fungsi yang digunakan untuk mengukur kedekatan suatu kandidat solusi dengan tujuannya. Dalam permasalahan *path planning, fitness function*-nya adalah jarak terpendek dari posisi awal ke posisi tujuan yang diinginkan. Oleh karena itu, *fitness function* diharapkan meningkat seiring dengan menurunnya jarak, dengan kata

lain, *fitness function* memiliki hubungan terbalik dengan persamaan jarak dari dua titik (*distance*).

$$distance(i,j) = \sqrt{\left(x_i - x_j\right)^2 + \left(y_i - y_j\right)^2} \tag{3.19}$$

$$fitness\ function = \frac{1}{distance(i,j)} \tag{3.2}$$

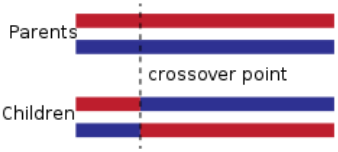
Oleh karena dalam proses pembuatan kromosom telah dipastikan bahwa semua kromosom pasti tidak melewati *obstacle* (*feasible*), maka nilai *fitness function* pada saat *path*-nya melewati *obstacle* tidak perlu ditentukan.

Setelah setiap kromosom telah diketahui nilai *fitness*-nya, dilakukan proses seleksi yang bertujuan untuk memilih kromosom yang akan dimasukkan ke dalam proses *crossover* (memilih *parent*). Proses seleksi *roulette wheel* sering kali disebut sebagai *fitness proportionate selection,* dimana proses seleksi dilakukan secara acak, namun kromosom dengan nilai *fitness* besar diberi probabilitas yang besar juga (proporsional). Proses tersebut dilakukan sebanyak *parent* yang diperlukan untuk membangkitkan generasi berikutnya yang memiliki jumlah populasi yang sama. Persamaan mengenai probabilitas sebuah kromosom terpilih sebagai parent melalui *roulette wheel selection* adalah sebagai berikut: (dengan N sebagai total populasi.)

$$P(parent = i) = \frac{fitness(i)}{\sum_{j=1}^{N} fitness(j)} \tag{3.3}$$

### 3.2.3 Operator Genetika

Operator genetika yang digunakan dalam kasus ini adalah *crossover* dan mutasi. *Crossover* (kawin silang) merupakan proses rekombinasi 2 *parent* untuk memperoleh dua kromosom baru. Metode *crossover* yang digunakan adalah metode *single point crossover*, dimana gen dalam kedua kromosom *parent* setelah *point crossover* ditukar. Nilai Pc (*crossover probability)* merupakan probabilitas bahwa sepasang kromosom akan melalui proses *crossover*.




Gambar 3. 6 *Crossover*

Operator genetika mutasi digunakan pada kromosom yang dihasilkan setelah melalui proses *crossover*. Mutasi diperlukan agar dimungkinkan muncul gen (*waypoint*) baru yang tidak ada di populasi sebelumnya. Di dalam kasus ini, semua kromosom melalui proses mutasi. Proses ini diawali dengan memilih secara acak (*weighted sampling*) gen di dalam sebuah kromosom yang akan melalui proses mutasi menggunakan nilai probabilitas Pm (*mutation probability*). Setelah terpilih, maka akan dipilih secara random sebuah angka yang diambil dari database tentang titik-titik yang tidak melewati *obstacle*. Baru setelah itu, dilanjutkan dengan uji *feasibility* untuk menetukan apakah garis yang dibentuk kedua titik menyinggung *obstacle*.

### 3.2.4 Uji *Feasibility* Kromosom

Pada tahap ini, kromosom yang telah ditemukan dari proses-proses sebelumnya dites *feasibility*-nya, dimana kriteria *feasible*-nya adalah baik titik belok maupun *path* yang berupa garis lurus antara kedua titik tidak boleh melewati *obstacle*. Tes dilakukan menggunakan persamaan matematika untuk mengecek apakah *path* (garis lurus) bersinggungan dengan suatu *obstacle* (lingkaran). Hal ini dilakukan dengan menerapkan teori garis singgung lingkaran yang telah terlampir di Bab 2.

Proses pengujian ini dilakukan setiap selesai proses pembentukan kromosom dan proses mutasi. Setelah proses *crossover* tidak dicek karena selain proses ini menggunakan *computational cost* yang besar, proses *crossover* juga langsung diikuti oleh proses mutasi yang membuat proses ini menjadi *redundant*.

### 3.2.5 *Elitism*

Salah satu bentuk upaya untuk mempertahankan kromosom terbaik dalam suatu generasi adalah dengan menerapkan *elitism*. Dalam proses *elitism,* kromosom yang memiliki nilai fitness paling tinggi tidak

akan mengikuti proses *crossover* dan mutasi, melainkan akan langsung menjadi kromosom berikutnya. Dalam tugas akhir ini, *elitism* diterapkan dengan hanya mengambil 1 kromosom terbaik agar populasi yang lain bisa berkesempatan mencari solusi yang lebih baik.

### 3.2.6 Alur Kerja *Path Planning* Statis

Alur kerja *path planning* statis adalah:

1. Tentukan parameter-parameter yang diperlukan seperti: ukuran populasi, ukuran kromosom, nilai Pc dan Pm, banyak generasi, apakah *elitism* berlaku, posisi dan jari-jari *obstacle,* dll.
2. Inisialisasi populasi pertama
3. Pastikan baik titik maupun garis *path* tidak menyinggung *obstacle*.
4. Untuk generasi pertama hingga ke-N, lakukan:
    a. Hitung nilai *fitness* masing-masing kromosom
    b. Seleksi
    c. *Crossover*
    d. Mutasi
    e. Pastikan baik titik maupun garis tidak menyinggung *obstacle*.
5. End.

## 3.3 Perancangan Simulasi

Simulasi mengenai *path planning* di lingkungan statis, dinamis dan *path following* oleh sebuah *differential drive robot* dilakukan menggunakan *software* MATLAB. Kode program untuk menjalankan program *path planning* di lingkungan statis dan dinamis terlampir. Simulasi *differential drive robot* untuk menjalankan *path* yang telah dihasilkan disusun berdasarkan persamaan dan teori yang terlampir di Sub-Bab 2.3.

Simulasi *path following* diperlukan untuk menunjukkan kapabilitas algoritma untuk melakukan *replanning* dari sebuah *path* yang tiba-tiba terhalang oleh sebuah *obstacle* setelah ia menjalani *path* asal. Simulasi *path following* ini men-simulasikan bagaimana sebuah *differential drive robot* akan bergerak mengikuti sebuah set *waypoints* hingga mencapai tujuan. Adapun kontroler yang digunakan adalah *PurePursuit* yang telah disinggung di Bab 2.

Simulasi *differential drive robot* yang digunakan memiliki spesifikasi sebagai berikut:

Tabel 3. 1 Spesifikasi *Differential Drive Robot* pada Simulasi

| **Deskripsi** | ***Value*** |
|---|---|
| Jari-jari robot | 0,2 m |
| Kontroler | *PurePursuit* |
| Kecepatan yang diinginkan | 0,3 m/s |
| Kecepatan Anguler maksimum | 2 rad/s |
| *Look ahead distance* | 0,5 m |
| *Rate* Kontroler | 10 Hz |

Oleh karena robot tidak memperoleh informasi sensor saat bergerak, maka *path* yang dirancang harus dipastikan untuk tidak menyinggung *obstacle*. Hal ini dilakukan dengan memberi toleransi 0,25 m pada setiap *obstacle,* yang berarti, ada toleransi kesalahan sebesar 0,05 m bagi program *path following* agar tidak menyerempet *obstacle*.

Urutan program simulasi *dynamic path planning:*

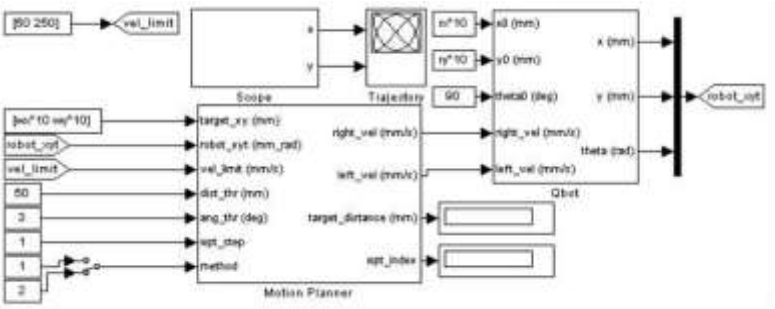
1. Menjalankan program *path planning* statis untuk memperoleh satu set *waypoint* yang minimum.
2. Menjalankan program *path following*, memasukkan *waypoint* yang telah diperoleh.
3. Setelah mencapai *waypoint* pertama, sinyal dummy bahwa akan terjadi tabrakan diberikan, sehingga diperlukan *re-planning.*
4. Menjalankan program *path planning* statis, namun digunakan *waypoint* lama sebagai populasi pertama untuk mempercepat proses.
5. Menjalankan program *path following* menggunakan *path* yang baru.

