SinarFe7
Seminar Nasional Fortei Regional 7

ISSN (Print) : 2621-3540
ISSN (Online) : 2621-5551

PENERAPAN SISTEM NAVIGASI MOBILE ROBOT PADA TURTLEBOT DENGAN DEPTH SENSOR CAMERA

Joannes Mahestra Robert Aprilino, Petrus Santoso, S. T., M.Sc.
Program Studi Teknik Elektro, Universitas Kristen Petra, Surabaya
robertaprilino@gmail.com, petrus@petra.ac.id

Abstrak*— *This research uses the Turtlebot platform as a mobile robot that has SLAM, Navigation, and Manipulation technology. The purpose of this research is to test the navigation system on Turtlebot that has a depth sensor camera installed.

The test of the sistem is done by taking any data that can be used from the depth sensor camera. After that the test will proceed with making maps with the SLAM algorithm. Maps that have been made will be compared with actual conditions. In the Navigation test, it can be seen that Turtlebot can go to the selected point on a map at without obstacle test for distances of 100 cm, 200cm, and 400cm and the test uses obstacles.

Key words— Mobile robot, Turtlebot, SLAM, Navigation, Depth Sensor Camera, Robot Operating System (ROS).

***Abstrak*— Penelitian ini menggunakan platform *Turtlebot* sebagai mobile robot yang memiliki teknologi *SLAM*, *Navigation*, dan *Manipulation* . Tujuan dari penelitian ini adalah menguji sistem navigasi pada *Turtlebot* yang telah terpasang *depth sensor camera*.**

Pengujian sistem dilakukan dengan mengambil data apa saja yang dapat digunakan dari *depth sensor camera*. Setelah itu pengujian akan dilanjutkan dengan pembuatan peta dengan algoritma SLAM. Peta yang sudah terbuat akan dibandingkan dengan kondisi sebenarnya. Berikutnya pada pengujian *Navigation*, dapat diketahui bahwa Turtlebot dapat berjalan menuju titik yang dipilih pada sebuah peta dalam pengujian tanpa halangan untuk jarak 100 cm, 200cm, dan 400cm dan pengujian menggunakan halangan.

Kata kunci— Mobile robot; Turtlebot; SLAM; Navigation; Depth Sensor Camera; Robot Operating System (ROS).

I. PENDAHULUAN

Mobile Robot adalah mesin yang dikendalikan oleh perangkat lunak yang menggunakan sensor dan teknologi lainnya untuk mengidentifikasi lingkungannya dan bergerak di sekitar lingkungannya. Fungsi dasar *mobile robot* mencakup kemampuan untuk bergerak dan menjelajahi, mengangkut muatan, atau menghasilkan kargo, dan menyelesaikan tugas rumit menggunakan sistem di atas kapal, seperti lengan robot. Sementara penggunaan industri mobile robot sangat populer, terutama di gudang dan pusat distribusi, fungsinya juga dapat diterapkan pada produksi obat-obatan, operasi, bantuan pribadi dan keamanan.[1]

Dua masalah utama dalam *mobile robot* adalah penentuan posisi global dan pelacakan posisi lokal. Sebuah *mobile robot* perlu dilakukan penetapan estimasi posisi global sebagai kemampuan untuk menentukan posisi robot dalam sebuah peta yang telah dipelajari sebelumnya, mengingat informasi lain selain bahwa robot berada di suatu tempat di atasnya. Jika tidak ada peta yang tersedia, banyak aplikasi memungkinkan untuk peta seperti itu dibangun seiring waktu ketika robot menggali lingkungannya. Setelah robot diberi sebuah posisi di dalam peta, pelacakan lokal adalah masalah melacak posisi itu dari waktu ke waktu. Kedua kemampuan ini diperlukan untuk memungkinkan untuk merencanakan dan menavigasi dengan andal di lingkungan yang kompleks. [2]

Pada *mobile robot* terdapat sebuah sistem pemetaan dengan menggunakan *Simultaneous Localization and Mapping* (SLAM). SLAM adalah sebuah algoritma di mana sebuah perangkat mengambil data dari sensor untuk membuat sebuah gambar dari situasi lingkungan sekitar dan mengetahui posisinya di dalam lingkungan tersebut. Sensor menggunakan data visual seperti kamera atau data yang tidak dapat dilihat seperti Sonar, Radar, atau LiDAR dan pengambilan data posisi menggunakan *Inertial Measurement Unit* (IMU). Perangkat menggunakan informasi tersebut untuk menghitung 'estimasi terbaik' dari perangkat tersebut dalam sebuah lingkungan yang dianalisa. Jika posisi dari perangkat dipindah, maka tembok, lantai, atau benda lain yang terbaca oleh sensor akan ikut bergerak sesuai dengan relasi yang telah dihitung sebelumnya dan algoritma SLAM akan mengetahui estimasi dari posisi yang baru. [3]

