BAB 5 IMPLEMENTASI SIMULATOR NAML

Bab ini akan menjelaskan tentang proses pengembangan simulator yang akan digunakan untuk pengujian algoritma yang telah dirancang sebelumnya. Proses pengembangan simulator ini sendiri akan dilakukan melalui beberapa tahap sesuai dengan masing-masing algoritma yang akan dilakukan pengujian. Proses pengembangan simulator ini sendiri dimulai dari tahap pemilihan teknologi yang akan digunakan, instalasi perangkat lunak yang dibutuhkan, perancangan simulator hingga implementasi kode program.

5.1 Tentang Simulator NAML

Setelah merancang algoritma yang akan digunakan untuk membangun jaringan komunikasi *multihop*, kegiatan yang selanjutnya dilakukan pada penelitian tugas akhir ini adalah pengembangan simulator yang akan digunakan sebagai alat pengujian algoritma yang dikembangkan agar kinerja dari masing-masing algoritma tersebut dapat dinilai secara objektif.

Kegiatan utama yang dilakukan pada proses implementasi simulator ini adalah proses koding algoritma ke dalam kode program yang sesuai hingga proses simulasi pembangunan jaringan komunikasi *multihop* ini dapat divisualisasikan dan dapat dilihat hasilnya. Sebelum dijelaskan mengenai proses koding, akan diperkenalkan dulu simulator yang dikembangkan tersebut.

Penulis menamakan simulator yang dikembangkan ini dengan nama NAML. Nama ini berasal dari singkatan "*From Natural Ants Into Mobile Labour*" atau jika diterjemahkan ke dalam bahasa Indonesia akan berarti "Dari Semut di Alam Hingga Menjadi Tenaga Bergerak". Penamaan ini disebabkan karena sebelumnya salah satu tujuan pengembangan simulator ini adalah untuk melakukan visualisasi terhadap salah satu algoritma yang baik untuk digunakan untuk permasalahan pencarian path terpendek pada suatu konfigurasi graph tertentu. Algoritma yang meniru perilaku semut di alam ini tadinya diharapkan dapat diterapkan pada penelitian tugas akhir ini. Akan tetapi, karena beberapa kendala yang kemudian

ditemukan pada tahapan penelitian, algoritma ini tidak dapat dikembangkan untuk menemukan solusi permasalahan dengan baik. Selain itu, nama NAML juga berasal dari bahasa Arab yang berarti "Semut".

5.2 Pemilihan Implementasi Teknologi

Salah satu hal yang pertama kali dilakukan pada proses implementasi rancangan algoritma ke dalam simulasi adalah pemilihan teknologi yang akan digunakan untuk membuat simulator. Berikut akan dijelaskan mengenai implementasi teknologi yang dipilih untuk pengembangan program simulator NAML ini.

5.2.1 Bahasa Pemrograman Java

Teknologi ini merupakan teknologi utama yang akan digunakan sebagai bahasa pemrograman dalam rangka pembuatan kode program untuk simulator NAML. Java adalah bahasa pemrograman yang berorientasi objek atau dikenal dengan istilah *object-oriented programming* sehingga pengembangan program simulator dapat dilakukan dengan cara merepresentasikan objek-objek yang dibutuhkan dalam simulasi ke dalam class-class yang sesuai. Pada pengembangan simulator NAML dalam penelitian tugas akhir ini, jenis teknologi Java yang digunakan adalah Java J2SE 6 dan dikembangkan dengan tools JDK 1.6.0 Untuk dapat menggunakan teknologi ini, dapat diunduh dari website http://java.sun.com.

5.2.2 Animasi OpenGL

Pemilihan teknologi ini dimaksudkan untuk melakukan visualisasi pergerakan robot dalam bentuk animasi secara grafis. Teknologi ini sendiri sebenarnya dikembangkan untuk digunakan dalam bahasa C atau C++. Akan tetapi dengan menggunakan *library* java yang bernama JOGL (Java OpenGL), teknologi ini kemudian dapat diterapkan pada bahasa pemrograman Java.

5.3 Tahap Perancangan Simulator

Setelah menentukan pemilihan implementasi teknologi yang digunakan pada pengembangan simulator NAML, kegiatan yang berikutnya dilaksanakan adalah perancangan class-class yang dibutuhkan dalam simulasi, serta setup beberapa kebutuhan lain yang diperlukan oleh program simulator.

5.3.1 Perancangan Class

Berikut adalah beberapa class yang terdapat pada program simulator NAML.

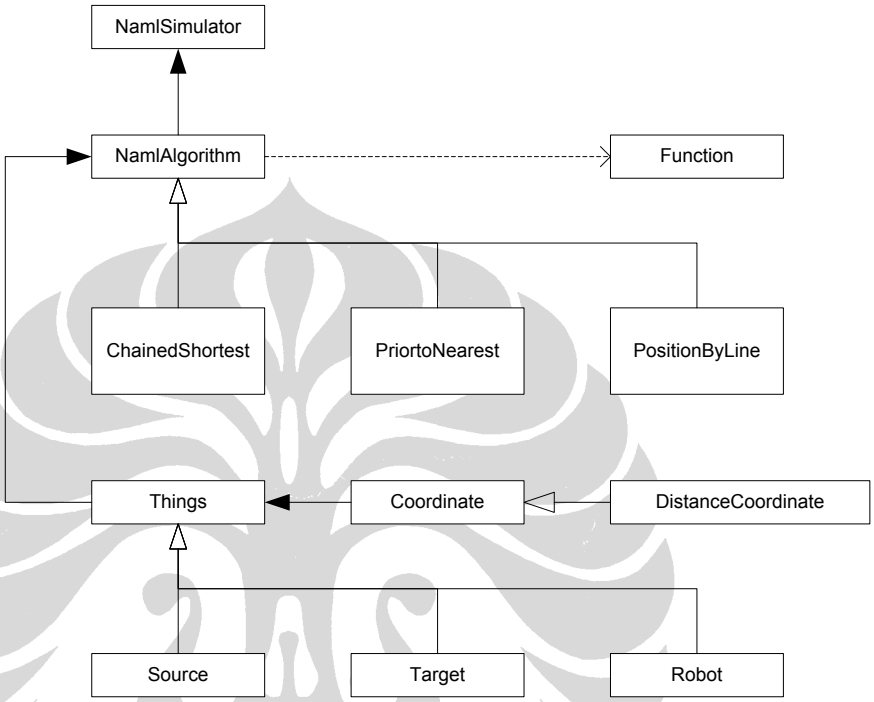
1. *NamlSimulator,* merupakan class GUI sebagai visualisasi simulator kepada pengguna. Class ini kemudian melakukan pemanggilan terhadap algoritma yang digunakan agar simulasi dapat berjalan. Class ini merupakan sub-class dari class *JFrame* yang sudah disediakan oleh Java.
2. *NamlAlgorithm,* merupakan interface yang harus diimplementasikan oleh algoritma yang akan digunakan di dalam simulasi. Interface ini memiliki dua method yang harus diterapkan oleh class yang menjadi sub-classnya, yaitu *initialization* dan *iteration*. Method *initialization* merupakan method yang dilakukan ketika simulasi berada pada fase inisialisasi, sedangkan method *iteration* dilakukan ketika simulasi berada pada fase iterasi.
3. *ChainedShortest,* merupakan class yang merepresentasikan penerapan algoritma *Chained Shortest Distance* dan *Optimized Chained Shortest* dalam simulasi. Class ini merupakan sub-class yang menerapkan interface NamlAlgorithm.
4. *PriorToNearest,* merupakan class yang merepresentasikan penerapan algoritma *Prior to the Nearest* dalam simulasi. Class ini merupakan sub-class yang menerapkan interface NamlAlgorithm.
5. *PositionByLine,* merupakan class yang merepresentasikan penerapan algoritma *Position By Line* dalam simulasi. Class ini merupakan sub-class yang menerapkan interface NamlAlgorithm.

6. *Things,* merupakan class yang merepresentasikan objek yang terlibat dalam kegiatan simulasi. Adapun pembuatan class ini sebagai superclass dari objek robot dan objek target dimaksudkan untuk mencegah redundansi pendefinisian informasi yang sama-sama dimiliki oleh kedua objek. Selain itu, juga untuk memungkinkan penggunaan fitur *polymorphism*.
7. *Robot*, merupakan class yang merepresentasikan objek robot di dalam simulasi. Pada class ini ditentukan sifat dan karakteristik yang digunakan pada masing-masing algoritma. Pada setiap algoritma, spesifikasi robot yang digunakan tidak selalu sama sehingga class *Robot* ini juga memiliki representasi yang berbeda tergantung algoritma yang digunakan.
8. *Target*, merupakan class yang merepresentasikan objek target yang akan dihubungkan melalui konfigurasi jaringan komunikasi *multihop*.
9. *Source* atau *Sink,* merupakan sub-class dari class *target*. Pada dasarnya, class ini juga untuk merepresentasikan objek target yang akan dihubungkan oleh konfigurasi jaringan komunikasi *multihop* di dalam simulasi. Adanya pembedaan class *source* ini dimaksudkan untuk membedakan jenis target pada algoritma tertentu, contohnya untuk membedakan perangkat pengawas dari objek yang diawasi pada algoritma *Prior to the Nearest*.
10. *Coordinate*, merupakan class yang merepresentasikan posisi objek di dalam ruang kerja simulasi berupa pasangan tiga angka, yaitu koordinat sumbu x, y dan z pada ruang Euclid berdimensi tiga.
11. *DistanceCoordinate,* merupakan sub-class dari class *coordinate* dengan tambahan informasi jarak. Class ini merepresentasikan posisi koordinat dengan suatu jarak tertentu dari suatu posisi koordinat yang lain.
12. *Function*, merupakan class yang berisikan fungsi komputasi matematik, seperti menghitung jarak antara dua titik, menghitung sudut, dan sebagainya.

Keseluruhan class ini kemudian dibuat dalam beberapa file .java yang disimpan pada suatu direktori yang sama. Untuk dapat menggunakan program, dilakukan

dengan cara menjalankan class *NamlSimulator* yang merupakan class GUI dari program simulator NAML.

Hubungan masing-masing class yang telah disebutkan di atas di dalam program simulasi dapat dilihat pada diagram pada gambar 5.1 berikut di bawah ini.



Gambar 5.1 - Diagram Hubungan Antar Class

Dari diagram tersebut dapat dilihat bahwa class *NamlSimulator* melakukan instansiasi objek dari interface *NamlAlgorithm* yang diimplementasikan oleh tiga class lainnya, yaitu *ChainedShortest*, *PriorToNearest*, serta *PositionByLine*. Class *NamlAlgorithm* ini melakukan instansiasi objek dari class *Things* yang merupakan superclass dari class *Source*, *Target*, serta *Robot*. Class *NamlAlgorithm* juga memanggil class *Function*. Selain itu, instansiasi objek dari class *Coordinate* terjadi pada class *Things*, di mana class *Coordinate* ini memiliki satu sub class lainnya, yaitu class *DistanceCoordinate*. Keseluruhan hubungan class-class inilah yang kemudian menentukan jalannya simulasi pada program simulator NAML.

5.3.2 Classpath dan Setup Program

Pada implementasi program simulator NAML ini digunakan beberapa *library* tambahan pada teknologi java, seperti JOGL dan JPL sehingga diperlukan pengesetan classpath agar java dapat mengenali *library* tambahan tersebut.

Untuk memudahkan pekerjaan, maka pengesetan classpath ini kemudian dilakukan melalui program *batch* untuk melakukan kompilasi dan penjalanan program simulator. Berikut adalah kedua program *batch* yang dibuat tersebut.

1. compile.bat, merupakan program *batch* untuk fungsi kompilasi. Adapun program ini berisikan kode sebagai berikut.

```
@echo off
echo Compiling %1
javac -classpath "D:\Classpath\JOGL-WIN32\jogl.jar;
        D:\Classpath\JPL\jpl.jar." %1
echo Finished.
```

2. run.bat, merupakan program *batch* untuk fungsi menjalankan program. Adapun program ini berisikan kode sebagai berikut.

```
@echo off
echo Compiling %1
javac -classpath "D:\Classpath\JOGL-WIN32\jogl.jar;
D:\Classpath\JPL\jpl.jar." %1
echo Finished.
```

Selain itu, juga terdapat beberapa file teks yang berfungsi untuk menyimpan informasi posisi awal masing-masing objek di dalam simulasi, seperti:

- *robot.txt*, berfungsi untuk menyimpan posisi awal robot-robot otonom.
- *source.txt*, berfungsi untuk menyimpan posisi awal target-target yang berjenis *sink*.

- *target.txt*, berfungsi untuk menyimpan posisi awal target-target yang akan dihubungkan dalam konfigurasi jaringan komunikasi *multihop* yang dicari selama proses simulasi.