## 3.4 Perancangan Implementasi Menggunakan Qbot

Pada sub-bab ini akan dibahas *global path planning* berbasis algoritma genetika dan kemudian *dynamic path planning* yang menggunakan algoritma genetika (*global path planning)* dan *polar histogram (local path planning)*.

### 3.4.1 *Global Path Planning*

Implementasi dirancang menggunakan *software* MATLAB Simulink, yang telah terpasang *software* QuaRC. QuaRC sendiri bertujuan untuk menghubungkan MATLAB Simulink dengan perangkat Qbot sehingga dimungkinkan untuk men-*download* program yang telah dirancang di Simulink maupun MATLAB ke Qbot. Selain digunakan untuk keperluan men-*download* program, QuaRC juga digunakan untuk mengirim data dari dan menuju Qbot secara *real time.*



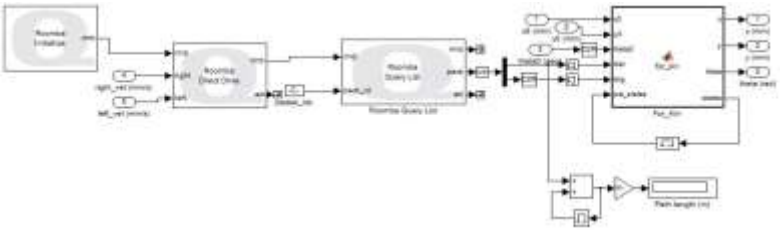

Gambar 3. 7 Diagram Simulink *Global Path Planning*

Menggunakan software tersebut, dirancang sebuah diagram blok Simulink agar bisa menghubungkan antara program *path planning* dengan Qbot. Pada Gambar 3.7, variabel wx dan wy merupakan *waypoint*, dan rx dan ry merupakan posisi awal robot. Blok Qbot akan diberi masukan vr dan vl (*right and left velocity*) dan mengeluarkan posisi robot saat itu. Posisi kemudian dikembalikan lagi ke blok *Motion planner* agar bisa diputuskan lagi vr dan vl nya.

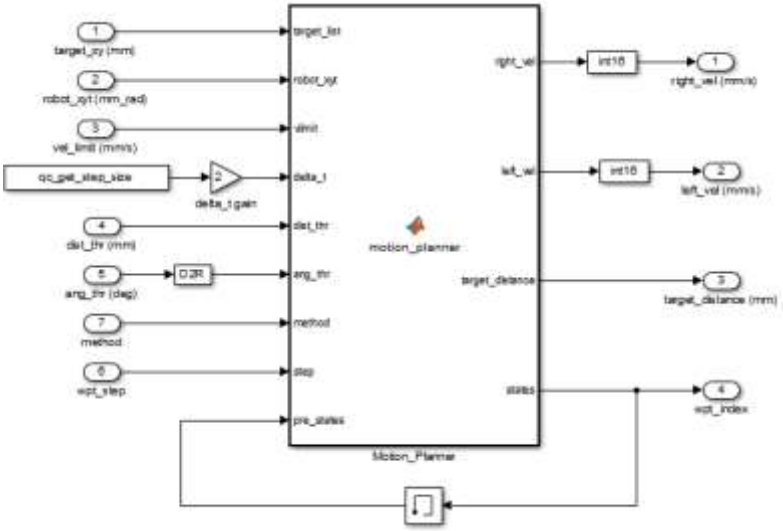
Dalam implementasi *global path planning*, digunakan *map* $200 \times 200\ cm$ dengan ukuran grid 10 cm. Program yang digunakan kemudian adalah program *path planning* statis seperti saat simulasi untuk mencari *waypoint*. *Waypoint* tersebut kemudian dimasukkan ke program *Global Path Planning* (Gambar 3.7) yang akan menggerakkan robot.

*Global path planning* tidak menggunakan informasi sensor untuk menghasilkan *waypoint*, melainkan hanya mengandalkan informasi

lingkungan yang diprogram. Oleh karena itu, informasi tersebut harus mencangkup keseluruhan dari lingkungan, dan hanya bisa menghindar dari *obstacle* statis.



Gambar 3. 8 Sub Blok Qbot dalam *Global Path Planning*

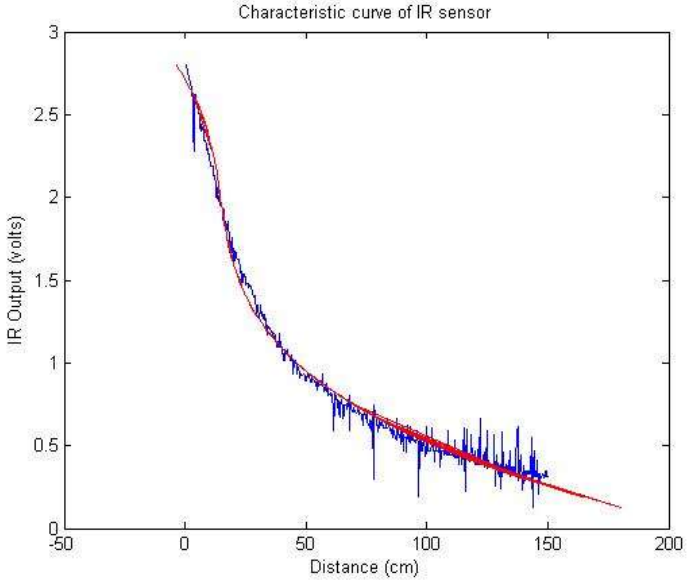



Gambar 3. 9 Sub Blok *Motion Planner* dalam *Global Path Planning*

### 3.4.2 Kalibrasi Sensor *Infrared* [12]

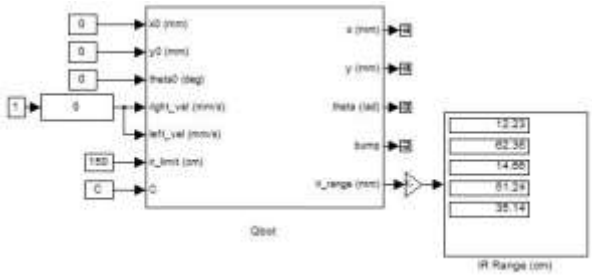
*Mobile robot* Qbot dilengkapi dengan 5 sensor *infrared* SHARP GP2Y0A02YK IR [13] yang mampu mendeteksi objek dengan jarak 20 hingga 150 cm. Sensor *infrared* diletakkan menghadap arah $0°, 45°, 90°, 135°, 180°$. Informasi yang dihasilkan sensor berupa tegangan yang kemudian diterjemahkan menjadi data *range*. Proses penerjemahan informasi tersebut diawali dengan kalibrasi sensor dengan

cara meletakkan sebuah objek kemudian robot dijalankan menjauhi objek tersebut. Data analog Output sensor kemudian di-*fitting* untuk mengetahui koefisien polynomial yang paling mendekati (kurva karakteristik sensor *infrared*)




Gambar 3. 10 Kurva Karakteristik Sensor *Infrared*

Gambar 3.10 menunjukkan bahwa data analog O*utput* sensor bisa didekati dengan polynomial orde 4 dengan koefisien polinomial (1,2363; -29,8753; 149,4331; -289,2674 dan 214,6719). Setelah diketahui koefisien polynomial, langkah berikutnya yang dilakukan adalah membandingkan pengukuran menggunakan sensor *infrared* dengan pengukuran menggunakan penggaris. Adapun hasil pengukuran yang diperoleh adalah (12,3; 62,5; 15,1; 51,9; 35) cm. Hasil ini namun berubah terus-menerus dan beberapa melewati batas toleransi kesalahan menurut datasheet yaitu $\pm 0{,}3\ mm$. [13]