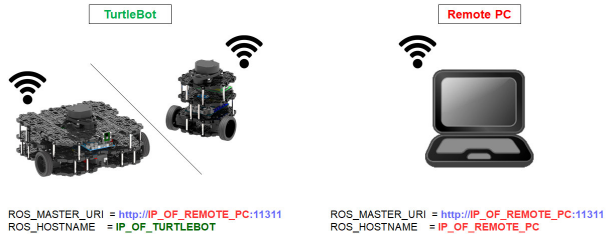
Pada penelitian ini, mobile robot yang akan digunakan adalah Turtlebot 3, dimana Turtlebot 3 dipilih karena memiliki fitur SLAM yang dapat digunakan. Untuk sensor yang digunakan untuk mendeteksi lingkungan akan digunakan *depth sensor camera* yang telah disesuaikan dengan ruang lingkup dimana Turtlebot akan dijalankan. Dengan adanya penelitian ini, akan dapat disimpulkan bahwa sistem navigasi self-driving car dapat diterapkan pada skala yang kecil yang kemudian dapat dikembangkan pada mobil dengan cara yang sederhana.

II. PERANCANGAN SISTEM DAN IMPLEMENTASI

Terdapat dua sistem yaitu sistem yang berjalan pada *Turtlebot* dan sistem yang berjalan pada *PC_Master*. Turtlebot terkoneksi dengan PC yang terhubung pada 1 jaringan *wifi*

ISSN (Print) : 2621-3540
ISSN (Online) : 2621-5551

yang sama dan terdapat pengaturan yang perlu dilakukan pada program */.bashrc* dari ROS yang terpasang di dua perangkat yang berbeda. Untuk melakukan pengaturan koneksi dapat mencermati gambar berikut ini :

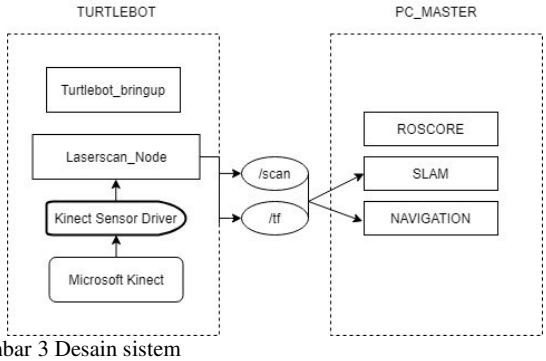


Gambar 1 Konfigurasi IP Address antara Turtlebot dan PC **[4]**

```
export
ROS_MASTER_URI=http://192.168.1.98:11311
export ROS_HOSTNAME=192.168.1.66
```
Gambar 2 Potongan program dari /.bashrc

Setelah mengatur konfigurasi IP Address, maka Turtlebot dengan PC telah terkoneksi dan perintah *roscore* akan dapat dijalankan tanpa ada masalah.

Pada *Turtlebot* terdapat dua node utama yang akan dijalankan, yaitu *Turtlebot_bringup* yang digunakan untuk menyalakan kebutuhan yang berkaitan dengan navigasi gerak dari robot, dan *laserscan_node* yang nanti akan memuat data dari *depth sensor camera* yang telah dapat berfungsi oleh sebuah *driver* yang dipasang pada *raspberry pi 3*.

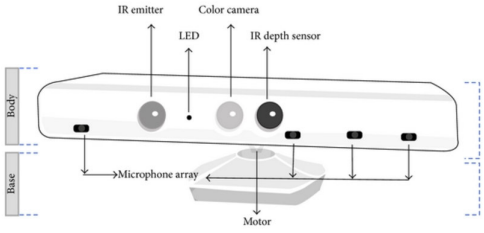


Gambar 3 Desain sistem

Sedangkan pada PC_Master, akan dijalankan server dari ROS menggunakan perintah *roscore*

A. Pemasangan Depth Sensor Camera

Pada penelitian ini, sensor yang digunakan adalah *Microsoft Kinect V1*, dimana sensor *Microsoft Kinect* terdapat *pointcloud data* dan *depth image data* yang pada langkah selanjutnya akan digunakan.



Gambar 4 Komponen pada Kinect V1 **[5]**

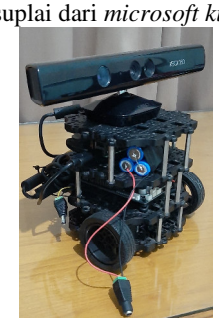
Pada ubuntu terdapat sebuah *driver 'freenect_stack'* yang digunakan untuk menghubungkan kinect dengan raspberry pi. *Freenect_stack* dapat dipasang dengan menjalankan perintah pada terminal Ubuntu sebagai berikut :

$ sudo apt install ros-kinetic-freenect-launch

Setelah driver telah terpasang, sensor akan aktif ketika *node* dari *freenect* dijalankan lewat terminal *ubuntu* dengan perintah sebagai berikut :

$ roslaunch freenect_launch freenect.launch

Microsoft Kinect memerlukan suplai tegangan 12 V dan 1 Ampere dari adaptor agar dapat berfungsi, tetapi adaptor kinect harus terpasang pada stop kontak dan membatasi gerak dari *Turtlebot*. Untuk menggantikan adaptor yang digunakan untuk menyalakan kinect, maka 3 baterai 18650 3.8 V akan digunakan sebagai suplai dari *microsoft kinect*.