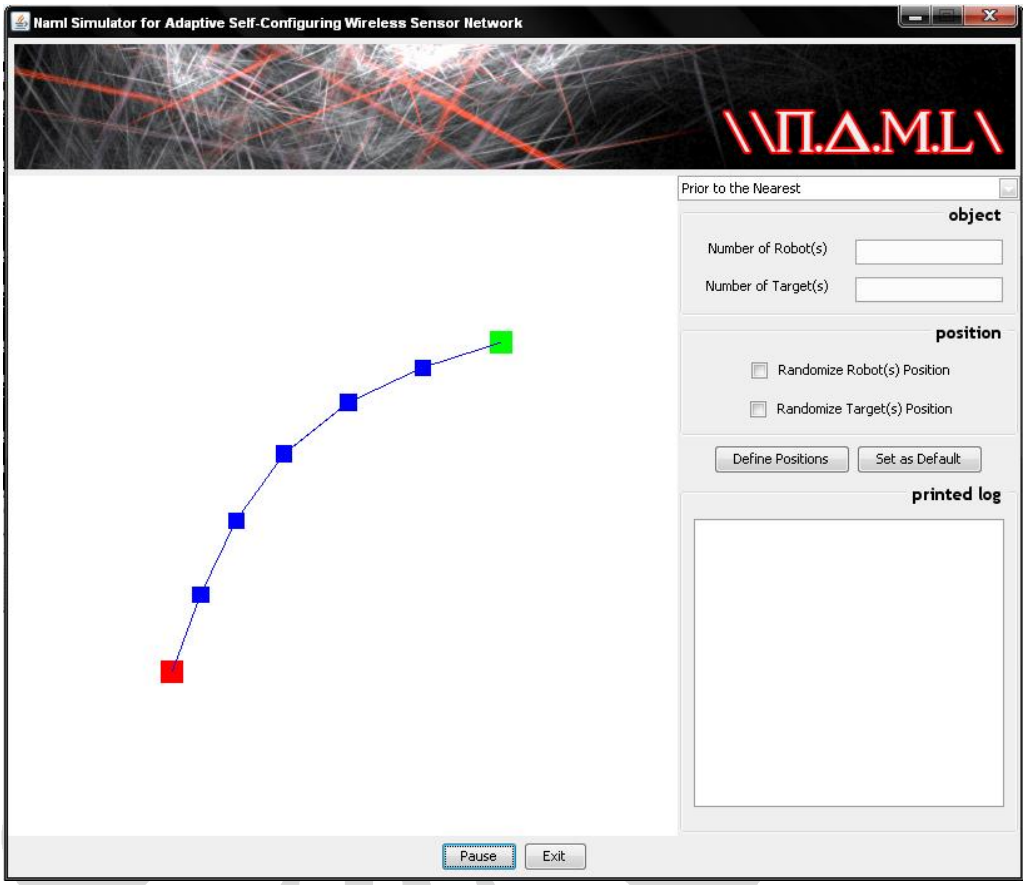
Pada masing-masing file ini, disimpan *n* baris pasangan dua angka *x* dan *y* bertipe *floating point* positif atau negatif yang akan menentukan posisi *n* buah objek pada posisi koordinat P(x, y) pada saat simulasi dimulai. Perlu diingat bahwa meskipun pada simulasi ini pergerakan robot hanya terjadi pada sumbu *x* dan sumbu *y*, posisi robot dan target sebenarnya direpresentasikan pada ruang Euclid berdimensi tiga. Adapun posisi semua objek dalam simulasi pada sumbu *z* diasumsikan sama, yaitu pada koordinat *z*=0. Jadi, penulisan notasi P(x, y) pada laporan tugas ini berarti sama dengan posisi koordinat P(x, y, 0).

5.4 Tahap Implementasi Simulator

Setelah merancang class-class yang akan digunakan dalam program simulator serta melakukan setup yang diperlukan, tahap yang selanjutnya dilakukan adalah kegiatan implementasi kode program. Berikut akan dijelaskan proses koding yang dilakukan hingga akhirnya diperoleh kode program yang sesuai.

5.4.1 Class NamlSimulator

Class ini merupakan class GUI yang akan menentukan visualisasi tampilan program simulator kepada pengguna. Untuk itu, perlu ditentukan terlebih dahulu bentuk tampilan program yang diinginkan, yaitu seperti yang ditunjukkan oleh gambar 5.2 berikut di bawah ini. Bagian kiri tampilan program merupakan layar yang menunjukkan proses berlangsungnya simulasi, sedangkan bagian kanan tampilan program digunakan untuk mengatur beberapa parameter simulasi.



Gambar 5.2 - Tampilan Program Simulator NAML

Dari gambar tersebut kemudian dapat ditentukan komponen-komponen apa yang dibutuhkan untuk dapat membangun tampilan GUI dari program simulator ini, misalnya label, text field, button, panel dan sebagainya. Selain itu, pada class *NamlSimulator* ini juga dilakukan proses inisialisasi animasi OpenGL yang akan digunakan pada saat pemanggilan class algoritma nantinya.

Berikut adalah beberapa method penting yang terdapat pada class ini.

1. *setupGL()*, merupakan method yang digunakan untuk proses inisialisasi animasi OpenGL di dalam program simulator.
2. *setupGUI()*, merupakan method yang digunakan untuk proses inisialisasi tampilan interface program simulator.
3. *init()*, merupakan method yang langsung dieksekusi oleh program simulator pada saat simulasi dimulai.

4. *display()*, merupakan method yang dieksekusi secara iteratif oleh program simulator selama program dijalankan. Dengan demikian, proses yang terjadi selama simulasi berlangsung dapat ditampilkan dalam bentuk animasi pergerakan robot.

5. *reshape()* dan *displayChanged()*, merupakan method yang dieksekusi oleh program jika terjadi perubahan pada jendela program selama simulasi berlangsung. Untuk memudahkan pekerjaan, ukuran dan event lainnya pada jendela program simulator dibuat agar tidak dapat berubah.

5.4.2 Class NamlAlgorithm

Class ini merupakan interface yang harus diimplementasikan oleh algoritma yang akan digunakan di dalam simulasi. Pada class ini hanya perlu dibuat dua abstract method yang implementasinya ditentukan oleh masing-masing class yang menjadi sub-class dari interface *NamlAlgorithm* ini. Kedua abstract method yang dimaksud adalah sebagai berikut.

- *initialization()*, yaitu method yang menentukan proses yang terjadi dalam simulasi pada saat robot berada pada fase inisialisasi.
- *iteration()*, yaitu method yang menentukan proses yang terjadi dalam simulasi pada saat robot berada pada fase iterasi.

5.4.3 Class ChainedShortest

Class ini merupakan class yang merepresentasikan penerapan algoritma *Chained Shortest Distance* di dalam simulasi. Class ini merupakan sub-class dari interface *NamlAlgorithm* sehingga harus dilakukan implementasi terhadap kedua abstract method yang dijelaskan sebelumnya. Berikut akan dijelaskan penerapan yang dilakukan pada kedua method tersebut pada algoritma ini.

1. *initialization()*

```
for each Target in ArrayOfTarget
        P_target ← read position from file ("target.txt")
end loop
loop each Robot in ArrayOfRobot
        P_robot ← define position randomly
        allGoal ← ArrayOfTarget
        goal ← allGoal[0]
        start
end loop
```

Pada fase inisialisasi, dilakukan instansiasi semua objek sesuai dengan jumlah yang ditentukan. Keseluruhan objek ini kemudian disimpan pada array yang sesuai. Pada saat instansiasi robot dilakukan, masing-masing robot diberikan informasi posisi target-target yang akan dihubungkan melalui jaringan komunikasi. Robot kemudian menentukan salah satu target yang dijadikan posisi tujuan, yaitu target yang mempunyai prioritas paling tinggi. Pada algoritma ini, prioritas tertinggi adalah target yang disimpan pada index terendah yang belum dikunjungi yaitu index 0.

2. *iteration()*

```
for each Target in ArrayOfTarget
        re-define position of Target in workspace
end loop
for each Robot in ArrayOfRobot
        define new position of Robot by movement algorithm
        if | P_robot – P_things | <= sensing area then
```

```
            if things==target then
                connectTo ← target
                distance ← | P_robot − P_target |
            else then
                if things.connected==TRUE or things.distance
                > connectTo.distance then
                    if connectTo==NULL
                        connectTo ← things
                        distance ← | P_robot − P_things | +
                                things.distance
                    end if
                end if
            end if
        end if
        if | P_robot − P_connectTo | > sensing area then
            stop
        end if
        if | P_robot − P_goal | <= sensing area then
            goal ← next target from allGoal
            if no more goal then
                stop
            end if
        end if
end loop
```

Pada fase iterasi, dilakukan pendefinisian ulang posisi semua objek di dalam simulasi. Karena target tidak bergerak selama simulasi berlangsung, maka tidak terjadi perubahan pada posisi semua target. Sedangkan pada robot akan terjadi perubahan posisi yang mengikuti algoritma pergerakan robot seperti yang sudah dijelaskan sebelumnya. Selain itu, robot juga akan melakukan pengecekan apakah dirinya terhubung dengan target yang sedang ditujunya atau tidak, apakah dengan pergerakan yang dilakukannya robot akan menjadi tidak terhubung lagi dengan target yang sudah dikunjunginya, serta apakah semua target sudah terhubung dengan robot.

5.4.4 Class PriorToNearest

Class ini merupakan class yang merepresentasikan penerapan algoritma *Prior to the Nearest* di dalam simulasi. Class ini merupakan sub-class dari interface *NamlAlgorithm* sehingga harus dilakukan implementasi terhadap kedua abstract method yang dijelaskan sebelumnya. Berikut akan dijelaskan penerapan yang dilakukan pada kedua method tersebut pada algoritma ini.

1. *initialization()*

```
for each Target in ArrayOfTarget
        P_target ← read position from file ("target.txt")
end loop
loop each Robot in ArrayOfRobot
        P_robot ← define position randomly
        1stTarget = Target where minimum | P_robot – P_target |
        2ndTarget = Target where maximum | P_robot - P_target |
        goal ← it's own position
        start
end loop
```

Pada fase inisialisasi dilakukan instansiasi semua objek sesuai dengan jumlah yang ditentukan. Keseluruhan objek ini kemudian disimpan pada array yang sesuai. Pada saat instansiasi robot dilakukan, masing-masing robot menentukan target berjarak paling dekat sebagai target pertama, serta target berjarak paling jauh sebagai target kedua. Selain itu, robot menentukan posisinya sendiri sebagai prioritas pada saat simulasi dimulai.

2. *iteration()*

```
for each Target in ArrayOfTarget
        re-define position of Target in workspace
end loop
for each Robot in ArrayOfRobot
        if | P_robot – P_things | <= unstable communication area then
                if | P_things – P_1stTarget | < | P_robot – P_1stTarget | then
                        if 1stPrior==NULL or | P_robot – P_1stPrior | >
                        | P_robot – P_things | then
                                1stPrior ← things
                        end if
                else
                        if 2ndPrior==NULL or | P_robot – P_2ndPrior | >
                        | P_robot – P_things | then
                                2ndPrior ← things
                        end if
                end if
        else
                if 1stPrior==NULL then
```

```
                    1stPrior ← 1stTarget
            end if
            if 2ndPrior==NULL then
                    2ndPrior ← 2ndTarget
            end if
    end if
    if | P_robot – P_1stPrior | <= sensing area then
            goal ← 2ndPrior
            if | P_robot – P_2ndPrior | <= sensing area then
                    stop
            else
                    start
            end if
    else then
            goal ← 1stPrior
            start
    end if
    define new position of Robot by movement algorithm
end loop
```

Pada fase iterasi, dilakukan pendefinisian ulang posisi semua objek di dalam simulasi. Sama seperti pada algoritma sebelumnya, posisi target tidak mengalami perubahan selama simulasi, sedangkan posisi robot akan mengalami perubahan dengan mengikuti algoritma pergerakan robot. Sebelum melakukan pergerakan, robot akan menentukan prioritas pertama dan kedua pergerakannya terlebih dahulu. Untuk prioritas pertama adalah robot lain terdekat yang berada paling

dekat dengan target pertama, atau target pertama jika tidak ada robot lain di sekitar robot tersebut. Prioritas kedua adalah robot lain terdekat yang berada paling dekat dengan target kedua, atau target kedua jika tidak ada robot lain di sekitar robot tersebut. Robot menjadikan prioritas kedua sebagai tujuan ketika sudah mencapai prioritas pertama. Robot akan berhenti jika prioritas kedua sudah tercapai. Jika belum, robot akan terus bergerak mendekati prioritas pertama.

5.4.5 Class PositionByLine

Class ini merupakan class yang merepresentasikan penerapan algoritma *Position By Line* di dalam simulasi. Class ini merupakan sub-class dari interface *NamlAlgorithm* sehingga harus dilakukan implementasi terhadap kedua abstract method yang dijelaskan sebelumnya. Berikut akan dijelaskan penerapan yang dilakukan pada kedua method tersebut pada algoritma ini.