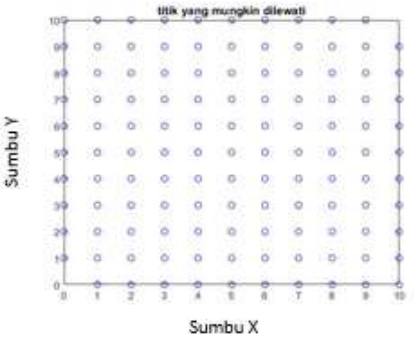



Gambar 3. 11 *Range Infrared*

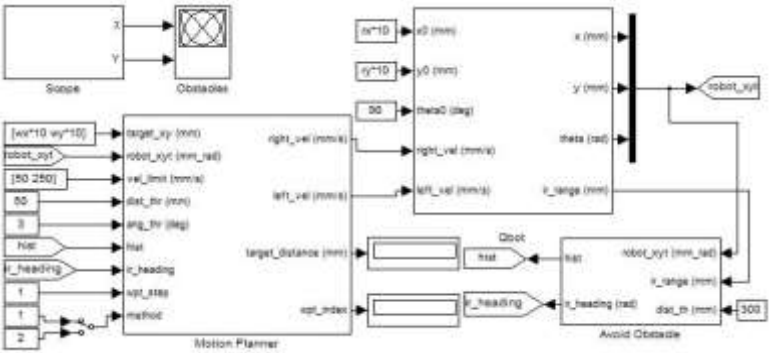
### 3.4.3 *Dynamic Path Planning*

*Dynamic path planning* menggabungkan metode *global path planning* dan *local path planning* agar robot bisa menghindari *obstacle* dinamis. *Local path planning* memanfaatkan sensor yang ada pada robot untuk menentukan arah gerak robot. Dalam kasus ini, akan digunakan 5 sensor *infrared* yang ada pada Qbot. Kemudian, informasi tersebut akan digunakan dalam proses *Polar Histogram* untuk menentukan sudut mana yang paling aman untuk dituju robot.

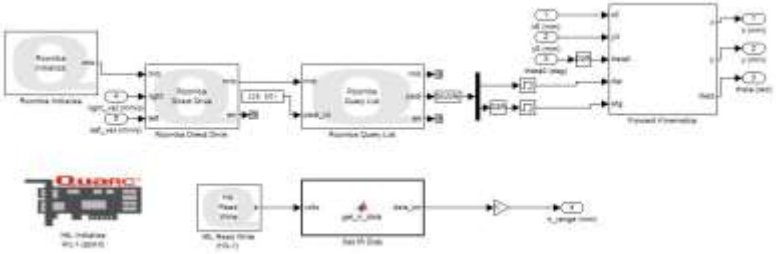
Lingkungan tempat robot akan dijalankan memiliki sedikit perbedaan dengan lingkungan yang sebelumnya, yakni ukuran grid diubah menjadi 10x10 dengan ukuran masing-masing grid sebesar 20x20 cm. Hal ini disebabkan oleh sensor *infrared* yang tidak linear jika jarak *obstacle* kurang dari 20 cm. Dengan demikian, dibuat sebuah grid seperti pada Gambar 3.12.




Gambar 3. 12 *Grid 10x10*




Gambar 3. 13 Diagram Simulink *Dynamic Path Planning*



Gambar 3. 14 Sub Blok Qbot pada *Dynamic Path Planning*

*Halaman ini sengaja dikosongkan*

# BAB 4
# HASIL DAN ANALISA

Pada bab ini akan dibahas hasil dan analisa yang diperoleh setelah menjalankan simulasi dan implementasi *path planning* dalam lingkungan yang statis dan dinamis berbasis algoritma genetika. Simulasi akan dilakukan menggunakan *software* MATLAB, sedangkan untuk implementasi, akan digunakan *differential drive robot* Qbot yang diproduksi oleh iRobot.

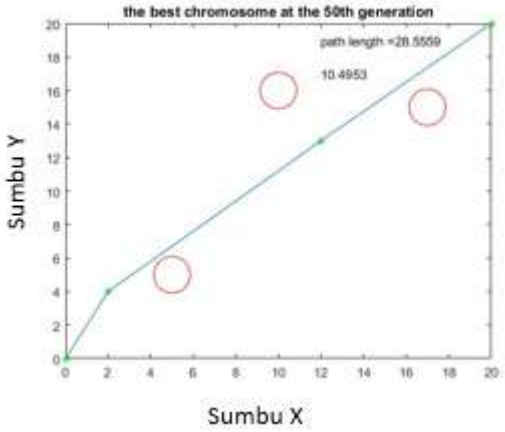
## 4.1 Simulasi *Path Planning* dalam Lingkungan Statis

Seperti yang telah disinggung dalam bab sebelumnya, tujuan dari *path planning* adalah menemukan *waypoint* yang bebas dari *obstacle* untuk mengarahkan robot bergerak dari titik awal (0, 0) ke tujuan (20, 20). Selain informasi mengenai *obstacle,* sebelum menjalankan program, harus ditetapkan dahulu ukuran populasi, ukuran kromosom, banyak generasi, nilai Pc dan Pm dan apakah *elitism* berlaku.

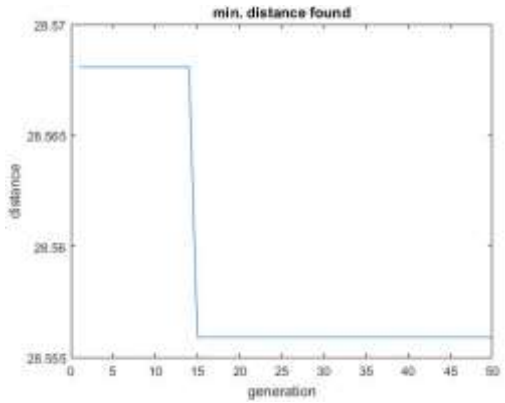
Pada subbab ini, akan dibahas mengenai pengaruh dari jumlah *obstacle* dalam sebuah *map,* pengaruh dari nilai elitism, dan pengaruh dari ukuran populasi dan ukuran kromosom.

### 4.1.1 Pengaruh dari Jumlah *Obstacle*

Menggunakan nilai-nilai berikut, ukuran populasi = 30, ukuran kromosom = 4 (2 *via points*), dilakukan dalam 50 generasi, Pc = 0,9 dan Pm =0,3 dan dengan menerapkan *elitism*, algoritma ini dicoba di berbagai macam konfigurasi *obstacle*. Pada Gambar 4.1, *path* terpendek dalam *map* dengan 3 *obstacle* ditemukan setelah 10,4293detik dengan panjang lintasan = 28,5559. Nilai itu mulai konvergen pada generasi ke-15.

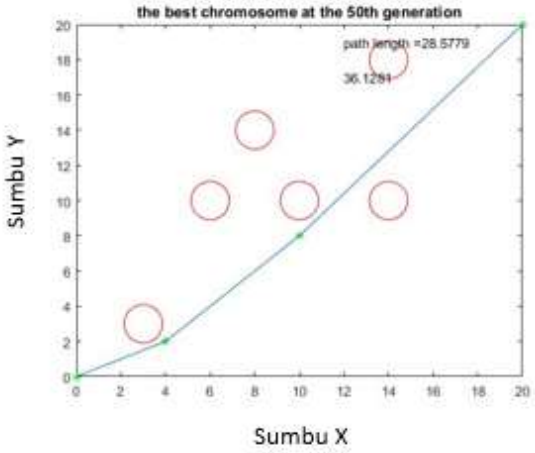



Gambar 4. 1 Path Terbaik dalam 50 Generasi di konf. 3 *Obstacle*

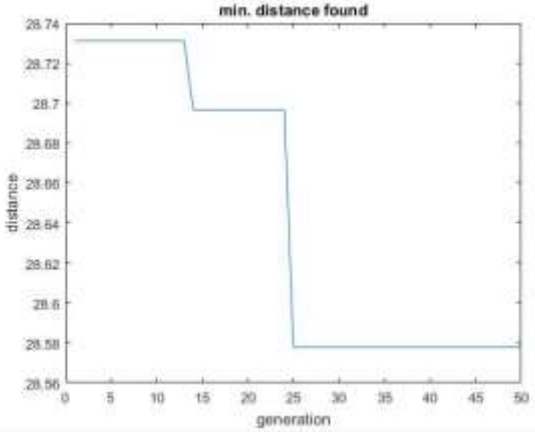



Gambar 4. 2 Generasi Terhadap *Path Length*

Menggunakan nilai parameter yang sama, namun jika konfigurasi *obstacle* diubah menjadi lebih banyak, *path* akan masih bisa ditemukan (dengan *path length* = 28,5779). Akan tetapi waktu komputasi-nya lebih lama, yakni 36,12 detik. Jika membahas nilai *path length* yang diperoleh, maka selisih yang diperoleh saat 3 *obstacle* dengan tanpa *obstacle* adalah 0,2731, sedangkan saat 6 *obstacle* adalah 0,2951 (*path length* tanpa *obstacle* adalah $20\sqrt{2} = 28{,}2828$).