Gambar 5 *Depth sensor camera* yang telah terpasang pada *Turtlebot3 burger*

B. Pemrogaman SLAM Turtlebot

Prinsip pada pembuatan peta oleh algoritma SLAM memerlukan sebuah data dari sensor dalam sebuah subscribed topic dan memberikan sebuah nilai output dalam published topic yang nanti akan diproses menjadi sebuah peta. Pada node SLAM yang dijalankan terdapat dua *subscribed topic* yang harus dipenuhi agar algoritma dapat bekerja, yaitu *'/scan'* dan *'/tf'*. Untuk /scan telah didapatkan dari node *depthimage_to_laserscan* yang telah dipublish pada langkah sebelumnya. Untuk */tf* merupakan sebuah topik dari *sensor transform* yang digunakan untuk memperkirakan posisi sensor yang tergabung pada robot ketika proses slam berjalan.

Setelah *subscribed topic* yang diperlukan telah terpenuhi maka node SLAM akan memiliki dua *published topic* utama yaitu */map* dan beberapa *topic* yang dapat dikategorikan

ISSN (Print) : 2621-3540
ISSN (Online) : 2621-5551

sebagai */pose topic*. Untuk published *topic* tersebut dapat dilihat pada display dari hasil mapping yang dapat diakses melalui *'rviz'* lewat *PC_Master*. Topik /map digunakan untuk menyimpan hasil dari *topic /scan* yang dalam bentuk peta yang tergambar sesuai dengan hasil *scanning* pada *depthsensor* yang dijalankan. Hasil dari peta nanti akan disimpan dengan menjalankan node *map_saver*. Untuk pose *topic* digunakan untuk menampilkan posisi sensor pada *display rviz* yang dapat membantu mendeteksi daerah yang belum dideteksi oleh sensor dan daerah yang telah dideteksi oleh sensor secara *real time*.

Berikutnya adalah pemrogaman untuk membuat konfigurasi *sensor transform.* Konfigurasi ini diperlukan untuk mengatur posisi sensor agar menjadi satu dengan badan robot yang digunakan. Program yang digunakan untuk membuat transform tersebut adalah sebagai berikut :

```
#include <ros/ros.h>
#include <tf/transform_broadcaster.h>
int main(int argc, char** argv){
  ros::init(argc, argv,
"robot_tf_publisher");
  ros::NodeHandle n;
  ros::Rate r(100);
  tf::TransformBroadcaster broadcaster;
  while(n.ok()){
    broadcaster.sendTransform(
      tf::StampedTransform(
        tf::Transform(tf::Quaternion(0,
0, 0, 1), tf::Vector3(0.0, 0.0, 0.0)),
        ros::Time::now(),"base_link",
"camera_link"));
    r.sleep();
  }
}
```

Gambar 6 Program sensor transform

Untuk melakukan langkah pembuatan peta, ada beberapa konfigurasi pada program yang perlu dilakukan. Langkah pertama adalah file launch dari transform sensor akan digabung dengan file launch dari package turtlebot3_bringup yang digunakan untuk mengaktifkan robot. Tidak hanya digunakan untuk menjalankan node transform, file launch tersebut juga akan digunakan untuk menjalankan node depthimage_to_laserscan. Terdapat 2 node yang akan dijalankan, yaitu depthimage_to_laserscan dan tf_broadcaster. Pada bagian <node pkg="robot_setup_tf" type="tf_broadcaster" name="tf_brodcaster" output="screen"> menunjukkan bahwa terdapat file dengan nama 'tf_brodcaster' yang berada didalam package 'robot_setup_tf' yang berisi program dari sensor transform yang telah dibuat sebelumnya.

```
<launch>
   <node pkg="depthimage_to_laserscan"
name="depthimage_to_laserscan"
type="depthimage_to_laserscan">
        <remap from="image"
to="camera/depth/image_raw"/>
                <param
name="output_frame_id"
value="camera_link" />
        <param name="range_min"
value="0.45" />
        </node>
 <node pkg="robot_setup_tf"
type="tf_broadcaster"
name="tf_brodcaster" output="screen">
 </node>
</launch>
```

Gambar 7 Program pada file *robot_configuration.launch*

Langkah berikutnya adalah menambahkan *file robot_configuration.launch* ke dalam *file turtlebot3_bringup.launch* agar konfigurasi pada sensor akan aktif bersamaan dengan aktivasi robot.

```
 <include file="$(find
turtlebot3_bringup)/launch/robot_configur
ation.launch">
  </include>
```

Gambar 8 Potongan program pada turtlebot3_bringup.launch

Setelah mengamati *topic* yang *dibroadcast* oleh konfigurasi *tf,* maka selanjutnya adalah melakukan pengaturan pada *file launch* dari SLAM agar dapat menggunakan *depth sensor camera* yang telah terpasang.