1. *initialization()*

```
for each Target in ArrayOfTarget
        P_target ← read position from file ("target.txt")
end loop
loop each Robot in ArrayOfRobot
        P_robot ← define position randomly
        allGoal ← ArrayOfTarget
        if there is any solution with number of robot then
                search for all feasible position
                goal ← closest feasible position
                start
        end
end loop
```

Pada fase inisialisasi, dilakukan instansiasi semua objek sesuai dengan jumlah yang ditentukan. Keseluruhan objek ini kemudian disimpan pada array yang sesuai. Pada saat instansiasi robot dilakukan, masing-masing robot diberikan informasi posisi target-target yang akan dihubungkan melalui jaringan komunikasi, serta informasi jumlah semua robot. Dengan memanfaatkan informasi jumlah semua robot tersebut, dapat ditentukan apakah terdapat solusi pada permasalahan yang diberikan atau tidak. Simulasi hanya akan dimulai jika terdapat solusi konfigurasi jaringan dengan jumlah robot yang tersedia. Selain itu, juga akan ditentukan nilai *feasible solution*, di mana posisi yang paling dekat dari robot akan dijadikan sebagai prioritas posisi tujuan, sesuai dengan penjelasan sebelumnya.

2. *iteration()*

```
for each Target in ArrayOfTarget
        re-define position of Target in workspace
end loop
for each Robot in ArrayOfRobot
        define new position of Robot by movement algorithm
        if | P_goal – P_things | <= speed then
                if things==robot and things.speed <= 0 then
                        goal ← next closest feasible position
                        if no more feasible position then
                                stop
                        end if
                end if
        end if
end loop
```

Pada fase iterasi, dilakukan pendefinisian ulang posisi semua objek di dalam simulasi. Posisi target akan selalu sama sepanjang simulasi, sedangkan posisi robot akan terjadi perubahan yang mengikuti algoritma pergerakan robot seperti yang sudah dijelaskan sebelumnya.

Selain itu, robot juga akan melakukan pengecekan apakah posisi yang sedang ditujunya sudah ditempati oleh objek lain atau tidak. Jika sudah ditempati dan jika objek yang menempati posisi itu merupakan robot lain yang sudah dalam keadaan tidak bergerak, maka robot akan menentukan *feasible position* terdekat berikutnya sebagai posisi tujuan. Hal ini mengikuti algoritma *Position By Line* seperti yang sudah dijelaskan sebelumnya.

5.4.6 Class Things

Class ini merepresentasikan semua objek di dalam simulasi sehingga menjadi superclass dari class *Robot*, *Target*, dan *Source*. Adapun pembuatan class ini sebagai superclass dari objek-objek pada simulasi dimaksudkan untuk mencegah redundansi penyimpanan informasi yang sama-sama dimiliki oleh semua objek. Informasi yang disimpan pada class ini adalah karakteristik yang membedakan suatu objek dari jenis objek yang lain, misalnya informasi posisi koordinat objek, serta ciri fisik objek yang bersangkutan, seperti warna dan ukuran objek.

5.4.7 Class Robot

Class ini merepresentasikan objek robot di dalam simulasi. Class ini merupakan sub-class dari class *Things* sehingga informasi fisik yang menunjukkan karakteristik objek robot sudah didefinisikan pada class *Things*. Selain itu terdapat beberapa informasi tambahan lain yang perlu disimpan pada class *Robot*, yaitu informasi sudut pergerakan, radius *sensing area*, range robot, arah pergerakan, serta posisi tujuan pergerakan robot. Selain itu, pada algoritma tertentu juga terdapat beberapa informasi tambahan yang diperlukan dalam rangka memenuhi spesifikasi robot yang dibutuhkan, misalnya informasi posisi semua target dalam

urutan yang memiliki prioritas tertentu, informasi radius *unstable communication area* serta informasi prioritas pertama dan prioritas kedua pergerakan robot pada algoritma *Prior to the Nearest*, ataupun informasi semua *feasible position* pada algoritma *Position By Line*.

5.4.8 Class Target dan Source

Class *Target* merepresentasikan objek target di dalam simulasi. Class ini merupakan sub-class dari class *Things*. Tidak ada informasi khusus yang disimpan pada class ini karena target hanyalah objek yang bersifat diam selama simulasi berlangsung, maka. Pembuatan class ini hanya dimaksudkan untuk merepresentasikan objek target dengan benar sehingga dapat dibedakan dari objek robot, sedangkan class *Source* merupakan sub-class dari class *Target* yang sebenarnya juga merepresentasikan objek yang serupa. Pembedaan class *Source* dari class *Target* dimaksudkan untuk merepresentasikan dua jenis objek target yang berbeda pada algoritma *Prior to the Nearest*, dalam penelitian Fukuda dkk.

5.4.9 Class Coordinate dan DistanceCoordinate

Class *Coordinate* merupakan class yang merepresentasikan posisi koordinat dalam pasangan tiga bilangan bertipe floating point. Masing-masingnya menunjukkan informasi posisi pada sumbu *x*, sumbu *y*, serta sumbu *z*.

Class *DistanceCoordinate* merupakan sub-class dari class *Coordinate* dengan tambahan informasi suatu bilangan floating point lain yang menunjukkan jarak dari koordinat tersebut terhadap suatu posisi. Class ini berguna dalam menentukan *feasible position* terdekat pada algoritma *Position By Line*.

5.4.10 Class Function

Class ini merupakan class yang berisikan fungsi matematik yang diperlukan di dalam simulasi. Terdapat dua method static pada class ini, yaitu sebagai berikut.

- Fungsi untuk menghitung jarak antara dua posisi koordinat, yaitu:

```
distanceOf2Nodes (Coordinate c1, Coordinate c2)
{
        distance ← √ ((c2.x - c1.x)^2 + (c2.y - c1.y)^2)
}
```

- Fungsi untuk menghitung kemiringan suatu garis atau sudut antara suatu garis dengan sumbu *x*, yaitu:

```
angleOf2Nodes (Coordinate c1, Coordinate c2)
{
        theta ← arc tan ((c2.y - c1.y) / (c2.x - c1.x))
}
```

BAB 6 UJI COBA DAN EVALUASI

Bab ini akan menjelaskan tentang uji coba simulasi yang dilakukan dengan menggunakan simulator NAML yang telah dikembangkan. Dari hasil yang diperoleh melalui kegiatan simulasi, akan dilakukan evaluasi untuk melihat kinerja algoritma yang digunakan. Uji coba dan evaluasi dilakukan agar algoritma yang dikembangkan pada kegiatan penelitian dapat dinilai secara objektif.

6.1 Pengujian Algoritma Chained Shortest Distance

Pada bagian ini, akan ditunjukkan hasil yang diperoleh melalui kegiatan pengujian algoritma *Chained Shortest Distance* dengan menggunakan simulator NAML yang telah dikembangkan. Kinerja algoritma dilihat melalui sejumlah permasalahan acak yang diberikan dengan parameter jumlah robot, radius sensing area dan range yang berbeda-beda. Pada percobaan ini akan dilakukan perbandingan jumlah permasalahan yang dapat diselesaikan dan jumlah permasalahan yang tidak dapat diselesaikan, waktu rata-rata yang dibutuhkan untuk menemukan solusi, serta waktu maksimum dan waktu minimumnya. Selain itu, masing-masing percobaan dibatasi waktu percobaannya agar tidak melebihi 20 detik. Setiap percobaan yang berjalan melebihi batas waktu tersebut akan diasumsikan tidak menemukan solusi.

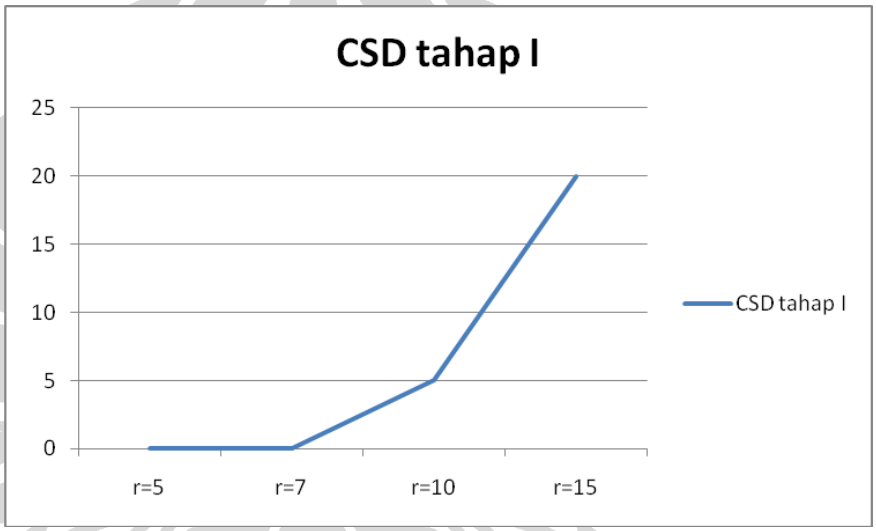
Kegiatan percobaan dibagi atas beberapa tahap. Pada tahap pertama, target berada pada posisi P_1(-0.5, -0.5) dan P_2(0.5, 0.5) dari titik acuan P_0(0, 0). Pada tahap kedua, posisi target diubah menjadi P_1(-0.5, 0.5) dan P_2(0.5, -0.5). Hal ini dilakukan untuk melihat perbandingan kinerja algoritma dengan posisi target yang berbeda. Pada tahapan selanjutnya akan dilihat kinerja algoritma berdasarkan perbedaan nilai radius *sensing area* dan range-nya.

Tabel 6.1 berikut di bawah ini merupakan hasil yang diperoleh melalui dua puluh percobaan pada tahap pertama dengan jumlah robot = 5, 7, 10, 15. Radius sensing area yang digunakan = 0.25 dan range = 0.0005.

Tabel 6.1 - Hasil Percobaan Tahap I Algoritma CSD

Jumlah Robot (dalam unit)	Tingkat Keberhasilan (dalam %)
5	0
7	0
10	5
15	20

Hasil yang diperoleh pada tabel ini kemudian dapat juga direpresentasikan dalam bentuk grafik seperti pada gambar 6.1 berikut di bawah ini.



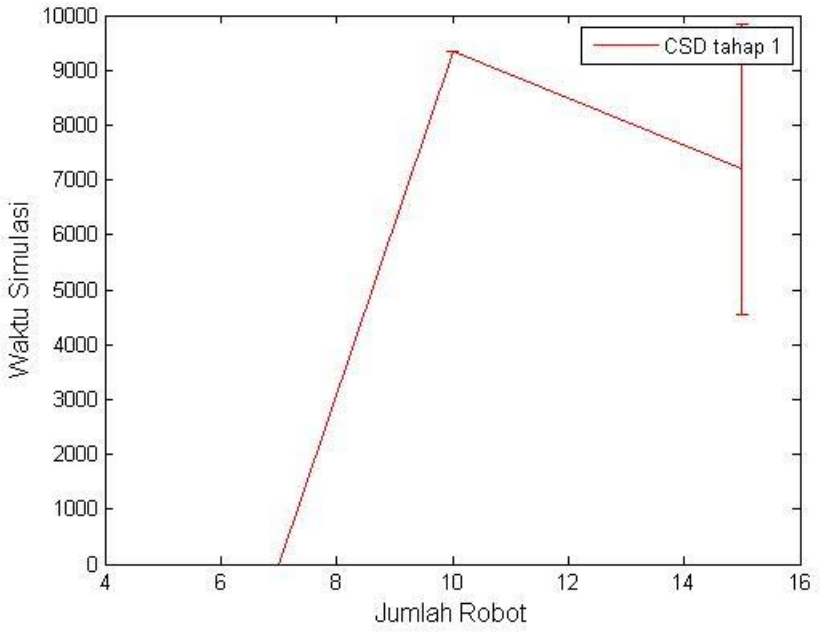
Gambar 6.1 - Grafik Tingkat Keberhasilan Algoritma CSD Tahap I

Tabel 6.2 berikut di bawah ini kemudian menunjukkan perbandingan waktu simulasi pada tahap ini. Pada tabel ini hanya diperhitungkan percobaan-percobaan yang berhasil ditemukan solusinya.

Tabel 6.2 - Waktu Simulasi Percobaan Tahap I Algoritma CSD

Jumlah Robot (dalam unit)	Waktu (dalam ms)				
	Ave	Min	Max	Std Dev	Range
10	9344	9344	9344	-	1
15	7193.5	3547	9578	2654.48	6032

Selanjutnya hasil yang diperoleh pada percobaan tahap I ditampilkan dalam bentuk grafik, seperti pada gambar 6.2 berikut di bawah ini. Pada grafik tersebut hanya ditampilkan hasil yang diperoleh pada percobaan dengan menggunakan 15 robot, karena pada pada percobaan lainnya algoritma ini tidak berhasil menemukan solusi permasalahan.



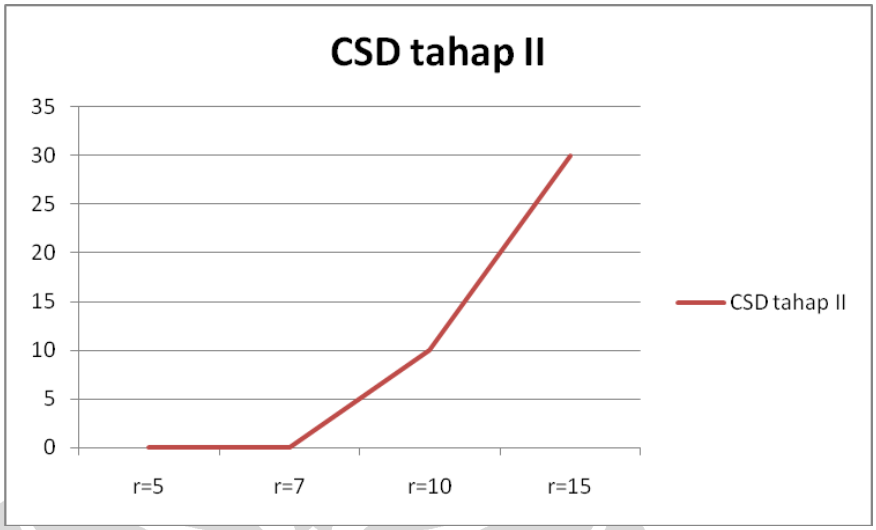
Gambar 6.2 - Grafik Waktu Solusi Algoritma CSD Tahap I

Tabel 6.3 berikut di bawah ini menunjukkan hasil yang diperoleh melalui dua puluh percobaan tahap kedua dengan jumlah robot = 5, 7, 10, 15. Radius sensing area yang digunakan = 0.25 dan range = 0.0005.