Gambar 4. 3 Path Terbaik dalam 50 Generasi di konf. 6 *Obstacle*

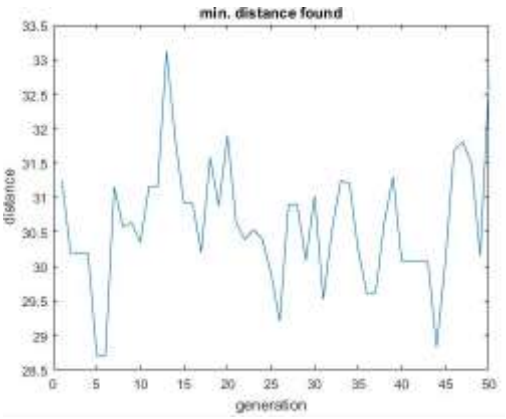



Gambar 4. 4 Generasi Terhadap *Path Length*

### 4.1.2 Pengaruh dari *Elitism*

*Elitism* berpengaruh untuk menjaga agar setiap generasi ditemukan kromosom yang memiliki nilai *fitness* yang lebih baik daripada generasi sebelumnya. Pengaruh dari *elitism* terlihat pada Gambar 4.5, dimana grafik *path length* dan generasi terlihat naik – turun, padahal parameter yang lain dijaga pada nilai yang sama dengan yang digunakan

Gambar 4.4, yang menghasilkan path yang lebih pendek pada generasi berikutnya. Pada grafik yang tidak menggunakan *elitism*, *path* terpendek justru ditemukan pada generasi ke-5.




Gambar 4. 5 Generasi terhadap *Path Length*

### 4.1.3 Pengaruh dari Ukuran Populasi dan Kromosom

Ukuran populasi menentukan banyak kromosom yang dibangkitkan setiap generasi. Dengan kata lain, ukuran populasi yang besar akan mampu mencangkup area yang lebih besar dibanding yang kecil. Akan tetapi, ukuran populasi akan membuat *computational cost* semakin besar. Untuk melihat efek dari ukuran kromosom saja, parameter yang lain dibuat konstan (ukuran kromosom = 4, konfigurasi 6 *obstacle,* dilakukan dalam 50 generasi, Pc = 0,9 dan Pm =0,35 dan dengan menerapkan *elitism*).

Tabel 4. 1 Pengaruh Ukuran Populasi, Panjang *Path*, dan Generasi

| **Ukuran Populasi** | **Panjang *Path*** | **Waktu Komputasi** | **Konvergen di Generasi ke-** |
|---|---|---|---|
| 6 | 28,6143 | 13,5346 s | 35 |
| 30 | 28,5779 | 36,1261 s | 25 |
| 60 | 28,5779 | 69,9026 s | 20 |
| 100 | 28,5921 | 109,3229 s | 7 |

Dari Tabel 4.1, mungkin tidak terlihat trade-off antara waktu komputasi dengan panjang *path*. Mungkin hal ini disebabkan oleh semesta yang relatif kecil (ukuran grid yang 20x20 dan ukuran kromosom yang hanya 4).

Jika menggunakan analogi, seharusnya dalam suatu populasi yang besar, terdapat kemungkinan lebih besar bahwa ada "seseorang" (kromosom) dengan "bakat" (nilai *fitness*) yang luar biasa. Dengan demikian, seharusnya akan dibutuhkan generasi lebih banyak bagi suatu "masyarakat" dengan populasi yang kecil agar bisa "menghasilkan seseorang" dengan "bakat" yang luar biasa, dan sebaliknya. Hal serupa juga ditemukan di kasus ini, dimana pada Tabel 4.1, terdapat hubungan berbalik lurus antara besar populasi dengan pada generasi ke-berapa nilai jarak terpendek ditemukan.

Ukuran kromosom merepresentasikan panjang *waypoint*. Disini, dilakukan 3 percobaan, yakni menggunakan 1, 2 dan 3 via points. Untuk melihat efek dari ukuran kromosom saja, parameter yang lain dibuat konstan (ukuran populasi = 30, konfigurasi 6 *obstacle,* dilakukan dalam 50 generasi, Pc = 0,9 dan Pm =0,35 dan dengan menerapkan *elitism*).

Tabel 4. 2 Pengaruh Panjang Kromosom, Panjang *Path* dan Generasi

| **Panjang Kromosom** | **Panjang *Path*** | **Waktu Komputasi** | **Konvergen di Generasi ke-** |
|---|---|---|---|
| 3 | 28,7309 | 11,2736 s | 2 |
| 4 | 28,5779 | 36,1261 s | 25 |
| 5 | 28,734 | 108,1085 s | 19 |
| 5 (100 generasi) | 28,6883 | 215,7416 s | 55 |

Dari Tabel 4.2, dapat disimpulkan bahwa panjang kromosom akan mempengaruhi waktu komputasi, dimana semakin panjan kromosom, maka waktu komputasi juga akan menjadi semakin lama. Dalam kasus panjang kromosom = 5, terlihat bahwa 50 generasi masih belum cukup untuk menghasilkan sebuah lintasan terpendek, mengingat pada generasi ke-55 masih terjadi perubahan nilai lintasan lagi.

Selain waktu komputasi, penentuan nilai kromosom juga akan mempengaruhi nilai lintasan terpendek yang dihasilkan. Dalam kasus panjang kromosom = 3, meskipun waktu komputasi yang diperlukan jauh

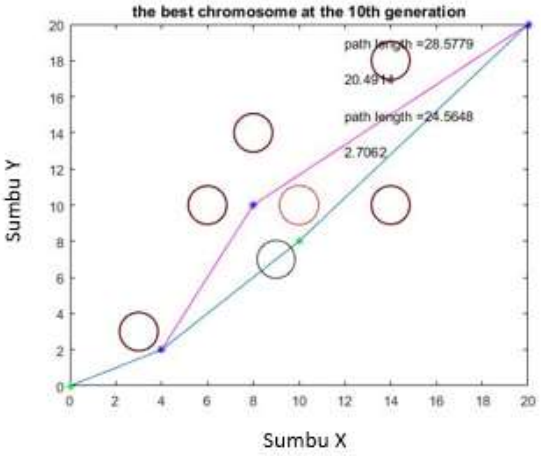
lebih cepat, namun tidak bisa diperoleh nilai yang lebih kecil dari kromosom =4 dalam pola tersebut. Semakin padat *obsctacle* dalam sebuah lingkungan, maka nilai kromosom yang diperlukan juga akan semakin besar.

## 4.2 Simulasi *Path Planning* dalam Lingkungan Dinamis

Untuk simulasi *path planning* dalam lingkungan dinamis, diperlukan juga simulasi *path following* dari *differential drive robot.* Oleh karena, untuk mensimulasikan robot yang dalam perjalanan menuju tujuan, namun tiba-tiba menemukan halangan sehingga harus mencari *path* baru yang bebas hambatan.

Dalam simulasi ini, digunakan program *path following controller example* yang mensimulasikan gerak sebuah *differential drive robot* dalam mengikuti sebuah set *waypoint.* Spesifikasi robot simulasi dan alur kerja simulasi terlampir di Bab 3.

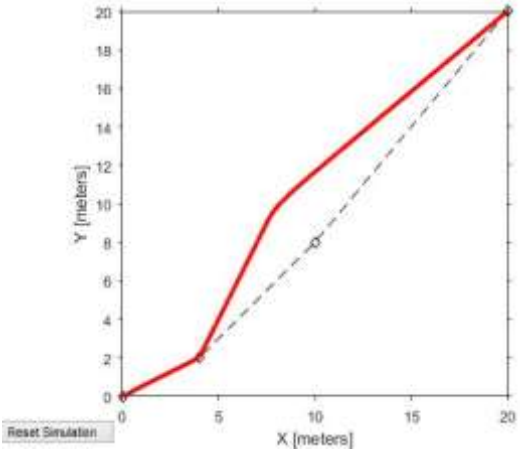
Pada percobaan dibawah, digunakan parameter sebagai berikut: ukuran populasi = 30, ukuran kromosom = 4, konfigurasi 6 *obstacle,* dilakukan dalam 20 generasi (pertama) dan 10 generasi (*re-planning*), nilai Pc = 0,9 dan Pm =1/3 dan menerapkan *elitism*.