Turtlebot memiliki program *launch turtlebot3_gmapping* yang ada pada *package turtlebot3_slam.* Pada *file launch* tersebut terdapat bagian "set_base_frame" yang perlu diganti namanya sesuai dengan konfigurasi 'tf' yang telah dilakukan sebelumnya. Karena pada langkah sebelumnya frame dari sensor bernama camera_link yang telah berganti base_link, maka pada bagian "set_base_frame" akan diganti dengan 'base_link'.

```
<launch>
  <!-- Arguments -->
  <arg name="model" default="$(env
TURTLEBOT3_MODEL)" doc="model type
[burger, waffle, waffle_pi]"/>
  <arg name="set_base_frame"
default="base_link"/>
  <arg name="set_odom_frame"
default="odom"/>
  <arg name="set_map_frame"
default="map"/>
```

Gambar 9 Potongan program turtlebot3_gmapping

C. Pemrogaman Navigation Turtlebot

Navigation Stack dalam ROS membutuhkan konfigurasi yang harus dipenuhi agar sistem navigasi robot dapat berjalan dengan baik. Pada *Navigation Stack ROS, Turtlebot3* telah

ISSN (Print) : 2621-3540
ISSN (Online) : 2621-5551

memiliki *move_base* dan *base controller* yang digunakan untuk menjalankan robot dan *turlebot3* juga sudah memiliki *map_server, odometry source*, dan *node amcl* yang dapat digunakan. Maka sistem *turtlebot* yang saat ini memerlukan konfigurasi pada sistem sensor agar *navigation stack* dapat bekerja. Konfigurasi sensor yang diperlukan adalah *sensor transform* ('/tf') dan *sensor sources* yang sudah dijelaskan pada pemrogaman SLAM turtlebot.

Terdapat file program amcl.launch yang berisi algoritma *Adaptive Monte Carlo Localization* yang berfungsi untuk untuk menentukan posisi robot dalam sebuah peta atau proses *Localization*. Parameter *base_frame_id* yang merupakan sebuah *transform* dari sensor akan memberikan informasi mengenai gerak robot secara *translasi* dan *rotasi*, sehingga posisi *odometry robot* akan berubah. Pada proses ini, algoritma AMCL akan menggunakan data dari *base_frame_id* untuk menentukan posisi robot pada sebuah peta. Oleh karena itu, AMCL akan melakukan estimasi posisi robot terhadap *map_frame* yang merupakan *topic* yang berisi data dari peta yang digunakan pada proses navigation. Untuk *base_frame_id* yang digunakan harus berisi sensor *transform* yang digunakan, yaitu pada sudah ditentukan bahwa data hasil transform sensor akan dipublish melalui *topic 'base_link'* yang sesuai dengan konfigurasi yang sudah dilakukan sebelumnya.

Setelah melakukan pengaturan pada *file amcl.launch*, pada langkah berikutnya adalah melakukan pengaturan pada *file parameter costmap_common* dan *local_costmap* dari *navigation stack* yang berada dalam *package* yang sama pada folder param. Untuk *costmap_common*, pada variable 'sensor_frame' perlu di isi dengan topic 'camera_link'. Hal ini dapat dilihat pada penjelasan dibagian driver sensor dimana pada tampilan *rqt graph, topic* yang muncul sebagai letak sensor adalah *camera_link*. Maka *sensor_frame* yang digunakan pada parameter *costmap_common* dari konfigurasi navigation turtlebot adalah *camera_link*.

```
scan: {sensor_frame: camera_link,
data_type: LaserScan, topic: scan,
marking: true, clearing: true}
```

Gambar 10 Potongan kode file *parameter costmap_common_params_burger.yaml*

Untuk *local_costmap*, Pada bagian *'robot_base_frame'* akan diisi dengan *topic 'base_link'*, karena menyesuaikan program konfigurasi *sensor transform*

```
robot_base_frame: base_link
```

Gambar 11 Kode file *parameter local_costmap_params.yaml*

III. PENGUJIAN SISTEM

A. Pengujian Sensor

Untuk memastikan bahwa *depth sensor camera* siap digunakan bersama dengan *turtlebot*, sensor perlu diuji apakah *driver* telah dapat berjalan dan dapat digunakan untuk mengambil data pada sensor, dan perlu untuk mencatat data apa yang dihasilkan oleh sensor yang dapat digunakan untuk proses SLAM dan *Navigation turtlebot*.

Terdapat 2 *file* program yang akan dijalankan, program pertama adalah *freenect.launch* yang terdapat pada *package* bernama *freenect_launch*. Program ini nantinya yang berfungsi untuk mendeteksi koneksi sensor dengan *raspberry pi* dan mengaktifkan perangkat yang ada pada sensor. Namun sebelum menyalakan program tersebut, tentu konfigurasi koneksi antara *Turtlebot* dengan *PC_Master* sebagai *server* dari ROS sudah diatur dengan benar.

Ketika menyalakan driver freenect.launch, terdapat informasi *"number devices connected : 1"* dan perangkat bernama *Xbox NUI Camera* from *Microsoft* sudah berhasil terpasang pada *raspberry pi*. Informasi berikutnya adalah driver bekerja untuk menyalakan fitur yang ada pada sensor agar driver dapat mengambil data dari sensor. Pada akhirnya, *driver* dapat mengenali sensor dan mengambil data dari sensor. Data yang diambil terdapat pada *topic rgb_frame_id* dan *depth_frame_id* yang nanti dapat digunakan pada tampilan *rviz*.Ketika *depthimage_to_laserscan* sudah dijalankan dengan *file laserscan.launch*, maka langkah berikutnya adalah melihat pembacaan sensor pada sebuah lingkungan tertentu, untuk menampilkan data pada sensor dapat menggunakan aplikasi *rviz* untuk melihat tampilan yang terjadi.