Tabel 6.3 - Hasil Percobaan Tahap II Algoritma CSD

Jumlah Robot (dalam unit)	Tingkat Keberhasilan (dalam %)
5	0
7	0
10	10
15	30

Hasil yang diperoleh pada tabel ini kemudian dapat juga direpresentasikan dalam bentuk grafik seperti pada gambar 6.3 berikut di bawah ini.



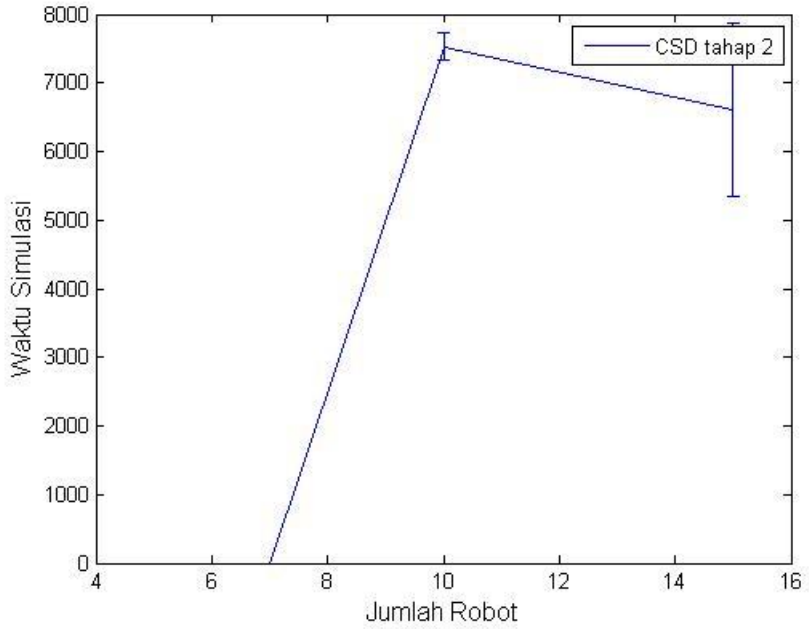
Gambar 6.3 - Grafik Tingkat Keberhasilan Algoritma CSD Tahap II

Tabel 6.4 berikut di bawah ini kemudian menunjukkan perbandingan waktu simulasi pada tahap ini. Pada tabel ini hanya diperhitungkan percobaan-percobaan yang berhasil ditemukan solusinya.

Tabel 6.4 - Waktu Simulasi Percobaan Tahap II Algoritma CSD

Jumlah Robot (dalam unit)	Waktu (dalam ms)				
	Ave	Min	Max	Std Dev	Range
10	7531	7390	7672	199.40	283
15	6606.67	4782	8093	1264.17	3312

Selanjutnya hasil yang diperoleh pada percobaan tahap II ditampilkan dalam bentuk grafik, seperti pada gambar 6.4 berikut di bawah ini. Pada grafik tersebut hanya ditampilkan hasil yang diperoleh pada percobaan dengan menggunakan 10 dan 15 robot, karena pada pada percobaan lainnya algoritma ini tidak berhasil menemukan solusi permasalahan.



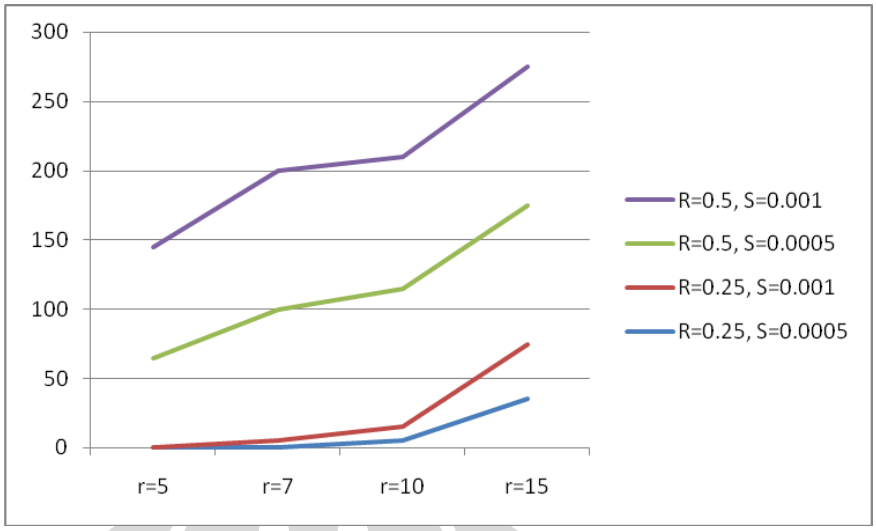
Gambar 6.4 - Grafik Waktu Solusi Algoritma CSD Tahap II

Pada tahap ketiga akan dilakukan percobaan untuk melihat perbedaan parameter radius *sensing area R* dan range *S* terhadap kinerja algoritma. Hasil yang diperoleh ditunjukkan pada tabel 6.5 berikut di bawah ini.

Tabel 6.5 - Tingkat Kesuksesan Algoritma CSD Berdasarkan Sensing Area dan Range untuk Kasus Dua Sumber

Jumlah Robot (dalam unit)	Tingkat Keberhasilan (dalam %)			
	R=0.25, S=0.0005	R=0.25, S=0.001	R=0.5, S=0.0005	R=0.5, S=0.001
5	0	0	65	80
7	0	5	95	100
10	5	10	100	95
15	35	40	100	100

Dari pengujian ini jika disajikan dalam bentuk grafik akan dilihat nilai perbandingan seperti pada gambar 6.5 berikut di bawah ini.



Gambar 6.5 - Grafik Pengaruh Sensing Area dan Range Robot Algoritma CSD

Kemudian perbandingan waktu simulasi pada masing-masing jumlah robot dengan beberapa variasi parameter radius *sensing area r* dan range *s* ditunjukkan pada tabel 6.6 – 6.9 berikut di bawah ini.

Tabel 6.6 - Perbandingan Waktu Simulasi Algoritma CSD Jumlah Robot=5

Radius Sensing	Range	Waktu (dalam ms)				
		Ave	Min	Max	Std Dev	Range
0.5	0.0005	3866.62	1031	6422	1950.32	5392
	0.001	1925.75	719	3563	897.74	2845

Tabel 6.7 - Perbandingan Waktu Simulasi Algoritma CSD Jumlah Robot=7

Radius Sensing	Range	Waktu (dalam ms)				
		Ave	Min	Max	Std Dev	Range
0.25	0.001	4891	4891	4891	-	1
0.5	0.0005	2526.32	704	4578	1000.44	3875
	0.001	1602.5	391	2985	891.36	2595

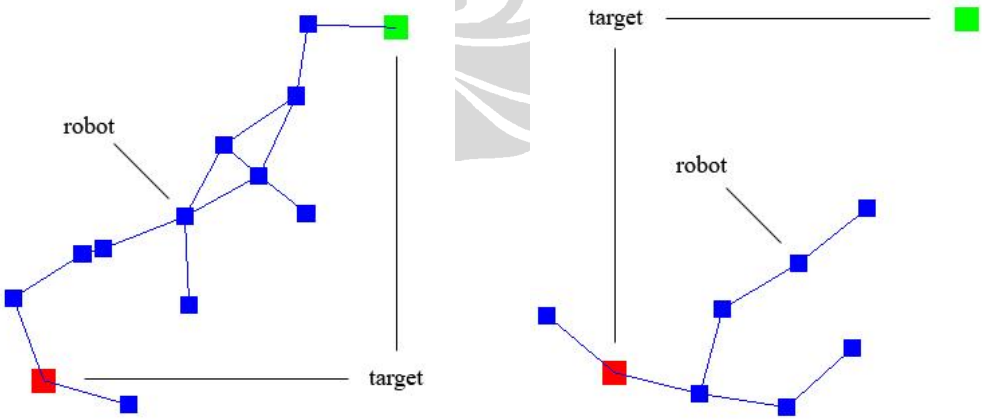
Tabel 6.8 - Perbandingan Waktu Simulasi Algoritma CSD Jumlah Robot=10

Radius Sensing	Range	Waktu (dalam ms)				
		Ave	Min	Max	Std Dev	Range
0.25	0.0005	4282	4282	4282	-	1
	0.001	3336	3110	3562	319.61	453
0.5	0.0005	2023.5	547	4219	1089.9	3673
	0.001	1359.52	515	3937	953.92	3423

Tabel 6.9 - Perbandingan Waktu Simulasi Algoritma CSD Jumlah Robot=15

Radius Sensing	Range	Waktu (dalam ms)				
		Ave	Min	Max	Std Dev	Range
0.25	0.0005	4584.86	2797	7625	1696.30	4829
	0.001	2902.5	1953	3906	640.64	1954
0.5	0.0005	1064.9	359	2219	540.88	1861
	0.001	2421.71	937	4046	1074.32	3110

Gambar 6.6 berikut di bawah ini merupakan contoh pengujian algoritma CSD, dimana permasalahan yang berhasil diselesaikan ditunjukkan gambar 6.6.a dan permasalahan yang tidak berhasil diselesaikan ditunjukkan gambar 6.6.b.



(a) **(b)**

Gambar 6.6 - Tampilan Pengujian Algoritma CSD

6.2 Pengujian Algoritma Prior to the Nearest

Pada bagian ini, akan ditunjukkan hasil yang diperoleh melalui kegiatan pengujian algoritma *Prior to the Nearest* dengan menggunakan simulator NAML yang telah dikembangkan. Kinerja algoritma dilihat melalui sejumlah permasalahan acak yang diberikan dengan parameter jumlah robot, radius sensing area dan range yang berbeda-beda. Pada percobaan ini akan dilakukan perbandingan jumlah permasalahan yang dapat diselesaikan dan jumlah permasalahan yang tidak dapat diselesaikan, waktu rata-rata yang dibutuhkan untuk menemukan solusi, serta waktu maksimum dan waktu minimumnya. Selain itu, masing-masing percobaan dibatasi waktu percobaannya agar tidak melebihi 20 detik. Setiap percobaan yang berjalan melebihi batas waktu tersebut akan diasumsikan tidak menemukan solusi.

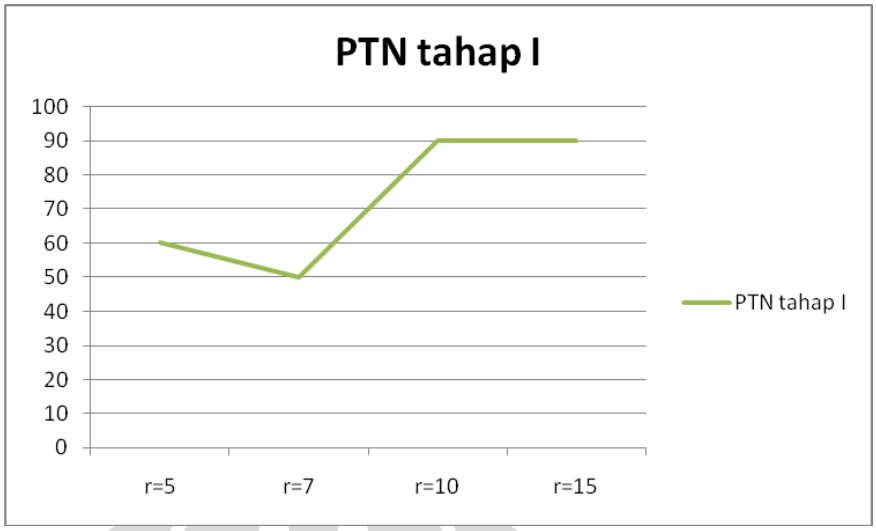
Kegiatan percobaan dibagi atas beberapa tahap. Pada tahap pertama, target berada pada posisi P_1(-0.5, -0.5) dan P_2(0.5, 0.5) dari titik acuan P_0(0, 0). Pada tahap kedua, posisi target diubah menjadi P_1(-0.5, 0.5) dan P_2(0.5, -0.5). Hal ini dilakukan untuk melihat perbandingan kinerja algoritma dengan posisi target yang berbeda. Pada tahapan selanjutnya akan dilihat kinerja algoritma berdasarkan perbedaan nilai radius *sensing area* dan range-nya.