Gambar 4. 6 *Dynamic Path Planning*

Pada Gambar 4.6, terdapat garis warna biru, yang merepresentasikan *path* awal, dan garis ungu yang merepresentasikan *path* yang dihasilkan setelah dilakukan *re-planning*. *Obstacle* yang berwarna merah merupakan *set obstacle* yang pertama, sedangkan yang berwarna hitam merupakan *set obstacle* yang kedua. Terlihat bahwa setelah mencapai *waypoint* pertama (4, 2), terdapat *obstacle* hitam di garis biru.

Dengan memanfaatkan kromosom terbaik yang diperoleh dari *path planning* statis yang pertama, menginisialisasi populasi tidak perlu dari angka yang benar-benar *random*. Selain itu, pencarian *path* baru hanya dilakukan hingga generasi ke-20. Dengan demikian, waktu komputasi yang tadinya 20,8 detik bisa menjadi 2,78 detik. *Trade-off* ini perlu dilakukan karena robot diusahakan agar tidak diam terlalu lama dalam mencari *path* yang baru.




Gambar 4. 7 Simulasi *Dynamic Path Planning*

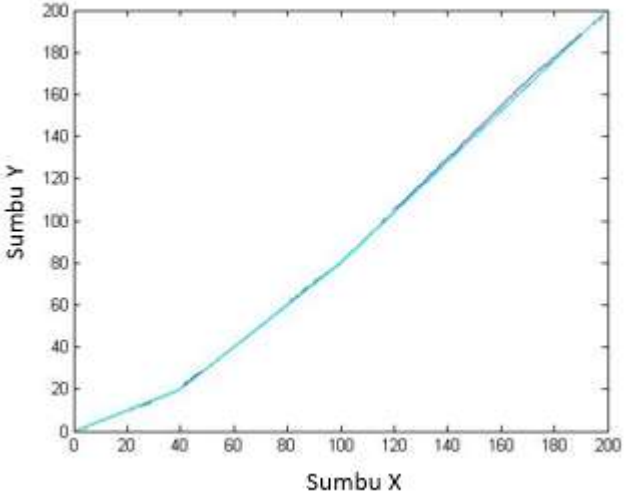
Pada simulasi *dynamic path planning*, digunakan parameter seperti yang tercantum pada Tabel 3.1. Dengan menyambungkan program yang telah dirancang ke simulator *path following* yang dibuat oleh MATLAB, dapat dilakukan validasi kapabilitas algoritma yang telah dirancang untuk merencanakan ulang *path* jika diperlukan.

## 4.3 Implementasi menggunakan Qbot

Menggunakan model Simulink dan Algoritma yang telah dirancang di Bab 3, implementasi menggunakan Qbot bisa dilakukan. Algoritma *path planning* akan menghasilkan dua set *waypoint* yang kemudian dimasukkan ke program lain lagi untuk menerjemahkan ke model Simulink yang akan berkomunikasi dengan *mobile robot*. Tujuan utama dari implementasi ini adalah membuktikan bahwa hasil waypoint yang diperoleh dari simulasi bisa diterapkan dalam sebuah robot asli. Selain itu, tes ini juga digunakan untuk mengetes apakah robot akan menyinggung *obstacle* meskipun pada bagian *path planning* telah dimasukkan *function* yang bertujuan untuk memberi kompensasi sesuai dimensi robot agar tidak terjadi tabrakan.

### 4.3.1 *Global Path Planning*

Dalam sub bab ini, hasil yang diperoleh dalam proses *path planning* statis akan dibawa ke ranah implementasi. Pada Gambar 4.8, terdapat perbandingan gerak sebuah robot (biru tua) dan *path* yang telah direncanakan. *Path* tersebut dihasilkan dengan menjalankan program *path planning* statis, namun dengan skala sesuai map yang baru. Terlihat bahwa meskipun tidak menggunakan kontroler, robot mampu mengikuti rencana.




*Gambar 4. 8 Trajektori Robot dan Rencana Jalur*

### 4.3.2 *Dynamic Path Planning*

Dalam sub bab ini, hasil yang diperoleh adalah robot mampu bergerak mengikuti *path* seperti dalam tahap *global path planning,* namun pada saat muncul sebuah *obstacle* dinamis yang berjarak 20-150 cm, robot mampu berhenti, dan kemudian merencanakan kembali *path agar* bisa mencapai tujuan.

*Gambar 4. 9 Dynamic Path Planning*

*Halaman ini sengaja dikosongkan*

# BAB 5
# PENUTUP

Dari perancangan dan analisa yang telah dilakukan, dapat dirangkum dalam sebuah kesimpulan. Untuk kekurangan dan kendala yang dihadapi, dituliskan dalam bagian saran, untuk membantu penelitian selanjutnya.

## 5.1 Kesimpulan

Setelah melakukan serangkaian simulasi mengenai *dynamic path planning* berbasis algoritma genetika, dapat disimpulkan bahwa algoritma ini mampu melakukan navigasi dalam lingkungan dengan *obstacle* statis dan dinamis. Untuk membantu mempercepat algoritma dalam proses mencari solusi terbaik (dalam kasus *dynamic path planning*), kromosom yang telah diperoleh pada saat menyelesaikan permasalahan *path planning* statis digunakan ulang saat membangkitkan populasi awal.

Tujuan implementasi menggunakan sebuah robot (Qbot) telah membuktikan bahwa algoritma ini juga bisa diterapkan untuk *path planning* yang mungkin bisa diimplementasikan dalam merencanakan gerak robot. Dalam lingkungan statis, *path* terpendek yang ditemukan dalam 20,4914 detik adalah 28,5779 satuan panjang. Untuk lingkungan dinamis, proses *re-planning* yang dilakukan dalam 2,7062 detik mampu menemukan sebuah *path* baru yang tidak menabrak *obstacle*.

## 5.2 Saran

Setelah melakukan penelitian ini, untuk pengembangan penelitian, disarankan beberapa hal, antara lain:

a. Perbandingan dan penggabungan dengan metode lain sehingga bisa diperoleh solusi yang lebih efisien baik dari segi waktu, maupun komputasi.
b. Penggunaan kontroler untuk memastikan *path following* yang lebih baik.
c. Penerapan dan pengembangan algoritma agar bisa diterapkan dalam sebuah sistem *multi-agent* untuk bisa pergi ke sebuah titik tanpa menabrak suatu *obstacle.*

*Halaman ini sengaja dikosongkan*

# DAFTAR PUSTAKA


[1] V. Ganapathy, S. C. Yun and S. Parasuraman, "Dynamic Path Planning Algorithm in Mobile Robot Navigation," in *2011 IEEE Symposium on Industrial Electronics and Applications*, Langkawi, 2011.

[2] I. Engedry and H. Gabor, "Artificial Neural Network based Mobile Robot Navigation," in *IEEE International Symposium on Intelligent Signal Processing*, Budapest, 2009.

[3] T. Lehtla, *Introduction to Robotics*, Tallinn: Department of Electrical Drives and Power Electronics, 2008.

[4] A. Basuki, *Algoritma Genetika,* Surabaya: PENS-ITS, 2003.

[5] D. E. Goldberg, *Genetic Algorithms in Search, Optimization, and Machine Learning*, Massachusets: Addison-Wesley, 1989.

[6] T. Pencheva, K. Atanassov and A. Shannon, "Modelling of a Stochastic Universal Sampling Selection Operator in Genetic Algorithms Using Generalized Nets," in *Tenth International Workshop on Generalized Nets*, Sofia, 2009.

[7] G. Dudek and M. Jenkin, *Computational Principles of Mobile Robotics*, New York: Cambridge University Press, 2010.

[8] …, "*User Manual Quanser Qbot,"* Quanser, Ontario, 2012.