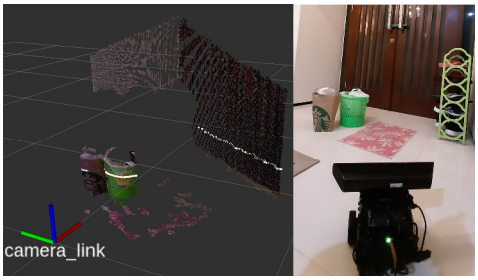
Dalam aplikasi *rviz*, terdapat daftar *topic* yang ditampilkan dan hasil tampilan berdasarkan *topic* yang digunakan. Setiap *topic* yang digunakan memuat masing – masing data yang berbeda pada tampilan. Untuk pengujian sensor saat ini, *topic* yang digunakan adalah sebagai berikut :

- **camera_link** : Digunakan untuk menentukan *frame sensor* yang digunakan, diatur pada bagian *Global Options*. Jika *Fixed Frame* diganti atau tidak diisi maka tampilan tidak akan memunculkan apapun karena *rviz* tidak menggunakan *frame* dari sensor yang sudah terpasang.
- **/camera/depth_registered/points** : Memunculkan data hasil pembacaan *PointCloud* pada sensor. Hal ini ditunjukkan dengan adanya ruangan dan barang yang terdeteksi sehingga pada *display rviz* dapat terlihat.
- **camera_link(TF)** : memunculkan letak sensor dalam sumbu x,y, dan z
- **/scan** : Memunculkan daerah yang terbaca oleh laserscan, *topic* ini muncul karena *depthimage_to_laserscan* yang sudah dijalankan. *Topic* ini dapat dilihat pada display ditandai dengan adanya garis warna putih yang membatasi barang dan dinding yang ada pada lingkungan yang terdeteksi.

Ketika sensor yang berada dalam kondisi diam tidak mengalami perubahan display atau proses scanning telah selesai, maka hasil display yang telah terbuat pada *rviz* dapat

ISSN (Print) : 2621-3540
ISSN (Online) : 2621-5551

dibandingkan dengan kondisi lingkungan nyata tempat sensor berada. Berikut ini adalah perbandingan hasil pembacaan sensor dengan metode pointcloud dengan kondisi lingkungan nyata :

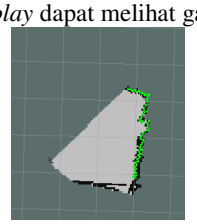


Gambar 12 Letak barang dan sensor pada keadaan nyata

Pada display *rviz*, dapat terlihat bahwa ada bagian kosong dari dinding yang tidak terbaca karena tertutup oleh benda di depannya, dan hal ini didukung oleh data *laserscan* yang memunculkan garis putih pada beda tersebut. Selain itu, terdapat benda yang tidak terdeteksi oleh *laserscan* karena letak dari benda yang berada di bawah range dari *laserscan* dan tidak di hadapan sensor secara langsung. Karena *range* pada sensor yang dibatasi oleh sudut tertentu, maka sensor tidak dapat membaca benda di sebelah kanan dan sebelah kanan.

B. Pembuatan pada proses SLAM

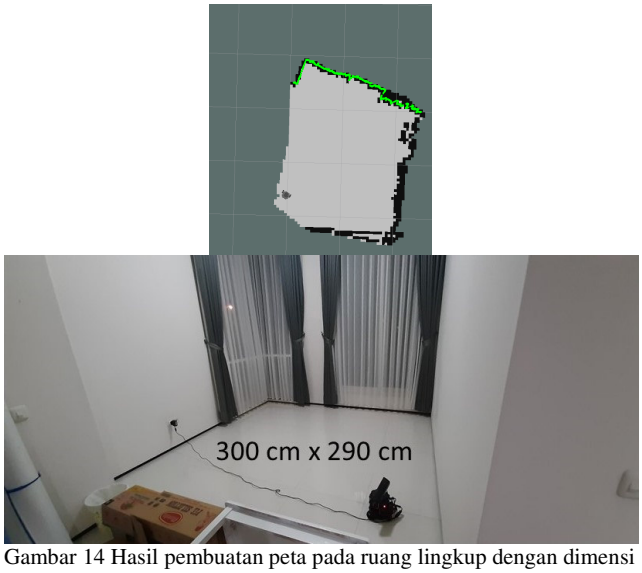
Ketika perintah *'roslaunch turtlebot3_slam turtlebot3_slam.launch slam_methods:=gmapping'* dijalankan oleh *PC Master* sebagai server dari ROS, maka akan muncul tampilan *rviz* pada display *PC Master*. Untuk melihat tampilan yang terjadi pada *display* dapat melihat gambar berikut ini :



Gambar 13 Tampilan awal *rviz* ketika *turtlebot3_slam.launch* dijalankan

Untuk langkah berikutnya, agar peta dari ruang lingkup gerak robot dapat terbentuk maka robot perlu digerakkan dengan menyesuaikan bagian dari ruangan yang belum terdeteksi oleh *rviz*. Cara yang dilakukan untuk menggerakkan robot adalah mengaktifkan fitur *turtlebot3_teleop* dengan menjalankan perintah *'roslaunch turtlebot3_teleop turtlebot3_teleop_key.launch'*. Setelah itu, maka robot akan dapat digerakkan menggunakan keyboard sesuai dengan petunjuk yang ada pada tampilan terminal ubuntu.