Tabel 6.10 berikut di bawah ini merupakan hasil yang diperoleh melalui dua puluh percobaan pada tahap pertama dengan jumlah robot = 5, 7, 10, 15. Radius sensing area yang digunakan = 0.25 dan range = 0.0005.

Tabel 6.10 - Hasil Percobaan Tahap I Algoritma PTN

Jumlah Robot (dalam unit)	Tingkat Keberhasilan (dalam %)
5	60
7	50
10	90
15	90

Hasil yang diperoleh pada tabel ini kemudian dapat juga direpresentasikan dalam bentuk grafik seperti pada gambar 6.7 berikut di bawah ini.



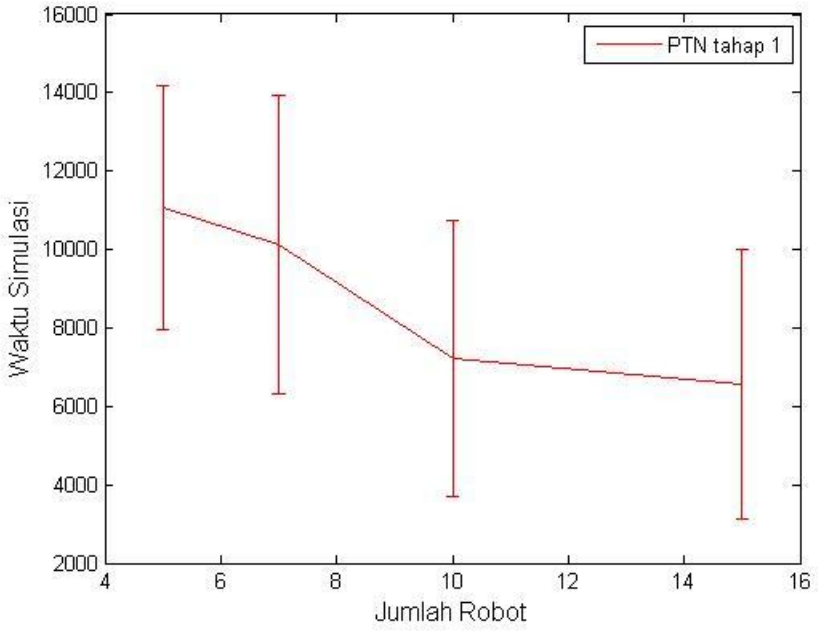
Gambar 6.7 - Grafik Tingkat Keberhasilan Algoritma PTN Tahap I

Tabel 6.11 berikut di bawah ini kemudian menunjukkan perbandingan waktu simulasi pada tahap ini.

Tabel 6.11 - Waktu Simulasi Percobaan Tahap I Algoritma PTN

Jumlah Robot (dalam unit)	Waktu (dalam ms)				
	Ave	Min	Max	Std Dev	Range
5	11078	7125	17281	11078	10157
7	10136	4453	14766	10136	10314
10	7203.89	2344	13609	7203.89	11266
15	6572.94	1531	16703	6572.94	15173

Selanjutnya hasil yang diperoleh pada percobaan tahap I ditampilkan dalam bentuk grafik, seperti pada gambar 6.8 berikut di bawah ini.



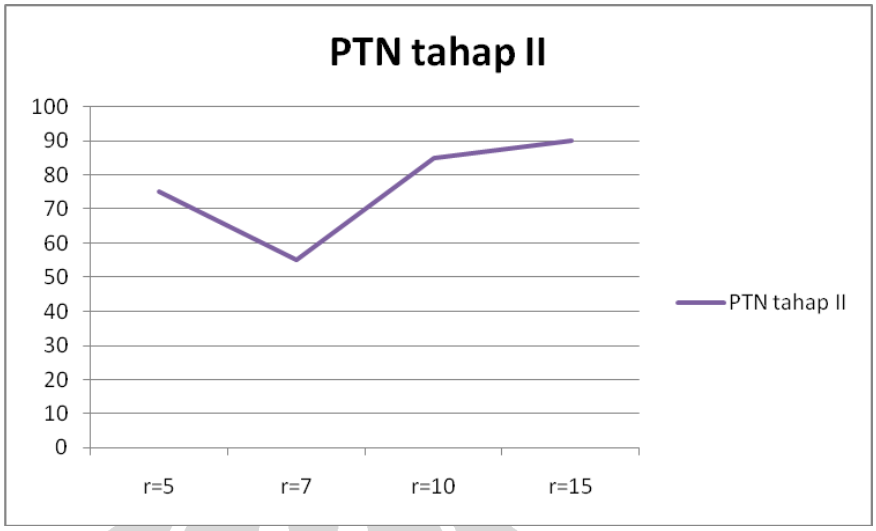
Gambar 6.8 - Grafik Pengujian Algoritma PTN Tahap I

Tabel 6.12 berikut di bawah ini menunjukkan hasil yang diperoleh melalui dua puluh percobaan tahap kedua dengan jumlah robot = 5, 7, 10, 15.

Tabel 6.12 - Hasil Percobaan Tahap II Algoritma PTN

Jumlah Robot (dalam unit)	Tingkat Keberhasilan (dalam %)
5	75
7	55
10	85
15	90

Hasil yang diperoleh pada tabel ini kemudian dapat juga direpresentasikan dalam bentuk grafik seperti pada gambar 6.9 berikut di bawah ini.



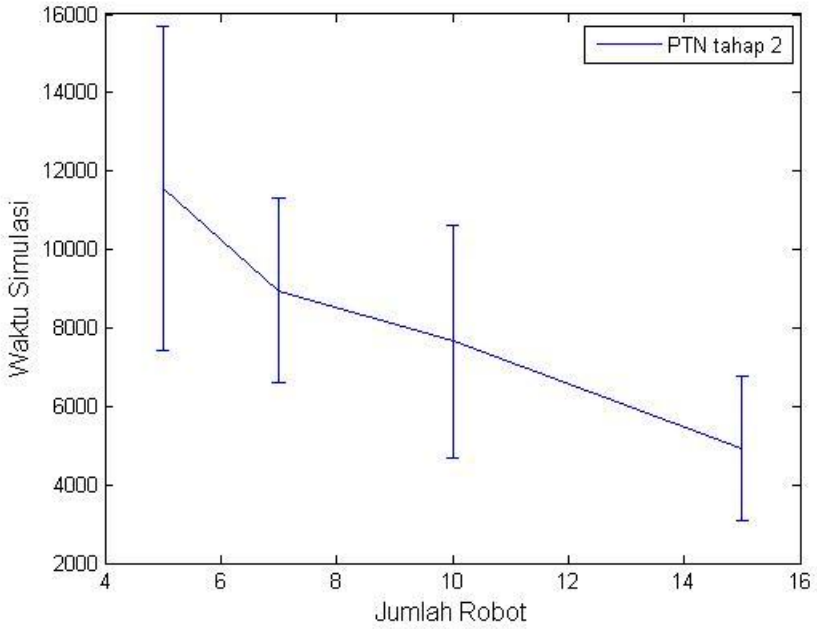
Gambar 6.9 - Grafik Tingkat Keberhasilan Algoritma PTN Tahap II

Tabel 6.13 berikut di bawah ini kemudian menunjukkan perbandingan waktu simulasi pada tahap ini. Pada tabel ini hanya diperhitungkan percobaan-percobaan yang berhasil ditemukan solusinya.

Tabel 6.13 - Waktu Simulasi Percobaan Tahap II Algoritma PTN

Jumlah Robot (dalam unit)	Waktu (dalam ms)				
	Ave	Min	Max	Std Dev	Range
5	11550.33	5069	18531	11550.33	13463
7	8944.54	5375	14640	8944.54	9266
10	7666.41	3563	13063	7666.41	9501
15	4926.17	2125	8281	4926.17	6157

Selanjutnya hasil yang diperoleh pada percobaan tahap II ditampilkan dalam bentuk grafik, seperti pada gambar 6.10 berikut di bawah ini.



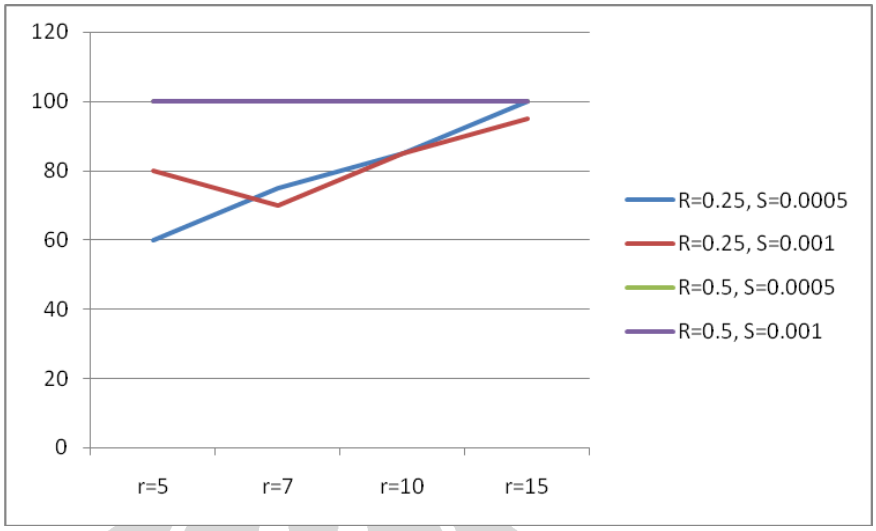
Gambar 6.10 - Grafik Waktu Solusi Algoritma PTN Tahap II

Pada tahap ketiga akan dilakukan percobaan untuk melihat perbedaan parameter radius *sensing area R* dan range *S* terhadap kinerja algoritma. Hasil yang diperoleh ditunjukkan pada tabel 6.14 berikut di bawah ini.

Tabel 6.14 - Tingkat Kesuksesan Algoritma PTN Berdasarkan Sensing Area dan Range untuk Kasus Dua Sumber

Jumlah Robot (dalam unit)	Tingkat Keberhasilan (dalam %)			
	R=0.25, S=0.0005	R=0.25, S=0.001	R=0.5, S=0.0005	R=0.5, S=0.001
5	60	80	100	100
7	75	70	100	100
10	85	85	100	100
15	100	95	100	100

Dari pengujian ini jika disajikan dalam bentuk grafik akan dilihat nilai perbandingan seperti pada gambar 6.11 berikut di bawah ini.



Gambar 6.11 - Grafik Pengaruh Sensing Area dan Range Robot Algoritma PTN

Kemudian perbandingan waktu simulasi pada masing-masing jumlah robot dengan beberapa variasi parameter radius *sensing area r* dan range *s* ditunjukkan pada tabel 6.15 – 6.18 berikut di bawah ini.

Tabel 6.15 - Perbandingan Waktu Simulasi Algoritma PTN Jumlah Robot=5

Radius Sensing	Range	Waktu (dalam ms)				
		Ave	Min	Max	Std Dev	Range
0.25	0.0005	9637.91	6047	14296	2497.11	8250
	0.001	4783.44	3125	7250	1203.24	4126
0.5	0.0005	2982.7	1265	5672	1314.6	4408
	0.001	1905.4	734	3375	672.48	2642

Tabel 6.16 - Perbandingan Waktu Simulasi Algoritma PTN Jumlah Robot=7

Radius Sensing	Range	Waktu (dalam ms)				
		Ave	Min	Max	Std Dev	Range
0.25	0.0005	6492.67	3687	10672	2281.06	6986
	0.001	4261.14	1984	17140	3941.43	15157
0.5	0.0005	2335.05	406	4062	1122.53	3657
	0.001	1316.4	484	2718	624.49	2235

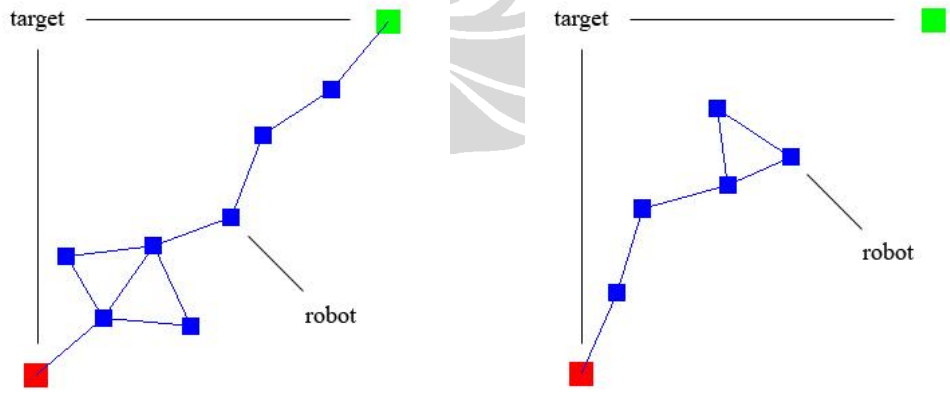
Tabel 6.17 - Perbandingan Waktu Simulasi Algoritma PTN Jumlah Robot=10

Radius Sensing	Range	Waktu (dalam ms)				
		Ave	Min	Max	Std Dev	Range
0.25	0.0005	5201.41	1485	11063	2147.96	9579
	0.001	2644.39	1266	5734	1018.43	4469
0.5	0.0005	1438.95	375	2813	685	2439
	0.001	1115.55	468	2125	456.67	1658

Tabel 6.18 - Perbandingan Waktu Simulasi Algoritma PTN Jumlah Robot=15

Radius Sensing	Range	Waktu (dalam ms)				
		Ave	Min	Max	Std Dev	Range
0.25	0.0005	4147.6	1843	8610	1471.72	6768
	0.001	2548.53	859	8047	1881.29	7189
0.5	0.0005	1160.15	344	4562	1065.21	4219
	0.001	2088.25	1031	4250	953.95	3220

Gambar 6.12 berikut di bawah ini merupakan contoh pengujian algoritma PTN, dimana permasalahan yang berhasil diselesaikan ditunjukkan gambar 6.12.a dan pada gambar 6.12.b ditunjukkan permasalahan yang tidak dapat diselesaikan..