[9] E. W. Weisstein, "*Circle.*," MathWorld--A Wolfram Web Resource, <URL: http://mathworld.wolfram.com/Circle.html> Juni, 2017

[10] E. W. Weisstein, "*Circle-Line Intersection*," MathWorld--A Wolfram Web Resource , <URL:

http://mathworld.wolfram.com/Circle-LineIntersection.html> Juni, 2017

[11] …, "*Pure Pursuit Controller*," Mathworks, <URL: https://www.mathworks.com/help/robotics/ug/pure-pursuit-controller.html> Juni, 2017.

[12] …, "Qbot Experiment #2: Motion Planning," Quanser, Ontario, 2011.

[13] …, "GP2Y0A02YK0F," SHARP Corporation, Osaka, 2006.

# LAMPIRAN A

Berikut program MATLAB untuk melakukan *path planning* berbasis *genetic algorithm* dalam lingkungan yang statis.

```
function [Xx,Yy,poos]=ga_1()
clc
wp=3;
%% globalvariables
tic;
pop=30; %population size
vp=[1 2 3 4 5]; %number of via points / gene
length
rad=1; %obs radius
map_size=20;
sp= [0;0] %starting point
%obs=[10,10] ;
obs=[3,3;10,10;14,10;14,18;8,14;6,10]' %obs
position
%obs=[5,5;17,15;10,16]';
ep=[map_size; map_size] %end point

%% posible locations
pos=[];
for k=1:map_size+1
 for l=1:map_size+1
 pos_t=[l-1;k-1];
 pos=[pos pos_t];
 end
end

%% pastikan via points tidak pada sp/ep/obs
no_via_ind1 =
find((pos(1,:)==sp(1)&pos(2,:)==sp(2))|(pos(1,:)
==ep(1)&pos(2,:)==ep(2)));
no_via_ind2=[];
for i=1:size(obs,2)
```

```
 no_via_ind2=[no_via_ind2 find((pos(1,:)-
obs(1,i)).^2+(pos(2,:)-obs(2,i)).^2<=rad^2)];
%titik yg di dlm obs
end
no_via = union(no_via_ind1,no_via_ind2);
% sop=pos;
% sopp=sop;
% sop(:,no_via_ind1)=[];
pos(:,no_via)=[];

% figure(21)
% plot( pos(1,:), pos(2,:), 'ro' );
% title( 'possible via points' );

% figure(10)
% plot(sop(1,:), sop(2,:), 'bo');
% title('titik yang mungkin dilewati');

    poos=[pos;zeros(2,length(pos))];
    for u=1:length(pos)
    if abs(poos(1,u)-poos(2,u))<=rad+2
    poos(3,u)=5;
    else
    poos(3,u)=2;
    end
    end

    u1=sum(poos(3,:)==5);
    u2=sum(poos(3,:)==2);

    for j=1:length(pos)
    if poos(3,j)==5
    poos(4,j)=0.55/u1;
    else
    poos(4,j)=0.45/u2;
    end
    end
```

```
 figure(22)
 for ii=1:length(poos)
     if poos(3,ii)==5
         plot(poos(1,ii), poos(2,ii), 'ro')
         hold on
     else
         plot(poos(1,ii), poos(2,ii), 'bo')
         hold on
     end
 end

[xch,ych]=generate_chr(pop,vp(wp),pos,sp,obs,ep,
rad,poos); %generate chromosome yang mungkin
 ch_dist=[];
 for i=2:vp(wp)+2
 ch_dist(:,i-1)= sqrt((xch(:,i-1)-
xch(:,i)).^2+(ych(:,i-1)-ych(:,i)).^2);
 end
 ch_fitness = sum(ch_dist')';
 best_ch_ind= find(ch_fitness ==
min(ch_fitness));

 ch_order = [[1:length(ch_fitness)']'
ch_fitness];
 ch_order = sortrows(ch_order,2); %re-oder
chromosomes menurut fitness

 konv_fitness=[];
%% iteration
for generation = 1:100
    Pc=0.9;
    Pm=0.35;
    elite= ch_order(1,:);
    xe=xch(elite(:,1),:);
    ye=ych(elite(:,1),:);
    order=ch_order(2:end,:);
    bit=round(rand*(vp(wp)-1));
    [xchild1,ychild1] =
cross(order,xch,ych,Pc,bit,vp(wp));
```

```
    chn_size=0;
    chn_size= pop-size(xchild1,1);
    [xchld2,ychld2]=
generate_chr(chn_size,vp(wp),pos,sp,obs,ep,rad,p
oos);
%buat jumlah current population = starting
population
    xchild1=[xchild1;xchld2];
    ychild1=[ychild1;ychld2];

[xchild2,ychild2]=mutate(xchild1,ychild1,pos,Pm,
obs,rad,poos);

 %%%update chromosome
 xchn= xchild2;
 ychn= ychild2;

 %jika chromosome kurang, buat baru
 chn_size=0;
 chn_size= pop-size(xchn,1);
 [xchn2,ychn2]=
generate_chr(chn_size,vp(wp),pos,sp,obs,ep,rad,p
oos);

 xch_t= [xchn ; xchn2];
 xch(2:end,:)= xch_t(1:pop-1,:);
 xch(1,:)=xe;
 ych_t= [ychn ; ychn2];
 ych(2:end,:)= ych_t(1:pop-1,:);
 ych(1,:)=ye;

 %%%chromosome re-evaluation
 ch_dist=[];
 for i=2:vp(wp)+2
 ch_dist(:,i-1)=sqrt((xch(:,i-1)-
xch(:,i)).^2+(ych(:,i-1)-ych(:,i)).^2);
 end
```

```
 ch_fitness = sum(ch_dist')';
 best_ch_ind=find(ch_fitness ==
min(ch_fitness));
 lols=min(ch_fitness);
 konv_fitness=[konv_fitness lols];

 ch_order = [[1:length(ch_fitness)']'
ch_fitness] ;
 ch_order = sortrows(ch_order,2) ; %reoder the
chromosomes menurut fitness
 %% hasil pada generasi ke x
 if generation==2
 xch2=xch;
 ych2=ych;
 order2=ch_order;
 best2=best_ch_ind;
 figure(1)
 plot(xch2(best2(1),:),ych2(best2(1),:),'g*')
 hold on
 line(xch2(best2(1),:),ych2(best2(1),:))

plot(obs(1,:),obs(2,:),'ro','markersize',rad*28)
 title('the best chromosome at the 2th
generation')
 text(12,19,['path length
=',num2str(order2(1,2))])
 text(12,17,num2str(toc))
 hold off
 print('-dtiff',['ob3_10_',num2str(wp)])
 end

 if generation==5
 xch5=xch;
 ych5=ych;
 order5=ch_order;
 best5=best_ch_ind;
 figure(2)
 plot(xch5(best5(1),:),ych5(best5(1),:),'g*')
```

```
 hold on
 line(xch5(best5(1),:),ych5(best5(1),:))

plot(obs(1,:),obs(2,:),'ro','markersize',rad*28)
 title('the best chromosome at the 5th
generation')
 text(12,19,['path length
=',num2str(order5(1,2))])
 text(12,17,num2str(toc))
 hold off
 print('-dtiff',['ob3_10_',num2str(wp)])
 end

 if generation==10
 xch10=xch;
 ych10=ych;
 order10=ch_order;
 best10=best_ch_ind;
 figure(3)

plot(xch10(best10(1),:),ych10(best10(1),:),'g*')
