ภาคผนวก

ภาคผนวก ก

คู่มือการติดตั้งการประยุกต์ใช้ระบบปฏิบัติการหุ่นยนต์สำหรับรถขนส่งสินค้าอัตโนมัติในโรงงานอุตสาหกรรม

นิยามศัพท์

ตารางที่ ก.1 การให้ความหมายคำเฉพาะที่ใช้ในการวิจัย

ตัวอักษรย่อ	ย่อมาจาก	อธิบายคำศัพท์
ROS	Robot Operating System	ซอฟต์แวร์สำหรับการเขียนโปรแกรมหุ่นยนต์
TF	Transform Configuration	ระบบพิกัดที่ใช้ในการอ้างอิงถึงจุดใดๆที่ต้องการสังเกต
SLAM	Simultaneous Localization and Mapping	การสร้างแผนที่และการค้นหาตำแหน่งของตนเองแบบฉับพลัน
AMCL	Adaptive Monte Carlo Localization	การหาตำแหน่ง ณ ปัจจุบันของหุ่นยนต์
Odometry	Odometry	การวัดการเคลื่อนที่ของหุ่นยนต์ในระบบพิกัดเฉพาะของตนเอง
Costmap	Costmap	การหลบหลีกสิ่งกีดขวาง
Move Base	Move Base	การหาเส้นทางการเคลื่อนที่ไปสู่ปลายทางให้กับหุ่นยนต์
Planner	Planner	เป็นอัลกอริทึมที่ทำหน้าที่ในการค้นหาเส้นทาง
Lidar	Light Detection and Ranging	เป็นเซ็นเซอร์ชนิดหนึ่ง
แผนที่สมบูรณ์	-	การสร้างแผนที่ครอบคลุมจุดหมายปลายทางทั้งหมดที่ต้องการให้รถเคลื่อนที่

คู่มือการติดตั้ง

ติดตั้งระบบปฏิบัติการหุ่นยนต์ (ROS) บน Ubuntu 16.04 LTS หรือ Ubuntu 18.04 LTS

ขั้นตอนที่ 1 ตรวจเช็คเวอร์ชั่นของ Ubuntu

```
$ lsb_release -a
```

ขั้นตอนที่ 2 ติดตั้งระบบปฏิบัติการหุ่นยนต์(ROS)

```
$ sudo sh -c ‘echo "deb http://packages.ros.org/ros/ubuntu $(lsb_release -sc) main" > /etc/apt/sources.list.d/ros-latest.list’
$ sudo apt-key adv --keyserver hkp://ha.pool.sks-keyservers.net:80 --recv-key 421C365BD9FF1F717815A3895523BAEEB01FA116
$ sudo apt-get update
```

สำหรับผู้ใช้งานที่ใช้ Ubuntu 16.04 LTS `$ sudo apt-get install ros-kinetic-desktop-full`

สำหรับผู้ใช้งานที่ใช้ Ubuntu 18.04 LTS `$ sudo apt-get install ros-melodic-desktop-full`

```
$ sudo rosdep init
$ rosdep update
```

สำหรับผู้ใช้งานที่ใช้ Ubuntu 16.04 LTS `$ echo "source /opt/ros/kinetic/setup.bash" >> ~/.bashrc source ~/.bashrc`

สำหรับผู้ใช้งานที่ใช้ Ubuntu 18.04 LTS `$ echo "source /opt/ros/melodic/setup.bash" >> ~/.bashrc source ~/.bashrc`

```
$ source ~/.bashrc
$ sudo apt-get install python-rosinstall python-rosinstall-generator python-wstool build-essential
```

ติดตั้งอาดูโน่ (Arduino)

```
sudo apt-get install ros-indigo-rosserial-arduino
sudo apt-get install ros-indigo-rosserial
```

ติดตั้งไลบรารีระบบปฏิบัติการหุ่นยนต์ (ROS)

```
cd <sketchbook>/libraries
rm -rf ros_lib
rosrun rosserial_arduino make_libraries.py .
```

ภาคผนวก ข

โปรแกรมการทำงานของระบบปฏิบัติการหุ่นยนต์สำหรับรถขนส่งสินค้าอัตโนมัติในโรงงานอุตสาหกรรม