Berikut ini adalah hasil peta dari menggerakkan robot menyesuaikan bagian yang kosong dari *rviz* :



Gambar 14 Hasil pembuatan peta pada ruang lingkup dengan dimensi 300 cm x 290 cm

Setelah peta sudah terbuat dan semua dinding pada ruang lingkup gerak robot telah terdeteksi, maka peta dapat disimpan agar dapat digunakan pada proses *navigation.* Untuk menyimpan peta yang terbuat oleh *turtlebot3_slam* dapat menggunakan perintah *'rosrun map_server map_saver'*. Pembuatan peta ini dipengaruhi oleh beberapa faktor parameter yang ada pada program dari metode *gmapping* yang digunakan. Untuk lebih jelas dapat melihat potongan program dari *turtlebot3_gmapping.launch* berikut ini :

```
<param name="linearUpdate" value="1"/>
<param name="angularUpdate" value="0.2"/>
<param name="temporalUpdate"
value="0.5"/>
<param name="resampleThreshold"
value="0.5"/>
<param name="particles" value="100"/>
```

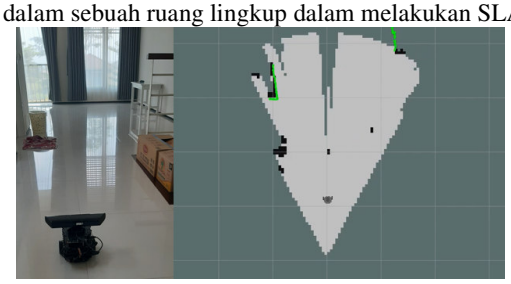
Gambar 15 Potongan program dari *turtlebot3_gmapping.launch*

Jika melihat isi kode dari turtlebot3_gmapping.launch, terdapat 3 parameter yang diberi nama *'linearUpdate', 'angularUpdate'*, dan *'particles'*. Menurut Kamarudin, Kamarulzaman, Mamduh, dan Syed Muhammad dalam artikel jurnal *Performance Analysis of the Microsoft Kinect Sensor for 2D Simultaneous Localization and Mapping (SLAM) Techniques* menyatakan bahwa 3 parameter tersebut adalah yang paling mempengaruhi bentuk dari peta yang dibuat oleh kinect. Untuk parameter *'particles'* merupakan angka yang menunjukkan probabilitas dari sebuah peta dapat terbentuk. Semakin tinggi angka dari parameter maka probabilitas dari pembuatan peta yang terbentuk akan semakin besar. Sedangkan untuk parameter *'linearUpdate'* dan *'angularUpdate'* adalah parameter yang menunjukkan nilai gerak translasi dan gerak rotasi yang dibutuhkan sebelum

ISSN (Print) : 2621-3540
ISSN (Online) : 2621-5551

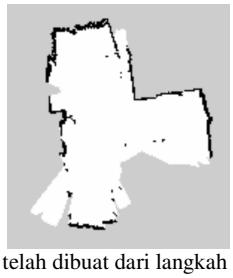
melakukan proses *scan* [6]. Penelitian ini pada akhirnya dapat digunakan sebagai pedoman untuk menghasilkan peta yang sesuai dengan lingkungan aslinya dengan melakukan penggantian nilai pada parameter – parameter secara *trial and error.*

Pengujian berikutnya pada robot dilakukan dengan membuat peta pada skala ruangan yang lebih besar dan menggunakan prosedur yang sama seperti yang sudah dilakukan pada langkah sebelumnya. Berikut ini adalah gerak robot dalam sebuah ruang lingkup dalam melakukan SLAM :



Gambar 16 Gerak *Turtlebot* ketika melakukan SLAM pada sebuah ruang lingkup

Setelah turtlebot sudah selesai mendeteksi semua benda dan dinding yang ada, maka hasil peta yang telah terbuat adalah sebagai berikut :



Gambar 17 Peta yang telah dibuat dari langkah SLAM *turtlebot* sebelumnya

Untuk melihat apakah peta ini sudah sesuai dengan kondisi ruang lingkup yang sebenarnya, dapat dilihat gambar mengenai ruangan yang digunakan sebagai ruang lingkup robot pada kondisi sebenarnya berikut ini :

Gambar 18 Kondisi ruang lingkup SLAM robot pada kondisi sebenarnya

Jika membandingkan hasil SLAM dengan kondisi sebenarnya maka akan ditemukan beberapa benda yang tidak dapat terdeteksi dengan baik. Pada bagian besi pengaman pada tangga rumah tidak dapat terdeteksi oleh sensor dari *turtlebot* karena memiliki bidang kosong, sehingga oleh sensor dianggap tidak memiliki halangan. Hal ini dapat dilihat pada peta yang terbuat akan terdapat ruang putih yang merupakan ruang kosong yang terbaca oleh sensor. Selain itu, robot tidak dapat mendeteksi benda secara utuh seperti bentuk benda pada kondisi sebenarnya. Sebagai contoh, pada Gambar 23 terdapat beberapa benda seperti kardus pembatas, tong sampah, dan meja tetapi pada peta hanya dapat terbaca sebagai titik kecil saja. Hal ini terjadi karena kemampuan *Microsoft Kinect* sebagai sensor memiliki sudut pindai tertentu pada jarak tertentu, maka untuk mendeteksi beberapa benda tersebut perlu menggerakkan robot sedekat mungkin dengan benda agar dapat mendeteksi secara utuh dan kelihatan pada peta yang terbuat.