 hold on
 line(xch10(best10(1),:),ych10(best10(1),:))

plot(obs(1,:),obs(2,:),'ro','markersize',rad*28)
 title('the best chromosome at the 10th
generation')
 text(12,19,['path length
=',num2str(order10(1,2))])
 text(12,17,num2str(toc))
 hold off
 print('-dtiff',['ob3_10_',num2str(wp)])
 end

 if generation==20
 xch20=xch;
 ych20=ych;
 order20=ch_order;
 best20=best_ch_ind;
```

```
 figure(4)

plot(xch20(best20(1),:),ych20(best20(1),:),'g*')
 hold on
 line(xch20(best20(1),:),ych20(best20(1),:))

plot(obs(1,:),obs(2,:),'ro','markersize',rad*28)
 title('the best chromosome at the 20th
generation')
 text(12,19,['path length
=',num2str(order20(1,2))])
 text(12,17,num2str(toc))
 hold off
 print('-dtiff',['ob3_10_',num2str(wp)])
 end

 if generation==50
 xch50=xch;
 ych50=ych;
 order50=ch_order;
 best50=best_ch_ind;
 figure(5)

plot(xch50(best50(1),:),ych50(best50(1),:),'g*')
 hold on
 line(xch50(best50(1),:),ych50(best50(1),:))

plot(obs(1,:),obs(2,:),'ro','markersize',rad*28)
 hold off
 title('the best chromosome at the 50th
generation')
 text(12,19,['path length
=',num2str(order50(1,2))])
 text(12,17,num2str(toc))
 print -dtiff fig_50
 end

 if generation==100
 xch100=xch;
```

```
 ych100=ych;
 order100=ch_order;
 best100=best_ch_ind;
 figure(6)

plot(xch100(best100(1),:),ych100(best100(1),:),'
g*')
 hold on
 line(xch100(best100(1),:),ych100(best100(1),:))

plot(obs(1,:),obs(2,:),'ro','markersize',rad*28)
 hold off
 title('the best chromosome at the 100th
generation')
 text(12,19,['path length
=',num2str(order100(1,2))])
 text(12,17,num2str(toc))
 print -dtiff fig_50
 end
end
toc;
figure(15)
plot(1:generation,konv_fitness)
title('min. distance found')
xlabel('generation')
ylabel('distance')
 Xx=xch(best_ch_ind(1),:);
 Yy=ych(best_ch_ind(1),:);
end

function [Xch,Ych] =
generate_chr(n,vp,pos,sp,obs,ep,rad,poos)
i=1;
Xch=zeros(n,vp+2);
Ych=zeros(n,vp+2);
    %a=true;
 while i < n+1
```

```
%chs_ind=datasample(1:409,2,'Replace',a,'Weights
',poos(3,:));

chs_ind=randsample(1:length(pos),vp,true,poos(3,
:)); %pop size dari possible comb.
 ch_t=[sp pos(:,chs_ind) ep];

[test_result]=test_chr(ch_t(1,:),ch_t(2,:),obs,r
ad);
 if test_result == 1
 Xch(i,:)= ch_t(1,:);
 Ych(i,:)= ch_t(2,:);
 i=i+1;
    %a=false;
 end
 end
end

% function [Xch,Ych] =
generate_chr(n,vp,pos,sp,obs,ep,rad)
% i=1;
% Xch=zeros(n,vp+2);
% Ych=zeros(n,vp+2);
%  while i < n+1
%  chs_ind=ceil(rand(1,vp)*length(pos)); %%%pop
size dari possible comb.
%  ch_t=[sp pos(:,chs_ind) ep];
%
%
[test_result]=test_chr(ch_t(1,:),ch_t(2,:),obs,r
ad);
%
%  if test_result == 1
%  Xch (i,:)= ch_t(1,:);
%  Ych (i,:)= ch_t(2,:);
%  i=i+1;
%  end
%  end
```

```
% end

function [test_result]=test_chr(xc,yc,obs,rad)
clear test_result;
rad=rad+0.25;
v_fit=zeros(size(xc,2),length(obs),size(xc,1));
for p=1: size(xc,1)
 for q=2: size(xc,2)
 for r=1: length(obs)
 x2= xc(p,q);
 y2=yc(p,q);
 x1=xc(p,q-1);
 y1=yc(p,q-1);
 x0= obs(1,r);
 y0=obs(2,r);

 if x1==x2  %%cari xsol ysol dulu
 pol1=1;
 pol2= -2*y0;
 pol3= y0^2+x1^2-2*x0*x1+x0^2-rad^2;

 ysol= roots([pol1 pol2 pol3]);
 xsol= [x1;x2];

 %%%% else represent in polynoimal,descending
order of X
 else
 m=(y2-y1)/(x2-x1);
 c=y1-m*x1;
 pol1= m^2+1;
 pol2= (2*m*c-2*y0*m-2*x0);
 pol3= c^2-2*y0*c+y0^2+x0^2-rad^2;

 xsol= roots([pol1 pol2 pol3]);
 ysol= m*xsol + c;
 end
 sol= [xsol,ysol];
 sol_d = dist([x1,y1],[xsol';ysol']);
 obs_d = dist([x1,y1],[x0;y0]);
```

```
 if isreal(sol) && (obs_d > min(sol_d) && obs_d
< max(sol_d))
 v_fit(q,r,p ) = 1;
 end
 end

 if sum(sum(v_fit(:,:,p))) <1
 test_result(p)=1;
 else
 test_result(p)=0;
 end
 end
end
end

%% 1 point crossover pada posisi bit ke-1
function [xchild,ychild] =
cross(order,xch,ych,Pc,bit,vp)
if bit < 0 || bit > 5
 error('choose diff bit number')
 return
end
ch_fitness_t = order(:,2);
ch_fitness= 1./ch_fitness_t;
p= ch_fitness/(sum(ch_fitness) ) ;
xp= xch(order(:,1),:);
yp= ych(order(:,1),:);
for i=1:length(p)
 q(i)=sum(p(1:i));
end
%%chromosome clone
roulette=rand(1,length(p));
for i=1:length(p)
qk_t=find(q>roulette(i));
qk(i)=qk_t(1);
end
xp=xp(qk,:);
yp=yp(qk,:);
```

```
%%chromosome crossover
roulette=rand(1,length(p));
vc=find(roulette<Pc);
xparent = xp;
xhead= xparent(vc,1:vp-bit);
xtail=xparent(vc,vp-bit+1:end);
yparent = yp ;
yhead= yparent(vc, 1:vp-bit);
ytail=yparent(vc,vp-bit+1:end);
xchild=xparent;
ychild=yparent;
xtail= flipud(xtail);
ytail= flipud(ytail);
xchild(vc,:)= [xhead xtail];
ychild(vc,:)= [yhead ytail];
end

function [xchild,ychild] =
mutate(xp,yp,pos,Pm,obs,rad,poos)
xhead= xp(:,1);
xbody= xp(:,2:end-1);
xtail= xp(:,end);
yhead= yp(:,1);
ybody= yp(:,2:end-1);
ytail= yp(:,end);
%%mutasi dilakukan pada bagian tengah chromosome
Ppm=rand(size(xbody,1),size(xbody,2));
m_ind=find(Ppm < Pm);

child_ind=randsample(1:length(pos),length(m_ind)
,true,poos(3,:));
xbody(m_ind)= pos(1,child_ind);
ybody(m_ind)= pos(2,child_ind);
xchild= [xhead xbody xtail];
ychild= [yhead ybody ytail];
[test_result]=test_chr(xchild,ychild,obs,rad);
xchild(find(test_result == 0),:)=[];
ychild(find(test_result == 0),:)=[];
end
```