(a) **(b)**

Gambar 6.12 - Tampilan Pengujian Algoritma PTN

6.3 Pengujian Algoritma Position By Line

Pada bagian ini, akan ditunjukkan hasil yang diperoleh melalui kegiatan pengujian algoritma *Position By Line* dengan menggunakan simulator NAML yang telah dikembangkan. Kinerja algoritma dilihat melalui sejumlah permasalahan acak yang diberikan dengan parameter jumlah robot, radius sensing area dan range yang berbeda-beda. Pada percobaan ini akan dilakukan perbandingan jumlah permasalahan yang dapat diselesaikan dan jumlah permasalahan yang tidak dapat diselesaikan, waktu rata-rata yang dibutuhkan untuk menemukan solusi, serta waktu maksimum dan waktu minimumnya. Selain itu, masing-masing percobaan dibatasi waktu percobaannya agar tidak melebihi 20 detik. Setiap percobaan yang berjalan melebihi batas waktu tersebut akan diasumsikan tidak menemukan solusi.

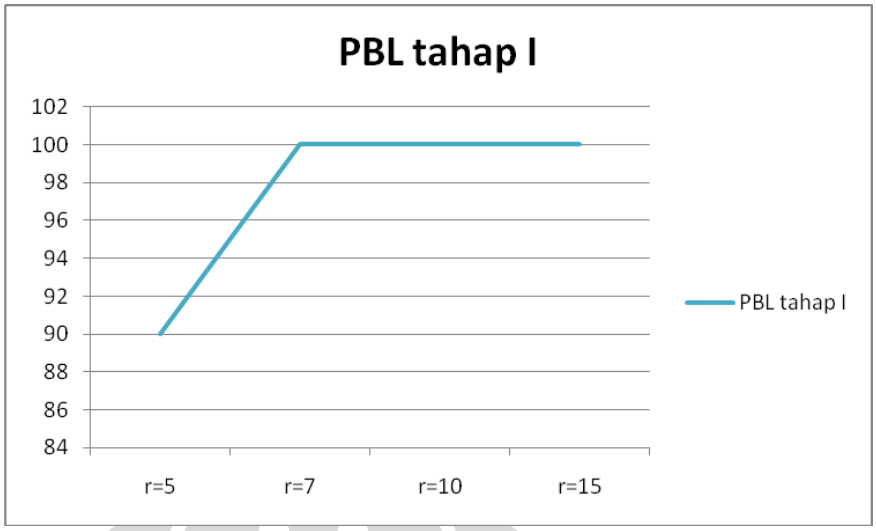
Kegiatan percobaan dibagi atas beberapa tahap. Pada tahap pertama, target berada pada posisi P_1(-0.5, -0.5) dan P_2(0.5, 0.5) dari titik acuan P_0(0, 0). Pada tahap kedua, posisi target diubah menjadi P_1(-0.5, 0.5) dan P_2(0.5, -0.5). Hal ini dilakukan untuk melihat perbandingan kinerja algoritma dengan posisi target yang berbeda. Pada tahapan selanjutnya akan dilihat kinerja algoritma berdasarkan perbedaan nilai radius *sensing area* dan range-nya.

Tabel 6.19 berikut di bawah ini merupakan hasil yang diperoleh melalui dua puluh percobaan pada tahap pertama dengan jumlah robot = 5, 7, 10, 15. Radius sensing area yang digunakan = 0.25 dan range = 0.0005.

Tabel 6.19 - Hasil Percobaan Tahap I Algoritma PBL

Jumlah Robot (dalam unit)	Tingkat Keberhasilan (dalam %)
5	90
7	100
10	100
15	100

Hasil yang diperoleh pada tabel ini kemudian dapat juga direpresentasikan dalam bentuk grafik seperti pada gambar 6.13 berikut di bawah ini.



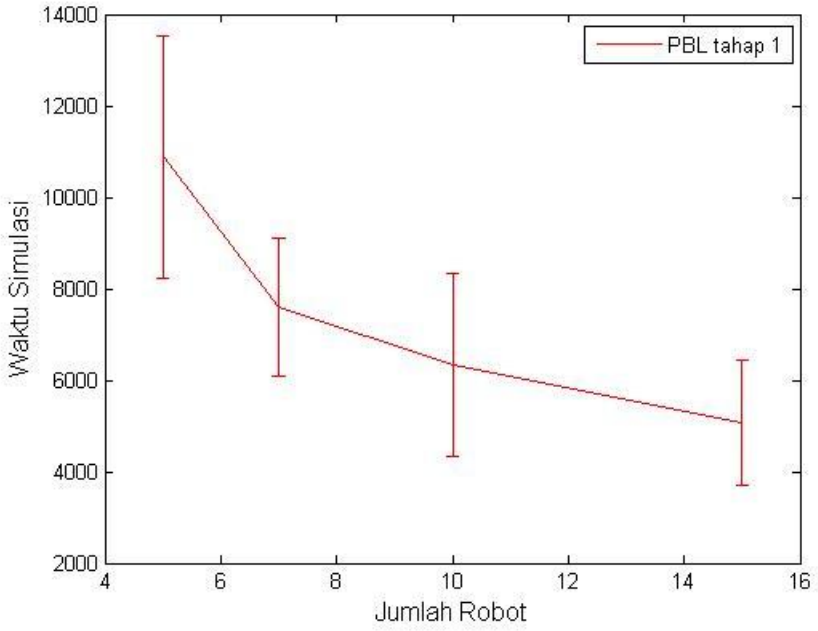
Gambar 6.13 - Grafik Tingkat Keberhasilan Algoritma PBL Tahap I

Tabel 6.20 berikut di bawah ini kemudian menunjukkan perbandingan waktu simulasi pada tahap ini.

Tabel 6.20 - Waktu Simulasi Percobaan Tahap I Algoritma PBL

Jumlah Robot (dalam unit)	Waktu (dalam ms)				
	Ave	Min	Max	Std Dev	Range
5	10883.56	6266	15875	2644.35	9610
7	7602.4	5078	10485	1517.14	5408
10	6326.6	4047	11360	2004.46	7314
15	5078.2	3000	8031	1363.92	5032

Selanjutnya hasil yang diperoleh pada percobaan tahap I ditampilkan dalam bentuk grafik, seperti pada gambar 6.14 berikut di bawah ini.



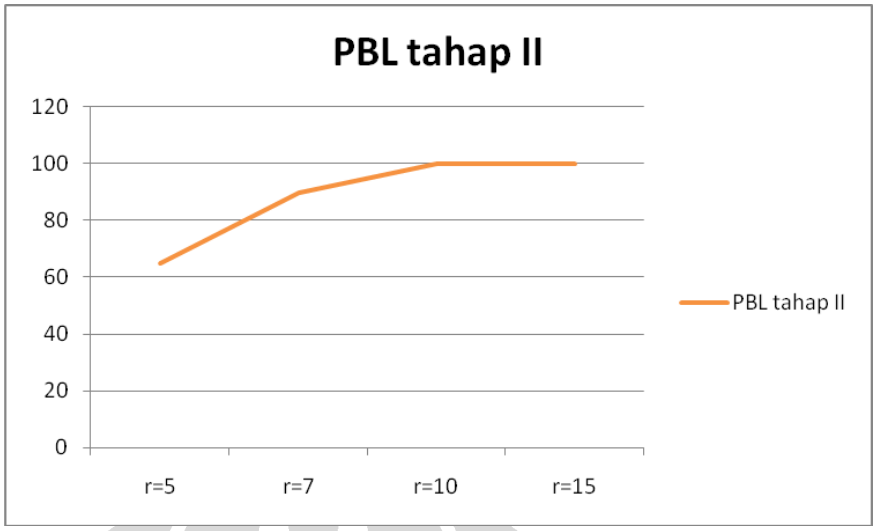
Gambar 6.14 - Grafik Pengujian Algoritma PBL Tahap I

Tabel 6.21 berikut di bawah ini menunjukkan hasil yang diperoleh melalui dua puluh percobaan tahap kedua dengan jumlah robot = 5, 7, 10, 15.

Tabel 6.21 - Hasil Percobaan Tahap II Algoritma PBL

Jumlah Robot (dalam unit)	Tingkat Keberhasilan (dalam %)
5	65
7	90
10	100
15	100

Hasil yang diperoleh pada tabel ini kemudian dapat juga direpresentasikan dalam bentuk grafik seperti pada gambar 6.15 berikut di bawah ini.



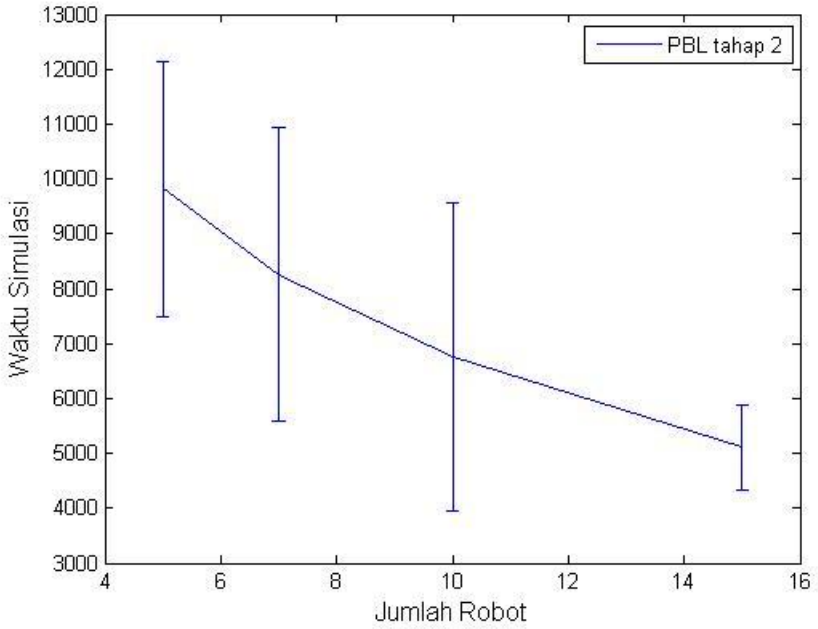
Gambar 6.15 - Grafik Tingkat Keberhasilan Algoritma PBL Tahap II

Tabel 6.22 berikut di bawah ini kemudian menunjukkan perbandingan waktu simulasi pada tahap ini. Pada tabel ini hanya diperhitungkan percobaan-percobaan yang berhasil ditemukan solusinya.

Tabel 6.22 - Waktu Simulasi Percobaan Tahap II Algoritma PBL

Jumlah Robot (dalam unit)	Waktu (dalam ms)				
	Ave	Min	Max	Std Dev	Range
5	9817.38	7047	13672	2329.66	6626
7	8258.61	4515	13125	2668.69	8611
10	6753.4	2531	16359	2811.43	13829
15	5112.55	3328	6250	778.07	2923

Selanjutnya hasil yang diperoleh pada percobaan tahap II ditampilkan dalam bentuk grafik, seperti pada gambar 6.16 berikut di bawah ini.



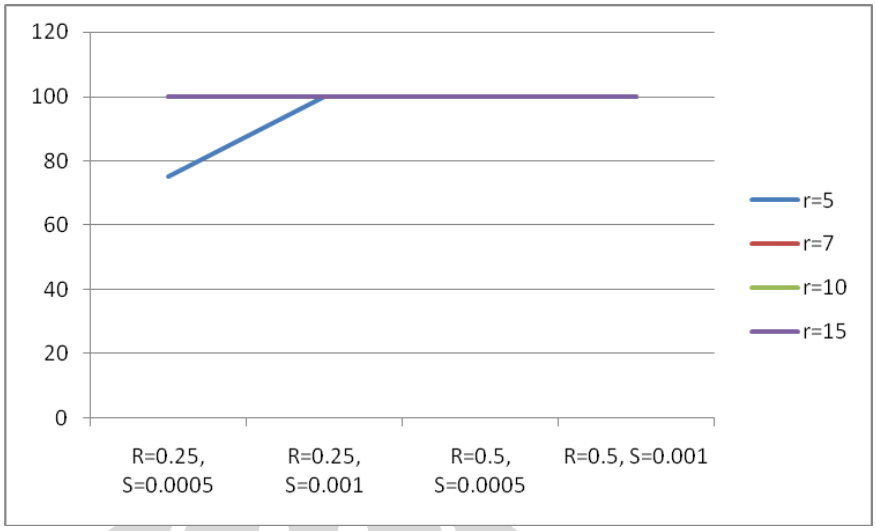
Gambar 6.16 - Grafik Waktu Solusi Algoritma PBL Tahap II

Pada tahap ketiga akan dilakukan percobaan untuk melihat perbedaan parameter radius *sensing area R* dan range *S* terhadap kinerja algoritma. Hasil yang diperoleh ditunjukkan pada tabel 6.23 berikut di bawah ini.