โปรแกรมการทำงานของอาดูโน่ (Arduino)

```
#include <ros.h>
#include <std_msgs/String.h>
#include <stdlib.h>
#include <geometry_msgs/Twist.h>
#define MT1_IN1 4
#define MT1_IN2 5
#define MT1_PWM 6
#define MT2_IN1 7
#define MT2_IN2 8
#define MT2_PWM 9

double CmdSpd[2] = {0,0};
double WheelSeparation = 0.44;
double girth = 0.38 ;
double maxRPS = 3 ;
int lpwm=0,rpwm=0 ;

void velCallback(const geometry_msgs::Twist&CVel){
    double vel_x = CVel.linear.x , vel_z = CVel.angular.z; double r_vel = 0.0 , l_vel =
0.0;
    if      (vel_x == 0) { r_vel = vel_z * WheelSeparation/2.0; l_vel = -r_vel; }
    else if (vel_z == 0) { l_vel = r_vel = vel_x;
    else                    { l_vel = vel_x - vel_z * WheelSeparation / 2.0;
                              r_vel = vel_x + vel_z * WheelSeparation / 2.0; }

    if ( l_vel >= 0 ) lpwm = constrain((( 255 * l_vel)/(girth * maxRPS * 1.00)), 0,255);
    else              lpwm = constrain(((-255 * l_vel)/(girth * maxRPS * 1.00)), 0,255);
    if ( r_vel >= 0 ) rpwm = constrain((( 255 * r_vel)/(girth * maxRPS * 1.8)), 0,255);
    else              rpwm = constrain(((-255 * r_vel)/(girth * maxRPS * 1.8)), 0,255);
  CmdSpd[0] = l_vel;      CmdSpd[1] = r_vel;
```

```
}
ros::NodeHandle nh;   ros::Subscriber<geometry_msgs::Twist> sub("cmd_vel",
&velCallback);
geometry_msgs::Twist  msg;  ros::Publisher Pub ("PWM", &msg);

void motorGo( ){
   if        ( CmdSpd[0] > 0){
      if     ( CmdSpd[1] > 0)  { digitalWrite(MT1_IN1,1); digitalWrite(MT1_IN2,0);
                                 digitalWrite(MT2_IN1,1); digitalWrite(MT2_IN2,0);
                                 analogWrite(MT1_PWM,lpwm);
                                 analogWrite(MT2_PWM,rpwm);}
      else if( CmdSpd[1] < 0){  digitalWrite(MT1_IN1,1); digitalWrite(MT1_IN2,0);
                                 digitalWrite(MT2_IN1,0); digitalWrite(MT2_IN2,1);
                                 analogWrite(MT1_PWM,lpwm);
                                 analogWrite(MT2_PWM,rpwm);}
      else                      {  digitalWrite(MT1_IN1,1); digitalWrite(MT1_IN2,0);
                                 digitalWrite(MT2_IN1,0); digitalWrite(MT2_IN2,0);
                                 analogWrite(MT1_PWM,lpwm);
                                 analogWrite(MT2_PWM,rpwm);}
      }
   else if  ( CmdSpd[0] < 0){
      if     ( CmdSpd[1] > 0)  { digitalWrite(MT1_IN1,0); digitalWrite(MT1_IN2,1);
                                 digitalWrite(MT2_IN1,1); digitalWrite(MT2_IN2,0);
                                 analogWrite(MT1_PWM,lpwm);
                                 analogWrite(MT2_PWM,rpwm);}
      else if( CmdSpd[1] < 0){ digitalWrite(MT1_IN1,0); digitalWrite(MT1_IN2,1);
                                 digitalWrite(MT2_IN1,0); digitalWrite(MT2_IN2,1);
                                 analogWrite(MT1_PWM,lpwm);
                                 analogWrite(MT2_PWM,rpwm);}
      else                      {  digitalWrite(MT1_IN1,0); digitalWrite(MT1_IN2,1);
                                 digitalWrite(MT2_IN1,0); digitalWrite(MT2_IN2,0);
```

```
                                    analogWrite(MT1_PWM,lpwm);
                                    analogWrite(MT2_PWM,rpwm);}
      }
   else                    {
      if     ( CmdSpd[1] > 0)  { digitalWrite(MT1_IN1,0); digitalWrite(MT1_IN2,0);
                                    digitalWrite(MT2_IN1,1); digitalWrite(MT2_IN2,0);
                                    analogWrite(MT1_PWM,lpwm);
                                    analogWrite(MT2_PWM,rpwm);}
      else if( CmdSpd[1] < 0){  digitalWrite(MT1_IN1,0); digitalWrite(MT1_IN2,0);
                                    digitalWrite(MT2_IN1,0); digitalWrite(MT2_IN2,1);
                                    analogWrite(MT1_PWM,lpwm);
                                    analogWrite(MT2_PWM,rpwm);}
      else                          {  digitalWrite(MT1_IN1,0); digitalWrite(MT1_IN2,0);
                                    digitalWrite(MT2_IN1,0); digitalWrite(MT2_IN2,0);
                                    analogWrite(MT1_PWM,lpwm);
                                    analogWrite(MT2_PWM,rpwm);}
   }
}
void setup() { nh.initNode(); nh.subscribe(sub); nh.advertise(Pub);
pinMode(MT1_IN1,OUTPUT); pinMode(MT1_IN2,OUTPUT);
pinMode(MT1_PWM,OUTPUT);  pinMode(MT2_IN1,OUTPUT);
pinMode(MT2_IN2,OUTPUT); pinMode(MT2_PWM,OUTPUT);  }
void loop() {  msg.linear.x=lpwm; msg.linear.y=rpwm; nh.spinOnce();
Pub.publish(&msg); motorGo(); delay(50); }
```