C. Pengujian pada proses Navigation

Seperti yang sudah dijelaskan pada bab 3 bagian pemrogaman *navigation stack* pada *Turtlebot 3*, Sistem *navigation* pada turtlebot terdapat pada program *turtlebot3_navigation.launch* yang berada dalam *package turtlebot3_navigation*. Untuk menjalankan program tersebut digunakan perintah *'roslaunch turtlebot3_navigation turtlebot3_navigation.launch map_file:=$HOME/map.yaml'* yang dijalankan pada *terminal* dari *PC Master*. Untuk bagian *'map_file'* diisi dengan lokasi tempat penyimpanan peta yang ingin digunakan sebagai ruang lingkup gerak.

Terdapat dua skenario pengujian yaitu pengujian tanpa halangan dan pengujian dengan halangan. Semua pengujian bertujuan untuk menjalankan robot agar dapat berjalan menuju titik yang telah dituju menggunakan fitur *rviz '2D Nav Goal'*.

Untuk pengujian tanpa halangan akan dilakukan selama beberapa kali dan akan diamati hasil waktu yang ditempuh robot dari titik awal menuju titik akhir tujuan robot. Berikut ini adalah titik awal dan tujuan robot pada pengujian untuk jarak 100 m :

Gambar 19 Rute robot untuk pengujian jarak 100cm

Setelah robot dijalankan beberapa kali dengan menggunakan rute seperti Gambar 24, berikut ini adalah hasil yang didapatkan :

ISSN (Print) : 2621-3540
ISSN (Online) : 2621-5551

Tabel 1 Hasil pengujian pada rute robot Gambar 24

Pengujian	Waktu yang ditempuh
1	21 Detik
2	23 Detik
3	8 Detik
4	10 Detik
5	9 Detik

Berikutnya adalah pengujian robot pada jarak 200cm yang memiliki rute seperti berikut ini :

Gambar 20 Rute robot untuk pengujian jarak 200cm

Setelah robot dijalankan beberapa kali dengan menggunakan rute seperti Gambar 25, berikut ini adalah hasil yang didapatkan :

Tabel 2 Hasil pengujian pada rute robot Gambar 25

Pengujian	Waktu yang ditempuh
1	23 detik
2	22 detik
3	12 detik
4	13 detik
5	12 detik

Berikutnya adalah pengujian robot pada jarak 400cm yang memiliki rute sebagai berikut ini :

Gambar 21 Rute robot untuk pengujian jarak 400cm

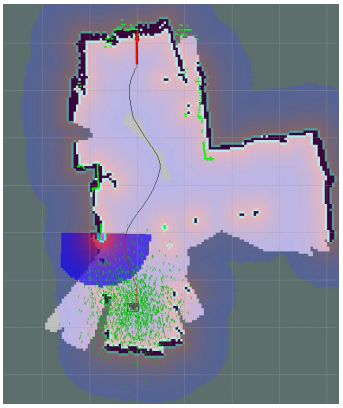
Setelah robot dijalankan beberapa kali dengan menggunakan rute seperti Gambar 26, berikut ini adalah hasil yang didapatkan :

Tabel 3 Hasil pengujian pada rute robot Gambar 26

Pengujian	Waktu yang ditempuh
1	56 detik
2	33 detik
3	28 detik
4	26 detik
5	25 detik

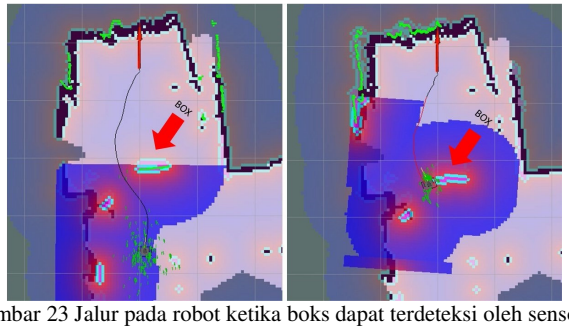
Dengan melihat hasil pengujian pada Tabel 1, Tabel 2, dan Tabel 3 hasil yang didapatkan memiliki sebuah konsistensi. Pada pengujian pertama dan kedua, waktu yang ditempuh hampir sama dan memiliki waktu yang lebih lama daripada pengujian ketiga sampai kelima. Untuk pengujian ketiga sampai kelima, waktu yang ditempuh robot untuk menuju ke titik yang dituju dengan tepat memiliki waktu yang hampir sama. Maka dengan pengujian ini dapat dipastikan bahwa robot perlu waktu untuk melakukan *update* pada peta dalam melakukan *localization* setidaknya dalam 2 kali pengujian gerak robot pada rute yang sama, setelah itu gerak robot akan mengalami sebuah konsistensi.