Tabel 6.23 - Tingkat Kesuksesan Algoritma PBL Berdasarkan Sensing Area dan Range untuk Kasus Dua Sumber

Jumlah Robot (dalam unit)	Tingkat Keberhasilan (dalam %)			
	R=0.25, S=0.0005	R=0.25, S=0.001	R=0.5, S=0.0005	R=0.5, S=0.001
5	75	100	100	100
7	100	100	100	100
10	100	100	100	100
15	100	100	100	100

Dari pengujian ini jika disajikan dalam bentuk grafik akan dilihat nilai perbandingan seperti pada gambar 6.17 berikut di bawah ini.



Gambar 6.17 - Grafik Pengaruh Sensing Area dan Range Robot Algoritma PBL

Kemudian perbandingan waktu simulasi pada masing-masing jumlah robot dengan beberapa variasi parameter radius *sensing area r* dan range *s* ditunjukkan pada tabel 6.24 – 6.27 berikut di bawah ini.

Tabel 6.24 - Perbandingan Waktu Simulasi Algoritma PBL Jumlah Robot=5

Radius Sensing	Range	Waktu (dalam ms)				
		Ave	Min	Max	Std Dev	Range
0.25	0.0005	9045.73	6047	14438	2426	8392
	0.001	4776.65	2360	8469	1462.8	6110
0.5	0.0005	3196.15	562	8234	1673.32	7673
	0.001	1939.8	422	3578	775.1	3157

Tabel 6.25 - Perbandingan Waktu Simulasi Algoritma PBL Jumlah Robot=7

Radius Sensing	Range	Waktu (dalam ms)				
		Ave	Min	Max	Std Dev	Range
0.25	0.0005	6165.65	3844	9812	1514.28	5969
	0.001	3456.95	1641	6969	1193.04	5329
0.5	0.0005	3261.6	672	6969	1389.2	6298
	0.001	1449.35	547	2531	552.63	1985

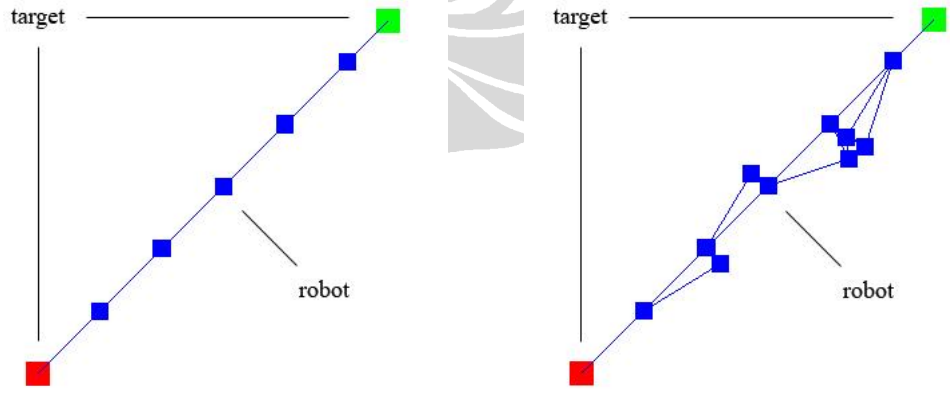
Tabel 6.26 - Perbandingan Waktu Simulasi Algoritma PBL Jumlah Robot=10

Radius Sensing	Range	Waktu (dalam ms)				
		Ave	Min	Max	Std Dev	Range
0.25	0.0005	4903.1	2969	8266	1199.39	5298
	0.001	2762.55	1282	6234	1087.06	4953
0.5	0.0005	2320.2	485	4015	1017.21	3531
	0.001	1437.4	828	2532	463.35	1705

Tabel 6.27 - Perbandingan Waktu Simulasi Algoritma PBL Jumlah Robot=15

Radius Sensing	Range	Waktu (dalam ms)				
		Ave	Min	Max	Std Dev	Range
0.25	0.0005	3550.85	2109	5375	967.93	3267
	0.001	2107.75	1250	2953	509.86	1704
0.5	0.0005	1741.35	407	3000	797.55	2594
	0.001	2260.2	1453	5406	865.57	3954

Gambar 6.18 berikut di bawah ini merupakan contoh pengujian algoritma PBL, dimana pada gambar 6.18.a robot yang digunakan berjumlah minimal dan pada gambar 6.18.b robot yang digunakan berjumlah banyak.



(a) **(b)**

Gambar 6.18 - Tampilan Pengujian Algoritma PBL

6.4 Permasalahan Banyak Sumber

Setelah pada bagian sebelumnya dijelaskan pengujian yang dilakukan terhadap perancangan jaringan komunikasi dengan dua sumber, pada bagian ini akan ditampilkan hasil percobaan yang diperoleh pada pengujian yang dilakukan terhadap perancangan jaringan komunikasi dengan banyak sumber.

6.4.1 Tingkat Kesuksesan Permasalahan Multisource

Tabel 6.28 - Tingkat Kesuksesan Algoritma CSD Berdasarkan Sensing Area dan Range untuk Kasus Banyak Sumber

Jumlah Robot (dalam unit)	Tingkat Keberhasilan (dalam %)			
	R=0.25, S=0.0005	R=0.25, S=0.001	R=0.5, S=0.0005	R=0.5, S=0.001
5	0	0	80	65
7	0	0	70	85
10	0	0	100	100
15	15	20	100	100

Tabel 6.29 - Tingkat Kesuksesan Algoritma PTN Berdasarkan Sensing Area dan Range untuk Kasus Banyak Sumber

Jumlah Robot (dalam unit)	Tingkat Keberhasilan (dalam %)			
	R=0.25, S=0.0005	R=0.25, S=0.001	R=0.5, S=0.0005	R=0.5, S=0.001
5	0	5	85	90
7	30	25	95	95
10	60	55	100	100
15	75	75	100	100

Tabel 6.30 - Tingkat Kesuksesan Algoritma PBL Berdasarkan Sensing Area dan Range untuk Kasus Banyak Sumber

Jumlah Robot (dalam unit)	Tingkat Keberhasilan (dalam %)			
	R=0.25, S=0.0005	R=0.25, S=0.001	R=0.5, S=0.0005	R=0.5, S=0.001
5	0	0	100	100
7	95	90	100	100
10	100	95	100	100
15	100	100	100	100

Kemudian juga perbandingan waktu yang dibutuhkan masing-masing algoritma untuk menemukan solusi permasalahan, ditampilkan pada tabel-tabel di bawah.

6.4.2 Algoritma CSD untuk Multisource

Tabel 6.31 - Perbandingan Waktu Simulasi Algoritma CSD Jumlah Robot=5

Radius Sensing	Range	Waktu (dalam ms)				
		Ave	Min	Max	Std Dev	Range
0.5	0.0005	3879.94	1531	6750	1680.86	5220
	0.001	1968.77	578	4297	1165.67	3720

Tabel 6.32 - Perbandingan Waktu Simulasi Algoritma CSD Jumlah Robot=7

Radius Sensing	Range	Waktu (dalam ms)				
		Ave	Min	Max	Std Dev	Range
0.5	0.0005	2466.57	781	4297	1314.77	3517
	0.001	1898.12	313	5297	1293.94	4985

Tabel 6.33 - Perbandingan Waktu Simulasi Algoritma CSD Jumlah Robot=10

Radius Sensing	Range	Waktu (dalam ms)				
		Ave	Min	Max	Std Dev	Range
0.5	0.0005	1812.4	391	3640	1002.3	3250
	0.001	887.6	390	1641	318.04	1252

Tabel 6.34 - Perbandingan Waktu Simulasi Algoritma CSD Jumlah Robot=15

Radius Sensing	Range	Waktu (dalam ms)				
		Ave	Min	Max	Std Dev	Range
0.25	0.0005	3046.67	1672	4437	1382.57	2766
	0.001	2832	2656	3063	178.69	408
0.5	0.0005	1362.3	453	4297	1007.76	3845
	0.001	760.9	359	1859	324.85	1501

6.4.3 Algoritma PTN untuk Multisource

Tabel 6.35 - Perbandingan Waktu Simulasi Algoritma PTN Jumlah Robot=5

Radius Sensing	Range	Waktu (dalam ms)				
		Ave	Min	Max	Std Dev	Range
0.25	0.001	5125	5125	5125	-	1
0.5	0.0005	3814.18	1359	7890	1694.95	6532
	0.001	2092	734	4187	1017.93	3454

Tabel 6.36 - Perbandingan Waktu Simulasi Algoritma PTN Jumlah Robot=7

Radius Sensing	Range	Waktu (dalam ms)				
		Ave	Min	Max	Std Dev	Range
0.25	0.0005	5708.5	3781	8594	1620.28	4818
	0.001	4700	2922	7672	1813.65	4751
0.5	0.0005	3041.1	375	9453	2253.78	9079
	0.001	1249.15	485	2907	769.49	2423

Tabel 6.37 - Perbandingan Waktu Simulasi Algoritma PTN Jumlah Robot=10

Radius Sensing	Range	Waktu (dalam ms)				
		Ave	Min	Max	Std Dev	Range
0.25	0.0005	6899.58	3813	12421	2809.46	8609
	0.001	3916.27	1812	8375	1666.3	6564
0.5	0.0005	2088.3	718	4969	1273.21	4252
	0.001	1113.5	438	3937	734.35	3500

Tabel 6.38 - Perbandingan Waktu Simulasi Algoritma PTN Jumlah Robot=15

Radius Sensing	Range	Waktu (dalam ms)				
		Ave	Min	Max	Std Dev	Range
0.25	0.0005	4974.93	2984	10391	1942.92	7408
	0.001	2500.4	1344	6172	1168.53	4829
0.5	0.0005	1328.2	391	6266	5876	1218.13
	0.001	716.35	344	1625	331.22	1282

6.4.4 Algoritma PBL untuk Multisource

Tabel 6.39 - Perbandingan Waktu Simulasi Algoritma PBL Jumlah Robot=5

Radius Sensing	Range	Waktu (dalam ms)				
		Ave	Min	Max	Std Dev	Range
0.5	0.0005	4268.05	1594	7625	1252	6032
	0.001	1970.95	828	3781	823.95	2954

Tabel 6.40 - Perbandingan Waktu Simulasi Algoritma PBL Jumlah Robot=7

Radius Sensing	Range	Waktu (dalam ms)				
		Ave	Min	Max	Std Dev	Range
0.25	0.0005	8749.1	4594	12734	2536.83	8141
	0.001	4896.78	2282	7797	1464.46	5516
0.5	0.0005	2489.95	859	4719	1083.09	3861
	0.001	1618	422	3234	775.55	2813

Tabel 6.41 - Perbandingan Waktu Simulasi Algoritma PBL Jumlah Robot=10

Radius Sensing	Range	Waktu (dalam ms)				
		Ave	Min	Max	Std Dev	Range
0.25	0.0005	6789.15	3188	13266	2401.02	10079
	0.001	4609.47	1891	7016	1467.22	5126
0.5	0.0005	2030.35	500	3953	873.36	3454
	0.001					

Tabel 6.42 - Perbandingan Waktu Simulasi Algoritma PBL Jumlah Robot=15

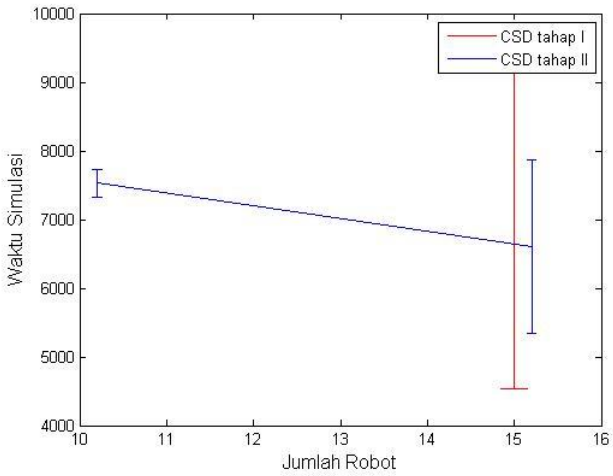
Radius Sensing	Range	Waktu (dalam ms)				
		Ave	Min	Max	Std Dev	Range
0.25	0.0005	4576.7	2547	6500	1170.34	3954
	0.001	3236.7	1453	6329	1142.77	4877
0.5	0.0005	1590.65	359	4360	963.34	4002
	0.001	1385.95	594	2375	442.06	1782

6.5 Perbandingan Kinerja Algoritma

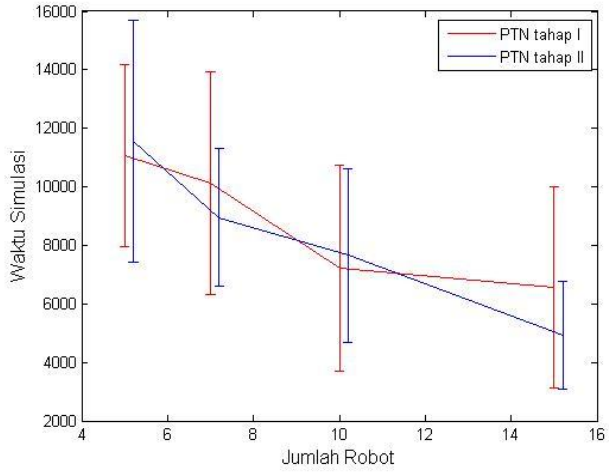
Pada bagian ini akan dibahas perbandingan kinerja algoritma dilihat dari beberapa sudut pandang, berdasarkan evaluasi yang diperoleh pada uji coba sebelumnya.