โปรแกรมการทำงานของเซนเซอร์ไลดาร์ (Lidar)

```
        <launch>
        <node name="ydlidar_node" pkg="ydlidar" type="ydlidar_node"
output="screen">
        <param name="port" type="string" value="/dev/ydlidar"/>
        <param name="baudrate" type="int" value="128000"/>
```

```
        <param name="frame_id" type="string" value="laser_frame"/>
        <param name="angle_fixed" type="bool" value="true"/>
        <param name="intensities" type="bool" value="false"/>
        <param name="low_exposure" type="bool" value="false"/>
        <param name="heartbeat" type="bool" value="false"/>
        <param name="resolution_fixed" type="bool" value="true"/>
        <param name="angle_min" type="double" value="-180" />
        <param name="angle_max" type="double" value="180" />
        <param name="range_min" type="double" value="0.05" />
        <param name="range_max" type="double" value="10.0" />
        <param name="ignore_array" type="string" value="" />
        <param name="samp_rate" type="int" value="10"/>
        <param name="frequency" type="double" value="5"/>
        </node> <node pkg="tf" type="static_transform_publisher"
name="base_link_to_laser" args="0.0 0.0 0.0 0.0 0.0 0.0 /base_link /laser_frame 40" />
</launch>
```

โปรแกรมการทำงานของ hector SLAM

```
<launch> <arg name="geotiff_map_file_path" default="$(find hector_geotiff)/maps"/>
        <param name="/use_sim_time" value="false"/>
        <node pkg="rviz" type="rviz" name="rviz" args="-d $(find
hector_slam_launch)/rviz_cfg/mapping_demo.rviz"/>
        <include file="$(find hector_mapping)/launch/mapping_default.launch"/>
        <include file="$(find hector_geotiff)/launch/geotiff_mapper.launch"> <arg
name="trajectory_source_frame_name" value="scanmatcher_frame"/> <arg
name="map_file_path" value="$(arg geotiff_map_file_path)"/>
        </include>
</launch>
```

โปรแกรมการทำงานของ rf2o_odometry

1 การทำงานในส่วนของ xml

```
<launch>
        <node pkg="rf2o_laser_odometry" type="rf2o_laser_odometry_node"
name="rf2o_laser_odometry" >
        <param name="laser_scan_topic" value="/scan"/>
        <param name="odom_topic" value="/odom" />
        <param name="publish_tf" value="true" />
        <param name="base_frame_id" value="/base_link"/>
        <param name="odom_frame_id" value="/odom" />
        <param name="init_pose_from_topic" value=""
        <param name="freq" value="12.0"/>
        <param name="verbose" value="true" />
        </node>
</launch>
```