Pengujian berikutnya untuk pengujian menggunakan halangan dengan metode yang sama. Dengan mengikuti prosedur menjalankan fitur navigation robot, maka robot akan digerakkan menuju sebuah titik seperti pada tampilan *rviz* berikut :



Gambar 22 Titik tujuan akhir dan jalur awal yang akan dilalui robot

Pada tampilan *rviz*, dapat dilihat bahwa sensor tidak dapat mendeteksi boks yang ada pada ruangan karena *scan* dari sensor tidak dapat menjangkau boks, sehingga jalur yang terbuat masih seolah – olah tidak ada boks di depannya. Ketika robot mulai bergerak mendekati boks, maka pada tampilan *rviz* akan muncul hasil scan dari sensor yang menunjukkan terdapat sebuah halangan pada jalur yang terbuat di awal. Untuk lebih jelas dapat melihat tampilan *rviz* berikut ini :



Gambar 23 Jalur pada robot ketika boks dapat terdeteksi oleh sensor

ISSN (Print) : 2621-3540
ISSN (Online) : 2621-5551

Dapat dilihat pada Gambar 29 jalur yang terbuat berubah yang pada awalnya garis hitam berbelok ke kanan akan berubah menjadi berbelok ke kiri karena sensor telah mendeteksi adanya boks yang berada di tengah jalur robot awalnya. Hal ini menandakan bahwa jika terjadi perubahan pada peta, sensor akan melakukan scan dan memperbaharui informasi pada peta yang digunakan, sehingga jalur yang digunakan menjadi berubah. Setelah jalur yang baru telah terbuat, maka robot akan bergerak mengikuti jalur yang baru dan tidak menabrak boks yang ada. Berikut ini adalah hasil yang didapatkan pada kondisi robot sebenarnya :

Gambar 24 Gerak robot pada keadaan sebenarnya

IV. KESIMPULAN

1. Dari hasil pengujian *Depth Sensor Camera*, penggunaan *driver freenect_launch* dapat digunakan untuk mengambil data pada *Microsoft Kinect*. Data yang dapat diambil adalah dalam bentuk pointcloud 3 Dimensi. Selain itu, penggunaan *package depthimage_to_laserscan* dapat digunakan dengan merubah data yang diambil oleh *driver freenect_launch*. Data hasil *depthimage_to_laserscan* dapat ditampilkan secara bersamaan dengan data dari *driver freenect_launch* pada tampilan *rviz*.
2. Dari hasil pengujian pada proses SLAM, pembuatan peta masih terdapat bagian yang tidak dapat terdeteksi oleh sensor dan bentuk dari peta yang dibentuk mengikuti arah hadap dari robot. Hal ini terjadi karena *Microsoft Kinect* memiliki sudut pandang tertentu dan terdapat dinding pembatas yang memiliki tinggi yang tidak sama dengan dinding lainnya. Parameter yang digunakan untuk membuat peta yang ditampilkan sebelumnya menggunakan ‘particles = 30’, ‘linearUpdate = 1.0’, dan ‘angularUpdate = 0.2’.
3. Dari hasil pengujian pada proses *Navigation*, robot hanya dapat berjalan dengan tepat pada 1 peta ketika posisi pada robot telah ditetapkan dengan fitur *2D Pose Estimate* pada *rviz* ketika *turtlebot3_navigation* dijalankan. Pengujian navigation ini juga berhasil ketika robot digunakan untuk mendeteksi benda asing selain dalam peta yang telah terbentuk sebelumnya

V. DAFTAR PUSTAKA

[1] M. Rouse, “What is a mobile robot? Definition from WhatIs.com.,” 2019. [Online]. Available: https://internetofthingsagenda.techtarget.com/definition/mobile-robot-mobile-robotics. [Accessed: 20-Dec-2019].

[2] L. Chen, P. Sun, G. Zhang, J. Niu, and X. Zhang, “Fast monte carlo localization for mobile robot,” *Commun. Comput. Inf. Sci.*, vol. 144 CCIS, no. PART 2, pp. 207–211, 2011, doi: 10.1007/978-3-642-20370-1_34.

[3] “What is SLAM? - GeoSLAM.” [Online]. Available: https://geoslam.com/what-is-slam/. [Accessed: 30-Oct-2019].

[4] “TurtleBot3.” [Online]. Available: https://emanual.robotis.com/docs/en/platform/turtlebot3/navigation/#send-navigation-goal. [Accessed: 10-Jul-2020].

[5] H. H. Pham, T. L. Le, and N. Vuillerme, “Real-time obstacle detection system in indoor environment for the visually impaired using microsoft kinect sensor,” *J. Sensors*, vol. 2016, no. January, 2016, doi: 10.1155/2016/3754918.

[6] K. Kamarudin, S. M. Mamduh, A. Y. Md Shakaff, and A. Zakaria, “Performance analysis of the microsoft kinect sensor for 2D simultaneous localization and mapping (SLAM) techniques,” *Sensors (Switzerland)*, vol. 14, no. 12, pp. 23365–23387, 2014, doi: 10.3390/s141223365.