6.5.1 Pengaruh Posisi Target yang Berbeda

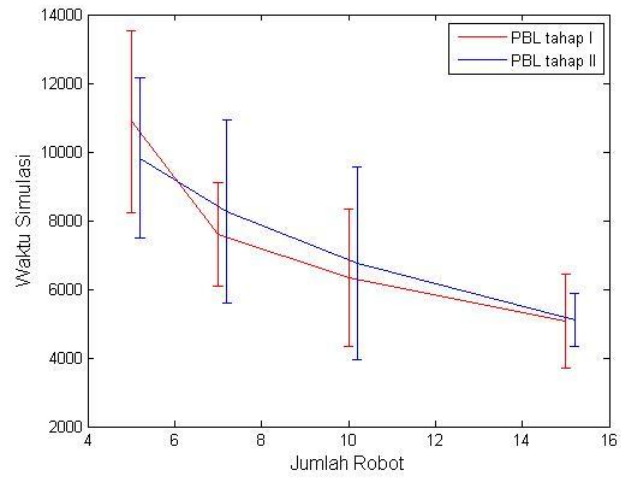
Gambar 6.19 di bawah menunjukkan grafik perbandingan waktu masing-masing algoritma yang sama untuk menemukan solusi pada dua tahap percobaan.



(a)



(b)



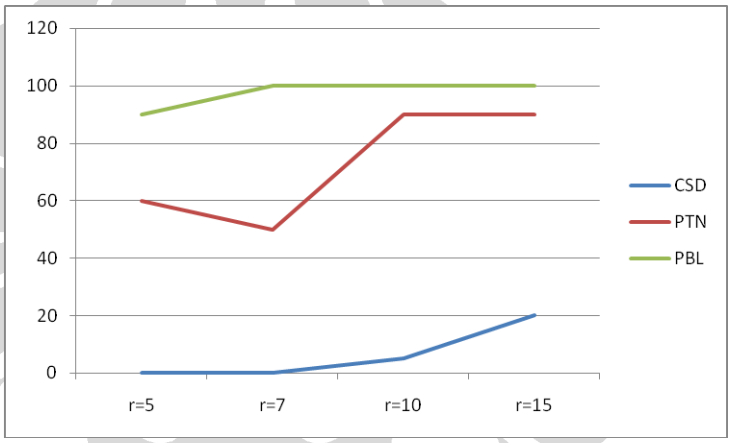
(c)

Gambar 6.19 - Perbandingan Waktu bagi Algoritma untuk Menemukan Solusi

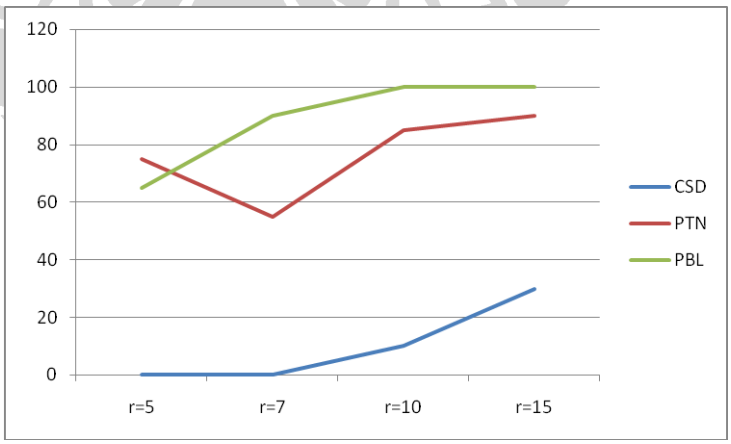
Berdasarkan ketiga grafik di atas, dapat dilihat bahwa perbedaan posisi target pada ruang kerja tidak akan mempengaruhi kinerja algoritma. Hal ini dapat dilihat pada grafik masing-masing algoritma yang tergambar saling berdekatan pada kedua tahap percobaan yang dilakukan.

6.5.2 Perbandingan Tingkat Kesuksesan

Berikutnya akan dilihat grafik perbandingan kinerja antara algoritma pada masing-masing tahap percobaan, seperti yang ditunjukkan oleh gambar 6.20 berikut di bawah ini. Berdasarkan gambar tersebut dapat dilihat bahwa algoritma PBL cenderung memiliki kinerja yang lebih baik dibanding kedua algoritma lainnya. Algoritma PTN juga menunjukkan kinerja yang baik walaupun tingkat kesuksesannya tidak sebaik algoritma PBL. Sementara itu, algoritma CSD menunjukkan kinerja yang buruk dengan rendahnya tingkat kesuksesan solusi permasalahan yang ditemukan dengan menggunakan algoritma ini.



(a)

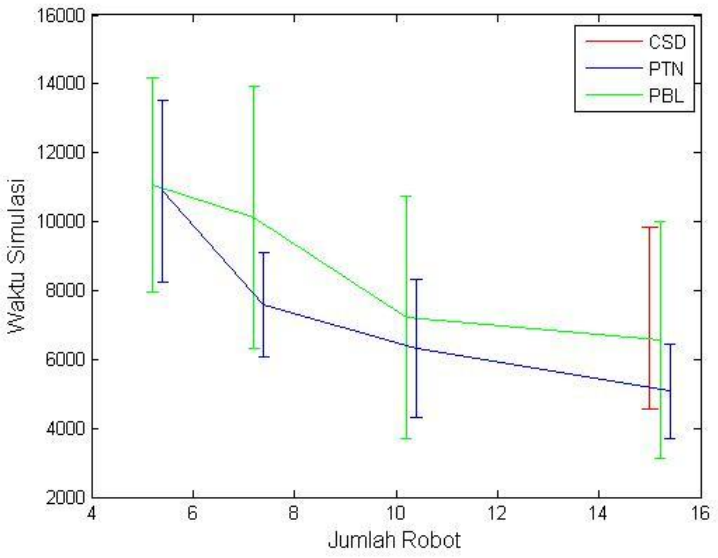


(b)

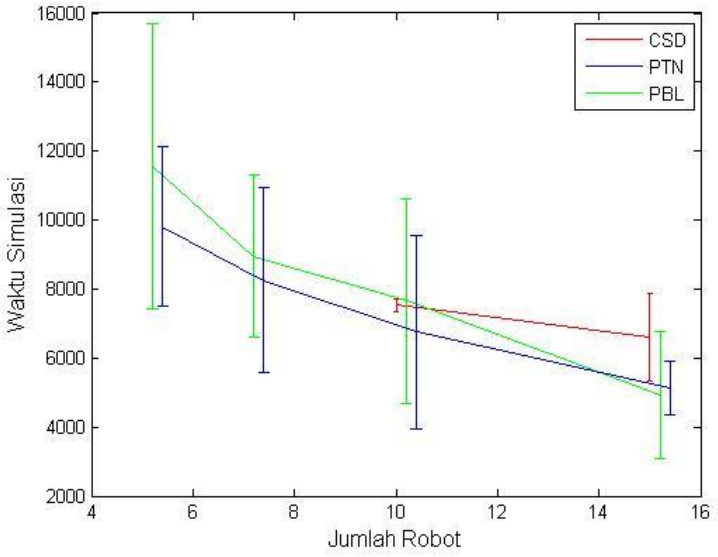
Gambar 6.20 - Kinerja Algoritma pada Kedua Tahap Percobaan

6.5.3 Perbandingan Waktu untuk Menemukan Solusi

Berikutnya akan dilihat perbandingan kinerja ketiga algoritma berdasarkan waktu yang dibutuhkan untuk menemukan solusi.



(a)



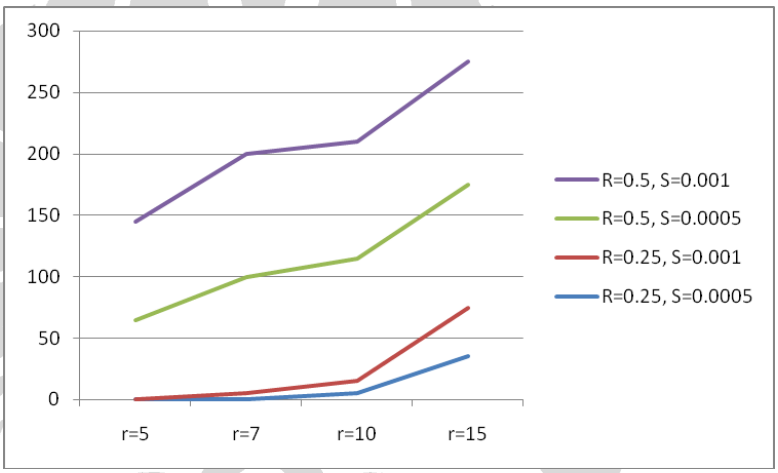
(b)

Gambar 6.21 - Kinerja Algoritma Dilihat dari Waktu untuk Menemukan Solusi

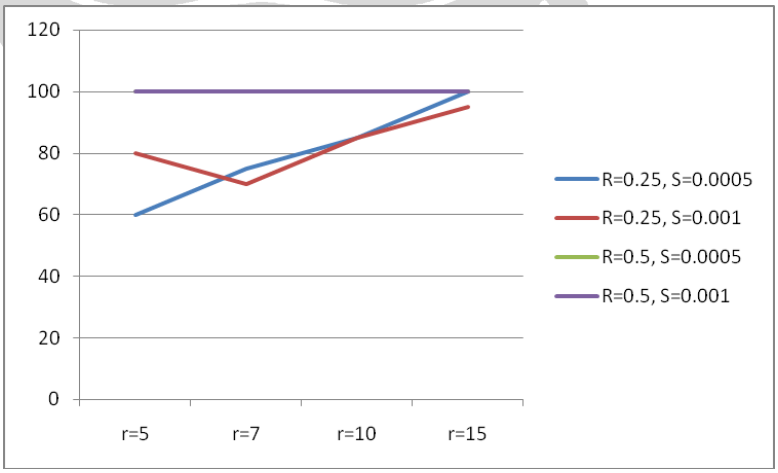
Berdasarkan grafik tersebut dapat dilihat bahwa algoritma PBL merupakan algoritma yang memerlukan waktu paling kecil untuk dapat menemukan solusi permasalahan. Algoritma PTN memerlukan waktu yang sedikit lebih lama, sedangkan algoritma CSD merupakan algoritma yang paling buruk jika kinerja algoritma dilihat berdasarkan waktu yang dibutuhkan untuk menemukan solusi.

6.5.4 Perbandingan Pengaruh Radius Sensing dan Range

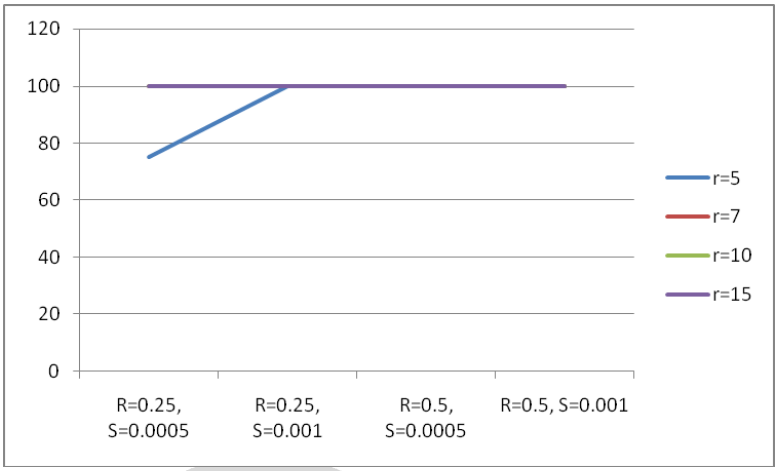
Berikutnya akan dilihat perbandingan pengaruh radius sensing dan range yang berbeda, seperti yang ditunjukkan oleh gambar 6.22 berikut di bawah ini.



(a)



(b)



(c)

Gambar 6.22 - Kinerja Algoritma pada Kedua Tahap Percobaan

Berdasarkan gambar tersebut dapat dilihat bahwa radius sensing area yang lebih jauh serta range robot yang lebih tinggi dapat meningkatkan kinerja dari setiap algoritma. Hal ini dapat terlihat jelas pada grafik perbandingan kinerja algoritma CSD, dimana dengan kenaikan rentang radius dan range tertentu, akan diperoleh peningkatan hasil yang jauh lebih tinggi. Akan tetapi perlu diingat bahwa penggunaan radius sensing area dan range yang lebih besar ini tentunya akan menambah biaya yang diperlukan untuk membangun suatu robot otonom tunggal.