2 การทำงานในส่วนของ C++

```
#include "rf2o_laser_odometry/CLaserOdometry2D.h"
#include <tf/transform_broadcaster.h>
#include <tf/transform_listener.h>
namespace rf2o {
class CLaserOdometry2DNode : CLaserOdometry2D
{
public:
    CLaserOdometry2DNode();
    ~CLaserOdometry2DNode() = default;
    void process(const ros::TimerEvent &);
    void publish();
    bool setLaserPoseFromTf();
public:
    bool publish_tf, new_scan_available;
```

```
    double freq;
    std::string  laser_scan_topic;
    std::string  odom_topic;
    std::string  base_frame_id;
    std::string  odom_frame_id;
    std::string  init_pose_from_topic;
    ros::NodeHandle              n;
    sensor_msgs::LaserScan    last_scan;
    bool                         GT_pose_initialized;
    tf::TransformListener        tf_listener;
    tf::TransformBroadcaster odom_broadcaster;
    nav_msgs::Odometry       initial_robot_pose;
    ros::Subscriber laser_sub, initPose_sub;
    ros::Publisher odom_pub;
    bool scan_available();
    void LaserCallBack(const sensor_msgs::LaserScan::ConstPtr& new_scan);
    void initPoseCallBack(const nav_msgs::Odometry::ConstPtr& new_initPose);
};
CLaserOdometry2DNode::CLaserOdometry2DNode() :
    CLaserOdometry2D()
{
    ROS_INFO("Initializing RF2O node...");
    ros::NodeHandle pn("~");
    pn.param<std::string>("laser_scan_topic",laser_scan_topic,"/scan");
    pn.param<std::string>("odom_topic", odom_topic, "/odom");
    pn.param<std::string>("base_frame_id", base_frame_id, "/base_link");
    pn.param<std::string>("odom_frame_id", odom_frame_id, "/odom");
    pn.param<bool>("publish_tf", publish_tf, true);
    pn.param<std::string>("init_pose_from_topic", init_pose_from_topic,
"/base_pose_ground_truth");
    pn.param<double>("freq",freq,10.0);
```

```
    pn.param<bool>("verbose", verbose, true);
    odom_pub = pn.advertise<nav_msgs::Odometry>(odom_topic, 5);
    laser_sub = n.subscribe<sensor_msgs::LaserScan>(laser_scan_topic,1,&
CLaserOdometry2DNode::LaserCallBack,this);
    if (init_pose_from_topic != "")
    {
        initPose_sub = n.subscribe<nav_msgs::Odometry>(init_pose_from_topic,1,&
CLaserOdometry2DNode::initPoseCallBack,this);
        GT_pose_initialized = false;
    }
else
{
        GT_pose_initialized = true;
        initial_robot_pose.pose.pose.position.x = 0;
        initial_robot_pose.pose.pose.position.y = 0;
        initial_robot_pose.pose.pose.position.z = 0;
        initial_robot_pose.pose.pose.orientation.w = 0;
        initial_robot_pose.pose.pose.orientation.x = 0;
        initial_robot_pose.pose.pose.orientation.y = 0;
        initial_robot_pose.pose.pose.orientation.z = 0;
    }
    setLaserPoseFromTf();
    module_initialized = false;
    first_laser_scan = true;
    ROS_INFO_STREAM("Listening laser scan from topic: " << laser_sub.getTopic());
}
bool CLaserOdometry2DNode::setLaserPoseFromTf()
{
    bool retrieved = false;
    tf::StampedTransform transform;
    transform.setIdentity();
```

```
    try
    {
        tf_listener.waitForTransform("/base_footprint","/laser_frame", ros::Time(),
ros::Duration(5.0));
        retrieved = true;
    }
    catch (tf::TransformException &ex)
    {
        ROS_ERROR("%s",ex.what());
        ros::Duration(1.0).sleep();
        retrieved = false;
    }
    const tf::Matrix3x3 &basis = transform.getBasis();
    Eigen::Matrix3d R;
for(int r = 0; r < 3; r++)
    for(int c = 0; c < 3; c++)
        R(r,c) = basis[r][c];
    Pose3d laser_tf(R);
    const tf::Vector3 &t = transform.getOrigin();
    laser_tf.translation()(0) = t[0];
    laser_tf.translation()(1) = t[1];
    laser_tf.translation()(2) = t[2];
    setLaserPose(laser_tf);
    return retrieved;
}
bool CLaserOdometry2DNode::scan_available()
{
    return new_scan_available;
}
void CLaserOdometry2DNode::process(const ros::TimerEvent&)
{
```

```
    if( is_initialized() && scan_available() )
    {
        odometryCalculation(last_scan);
        publish();
        new_scan_available = false; }
    else
    {
        ROS_WARN("Waiting for laser_scans....") ;
    }
}
void CLaserOdometry2DNode::LaserCallBack(const sensor_msgs::LaserScan::ConstPtr&
new_scan)
{
    if (GT_pose_initialized){
        last_scan = *new_scan;
        current_scan_time = last_scan.header.stamp;
        if (!first_laser_scan)
        {
            for (unsigned int i = 0; i<width; i++)
                range_wf(i) = new_scan->ranges[i];
            new_scan_available = true;
        }
        else
        {
            init(last_scan, initial_robot_pose.pose.pose);
            first_laser_scan = false;
        }
    }
}
void CLaserOdometry2DNode::initPoseCallBack(const nav_msgs::Odometry::ConstPtr&
new_initPose)
```

```
{
    if (!GT_pose_initialized)
    {
        initial_robot_pose = *new_initPose;
        GT_pose_initialized = true;
    }
}
void CLaserOdometry2DNode::publish()
{
    if (publish_tf)
    {
        geometry_msgs::TransformStamped odom_trans;
        odom_trans.header.stamp = ros::Time::now();
        odom_trans.header.frame_id = odom_frame_id;
        odom_trans.child_frame_id = base_frame_id;
        odom_trans.transform.translation.x = robot_pose_.translation()(0);
        odom_trans.transform.translation.y = robot_pose_.translation()(1);
        odom_trans.transform.translation.z = 0.0;
        odom_trans.transform.rotation =  tf::createQuaternionMsgFromYaw
(rf2o::getYaw(robot_pose_.rotation()));
        odom_broadcaster.sendTransform(odom_trans);
    }
    nav_msgs::Odometry odom;
    odom.header.stamp = ros::Time::now();
    odom.header.frame_id = odom_frame_id;
    odom.pose.pose.position.x = robot_pose_.translation()(0);
    odom.pose.pose.position.y = robot_pose_.translation()(1);
    odom.pose.pose.position.z = 0.0;
    odom.pose.pose.orientation = tf::createQuaternionMsgFromYaw
(rf2o::getYaw(robot_pose_.rotation()));
    odom.child_frame_id = base_frame_id;
```

```
    odom.twist.twist.linear.x = lin_speed; //linear speed
    odom.twist.twist.linear.y = 0.0;
    odom.twist.twist.angular.z = ang_speed; //angular speed
    odom_pub.publish(odom);
}
}
int main(int argc, char** argv)
{
    ros::init(argc, argv, "RF2O_LaserOdom");

    rf2o::CLaserOdometry2DNode myLaserOdomNode;
    ros::TimerOptions timer_opt;
    timer_opt.oneshot = false;
    timer_opt.autostart = true;
    timer_opt.callback_queue = ros::getGlobalCallbackQueue();
    timer_opt.tracked_object = ros::VoidConstPtr();
    timer_opt.callback = boost::bind(&rf2o::CLaserOdometry2DNode::process,
&myLaserOdomNode, _1);
    timer_opt.period = ros::Rate(myLaserOdomNode.freq).expectedCycleTime();
    ros::Timer rf2o_timer = ros::NodeHandle("~").createTimer(timer_opt);
    ros::spin();
    return EXIT_SUCCESS;
}
```