# LAMPIRAN B

Berikut kode program Path Following buatan MATLAB

```
%% Path Following for a Differential Drive Robot
%% Introduction
% This example demonstrates how to control a
robot to follow a desired path
% using a Robot Simulator. The example uses the
Pure Pursuit path following
% controller to drive a simulated robot along a
predetermined path. A desired path is a
% set of waypoints defined explicitly or
computed using a path planner (refer to
% <docid:robotics_examples.example-
PathPlanningExample>). The Pure Pursuit
% path following controller for a simulated
differential drive robot is created and
% computes the control commands to follow a
given path. The computed control commands are
% used to drive the simulated robot along the
desired trajectory to
% follow the desired path based on the Pure
Pursuit controller.

% Copyright 2014-2015 The MathWorks, Inc.
%% Define waypoints
% Define a set of waypoints for the desired path
for the robot

path = [2.00    1.00;
        1.25    1.75;
        5.25    8.25;
        7.25    8.75;
        11.75   10.75;
        12.00   10.00];

% Set the current location and the goal location
of the robot as defined by the path
```

```
robotCurrentLocation = path(1,:);
robotGoal = path(end,:);

%%
% Assume an initial robot orientation (the robot
orientation is the angle
% between the robot heading and the positive X-
axis, measured
% counterclockwise).
initialOrientation = 0;

%%
% Define the current pose for the robot [x y
theta]
robotCurrentPose = [robotCurrentLocation
initialOrientation];

%% Initialize the robot simulator
% A simple robot simulator is used in this
example that updates and
% returns the pose of the differential drive
robot for given control inputs. An external
% simulator or a physical robot will require a
localization mechanism to
% provide an updated pose of the robot.

%%
% Initialize the robot simulator and assign an
initial pose. The simulated
% robot has kinematic equations for the motion
of a two-wheeled differential drive robot.
% The inputs to this simulated robot are linear
and angular velocities. It also has plotting
% capabilities to display the robot's current
location and draw the
% trajectory of the robot.

robotRadius = 0.4;
```

```
robot =
ExampleHelperRobotSimulator('emptyMap',2);
robot.enableLaser(false);
robot.setRobotSize(robotRadius);
robot.showTrajectory(true);
robot.setRobotPose(robotCurrentPose);

%%
% Visualize the desired path
plot(path(:,1), path(:,2),'k--d')
xlim([0 13])
ylim([0 13])

%%
% <<path_robot_simulator_part1.png>>
%

%% Define the path following controller
% Based on the path defined above and a robot
motion model, you need a path
% following controller to drive the robot along
the path. Create the path
% following controller using the
|<docid:robotics_ref.buoofp1-1
robotics.PurePursuit>|  object.
controller = robotics.PurePursuit

%%
% Use the path defined above to set the desired
waypoints for the
% controller
controller.Waypoints = path;

%%
% Set the path following controller parameters.
The desired linear
% velocity is set to 0.3 meters/second for this
example.
controller.DesiredLinearVelocity = 0.3;
```

```
%%
% The maximum angular velocity acts as a
saturation limit for rotational velocity, which
is
% set at 2 radians/second for this example.
controller.MaxAngularVelocity = 2;

%%
% As a general rule, the lookahead distance
should be larger than the desired
% linear velocity for a smooth path. The robot
might cut corners when the
% lookahead distance is large. In contrast, a
small lookahead distance can
% result in an unstable path following behavior.
A value of 0.5 m was chosen
% for this example.
controller.LookaheadDistance = 0.5;

%% Using the path following controller, drive
the robot over the desired waypoints
% The path following controller provides input
control signals for the
% robot, which the robot uses to drive itself
along the desired path.
%
% Define a goal radius, which is the desired
distance threshold
% between the robot's final location and the
goal location. Once the robot is
% within this distance from the goal, it will
stop. Also, compute the current
% distance between the robot location and
% the goal location. This distance is
continuously checked against the goal
% radius and the robot stops when this distance
is less than the goal radius.
%
```

```
% Note that too small value of the goal radius
may cause the robot to miss
% the goal, which may result in an unexpected
behavior near the goal.
goalRadius = 0.1;
distanceToGoal = norm(robotCurrentLocation -
robotGoal);

%%
% The |<docid:robotics_ref.buopp2z-1 step>|
function computes control commands for the
robot.
% Drive the robot using these control commands
until it reaches within the
% goal radius. If you are using an external
simulator or a physical robot,
% then the controller outputs should be applied
to the robot and a localization
% system may be required to update the pose of
the robot. The controller runs at 10 Hz.
controlRate = robotics.Rate(10);
while( distanceToGoal > goalRadius )

    % Compute the controller outputs, i.e., the
inputs to the robot
    [v, omega] = step(controller,
robot.getRobotPose());

    % Simulate the robot using the controller
outputs.
    drive(robot, v, omega);

    % Extract current location information
([X,Y]) from the current pose of the
    % robot
    robotCurrentPose = robot.getRobotPose();

    % Re-compute the distance to the goal
```

```
    distanceToGoal = norm(robotCurrentPose(1:2)
- robotGoal);

    waitfor(controlRate);

end

%%
% <<path_completed_path_part1.png>>
%

%%
% The simulated robot has reached the goal
location using the path following
% controller along the desired path. Close
simulation.
delete(robot)
```

# LAMPIRAN C

Berikut isi dari blok Motion Planner

```
function [right_vel, left_vel, target_distance,
states] = ...
    motion_planner(target_list, robot_xyt,
vlimit, ...
    delta_t, dist_thr, ang_thr, method, step,
pre_states)

% Initialize output variables
right_vel = int16(0);
left_vel = int16(0);
states = pre_states;

rx = robot_xyt(1);
ry = robot_xyt(2);
rtheta = check_angle(robot_xyt(3));
target_xy = [rx ry];
[n xy] = size(target_list);
if n==2 && xy==1
    target_xy = [target_list(1,1)
target_list(2,1)];
    n = 1;
else
    for i=1:n
        if states(1) == i
            for j=1:xy
                target_xy(j) = target_list(i,
j);
            end
        end
    end
end

tx = target_xy(1);
ty = target_xy(2);
target_distance = find_dist(rx, ry, tx, ty);
```

```
if ((states(1) == n) && (target_distance <=
dist_thr)) %|| states(1) == -1
%          states(1) = -1;
        right_vel = int16(0);
        left_vel = int16(0);
%          target_distance = 0;
else
    if target_distance <= dist_thr
        states(1) = pre_states(1) + step;%1;
        if states(1) > n
            states(1) = n;
        end
    else
        [ang_to_tar]= find_theta(rx, ry, rtheta,
tx, ty);
        if method == 2
            [right_vel, left_vel] =
solve_inv_kin(target_distance, ...
                ang_to_tar, vlimit, delta_t);
        else
            [right_vel, left_vel] =
rotate_and_go(ang_to_tar, ...
                vlimit, ang_thr);
        end

    end
end

return;

%-----------------------------------------------
---------------------------
function [y] = check_angle(x)
y = x;
if x > pi
    y = x - 2*pi;
elseif x < -pi
    y = x + 2*pi;
```

```
end
return;
% ------------------------------------------------
--------------------------
function dist = find_dist(rx, ry, tx, ty)
dist = sqrt((rx-tx)^2 + (ry-ty)^2);
return;
% ------------------------------------------------
--------------------------
function [theta]= find_theta(rx, ry, rtheta, tx,
ty)

X = tx - rx;
Y = ty - ry;
theta = atan2(Y, X);
theta = check_angle(theta -
check_angle(rtheta));

return;
% ------------------------------------------------
--------------------------
function [vr, vl] = solve_inv_kin(dist, theta,
vlimit, delta_t)

d = 252.5;
vmax = vlimit(2);
wmax = (2*vmax)/d;
w = theta/delta_t;
w_sign = sign(w);
if abs(w) > wmax
    w = w_sign*wmax;
    vr = int16(round((d*w)/2));
    vl = int16(-vr);
else
    v = dist/delta_t;
    vr_tmp = (2*v + d*w)/2;
    vl_tmp = 2*v - vr_tmp;

    max_of_vrvl = abs(max(vr_tmp, vl_tmp));
```

```
    if max_of_vrvl > vmax
        vr_tmp = (vr_tmp/max_of_vrvl)*vmax;
        vl_tmp = (vl_tmp/max_of_vrvl)*vmax;
    end
    vr = int16(vr_tmp);
    vl = int16(vl_tmp);

end


return;
% ------------------------------------------------
---------------------------
function [vr, vl] = rotate_and_go(theta, vlimit,
ang_thr)

if abs(theta) > ang_thr
    if theta >= 0
        vr = int16(vlimit(1));
        vl = int16(-vlimit(1));
    else
        vr = int16(-vlimit(1));
        vl = int16(vlimit(1));
    end
else
        vr = int16(vlimit(2));
        vl = int16(vlimit(2));
end

return;
% ------------------------------------------------
---------------------------
```

Berikut isi dari blok for_kin dalam sub blok Qbot

```
function [x, y, theta, states] = for_kin(x0, y0,
theta0, dist, ang, pre_states)
% This block supports an embeddable subset of
the MATLAB language.
```

```
% See the help menu for details.

states = pre_states;

if states(1) == 0
    x = x0;
    y = y0;
    theta = check_angle(theta0);
    states(1) = 1;
else
    theta = check_angle(check_angle(states(4)) +
check_angle(ang));
    x = states(2) + dist*cos(theta);
    y = states(3) + dist*sin(theta);
end

states(2) = x;
states(3) = y;
states(4) = theta;

return;

function y = check_angle(x)
y = x;
if y > pi
    y = x - 2*pi;
elseif y < -pi
    y = x + 2*pi;
end
return;
```

*Halaman ini sengaja dikosongkan*

# RIWAYAT PENULIS

Jeffry Susanto lahir di Surabaya, 7 Agustus 1995. Penulis menamatkan pendidikan sekolah dasar di SDK St. Clara Surabaya, SMPK St. Clara Surabaya, dan SMAK St. Louis 1 Surabaya. Setelah itu, penulis melanjutkan kuliah S1 Teknik Elektro di Institut Teknologi Sepuluh Nopember dengan konsentrasi di Teknik Sistem Pengaturan. Pada bulan Juli 2017, penulis mengikuti proses ujian Tugas Akhir sebagai syarat untuk memperoleh gelar Sarjana dalam bidang Teknik Elektro.