โปรแกรมการทำงานของการระบุตำแหน่งด้วยวิธีการมอนติคาร์โลแบบปรับตัว (AMCL)

```
<launch>
<node pkg="amcl" type="amcl" name="amcl" output="screen">
<param name="odom_model_type" value="diff"/>
<param name="odom_alpha5" value="0.1"/>
<param name="transform_tolerance" value="0.2" />
<param name="gui_publish_rate" value="10.0"/>
```

```
        <param name="laser_max_beams" value="30"/>
        <param name="min_particles" value="500"/>
        <param name="max_particles" value="5000"/>
        <param name="kld_err" value="0.05"/>
        <param name="kld_z" value="0.99"/>
        <param name="odom_alpha1" value="0.2"/>
        <param name="odom_alpha2" value="0.2"/>
        <param name="odom_alpha3" value="0.8"/>
        <param name="odom_alpha4" value="0.2"/>
        <param name="laser_z_hit" value="0.5"/>
        <param name="laser_z_short" value="0.05"/>
        <param name="laser_z_max" value="0.05"/>
        <param name="laser_z_rand" value="0.5"/>
        <param name="laser_sigma_hit" value="0.2"/>
        <param name="laser_lambda_short" value="0.1"/>
        <param name="laser_lambda_short" value="0.1"/>
        <param name="laser_model_type" value="likelihood_field"/>
        <param name="laser_likelihood_max_dist" value="2.0"/>
        <param name="update_min_d" value="0.2"/>
        <param name="update_min_a" value="0.5"/>
        <param name="odom_frame_id" value="odom"/>
        <param name="resample_interval" value="1"/>
        <param name="transform_tolerance" value="0.1"/>
        <param name="recovery_alpha_slow" value="0.0"/>
        <param name="recovery_alpha_fast" value="0.0"/>
        </node>
</launch>
```

โปรแกรมการทำงานของการคำนวณแผนที่ (Costmap)

```
map_type: costmap
robot_base_frame: /base_link
```

transform_tolerance: 0.5

obstacle_range: 2.5

max_obstacle_height: 1.0

raytrace_range: 3.0

footprint: [[-0.22,-0.32] , [-0.22,0.32], [0.62,0.32] ,[0.62,-0.32]]

inscribed_radius: 0.2

circumscribed_radius: 0.3

inflation_radius: 0.2

map_server lethal_cost_threshold: 100

observation_sources: sensor sensor: {data_type: LaserScan, sensor_frame:
/laser_frame, topic: /scan, marking: true, clearing: true}

1 global Costmap

global_costmap:

costmap global_frame: /map

update_frequency: 2.0

publish_frequency: 0.5

costmap static_map: true

2 local Costmap

local_costmap:

costmap global_frame: /odom

update_frequency: 8.0

publish_frequency: 4.0

static_map: false

rolling_window: true

width: 6.0

height: 6.0

resolution: 0.10

โปรแกรมการทำงานของผู้วางแผน (Planner)

```
TrajectoryPlannerROS:
        acc_lim_theta: 0.2
        acc_lim_x: 0.1
        acc_lim_y: 0.1
        max_vel_x: 0.30
        min_vel_x: 0.15
        max_vel_theta: 1.5
        min_vel_theta: -1.5
        min_in_place_rotational_vel: 0.4
        Metre meter_scoring: true
        holonomic_robot: false
```

โปรแกรมการทำงานของการเคลื่อนย้ายฐาน (Move Base)

```
#include <move_base/move_base.h>
#include <tf2_ros/transform_listener.h>
int main(int argc, char** argv){
        ros::init(argc, argv, "move_base_node");
        tf2_ros::Buffer buffer(ros::Duration(10));
        tf2_ros::TransformListener tf(buffer);
        move_base::MoveBase move_base( buffer );
        ros::spin();
        return(0);
}
```

ภาคผนวก ค

การใช้งานระบบปฏิบัติการหุ่นยนต์สำหรับรถขนส่งสินค้าอัตโนมัติในโรงงานอุตสาหกรรม

การสร้างแผนที่

ขั้นตอนที่ 1 เปิดการทำงานของ hector slam โดยใช้คำสั่ง

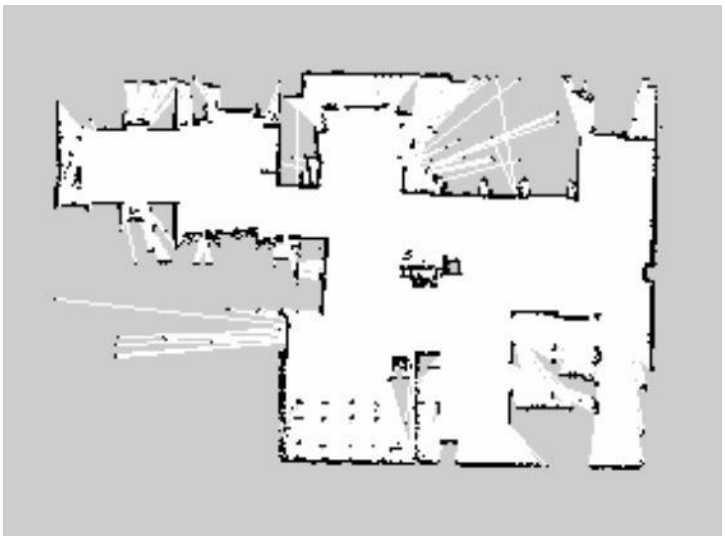
$ roslaunch hector_slam tutorial.launch

ขั้นตอนที่ 2 สั่งให้หุ่นยนต์เดินไปตามจุดหมายทั้งหมดที่ต้องการให้รถเคลื่อนที่

$ rosrun teleop_twist_keyboard teleop_twist_keyboard.py

ขั้นตอนที่ 3 ทำการบันทึกภาพแผนที่โดยใช้คำสั่ง

$ rosrun map_server map_server -f my_map



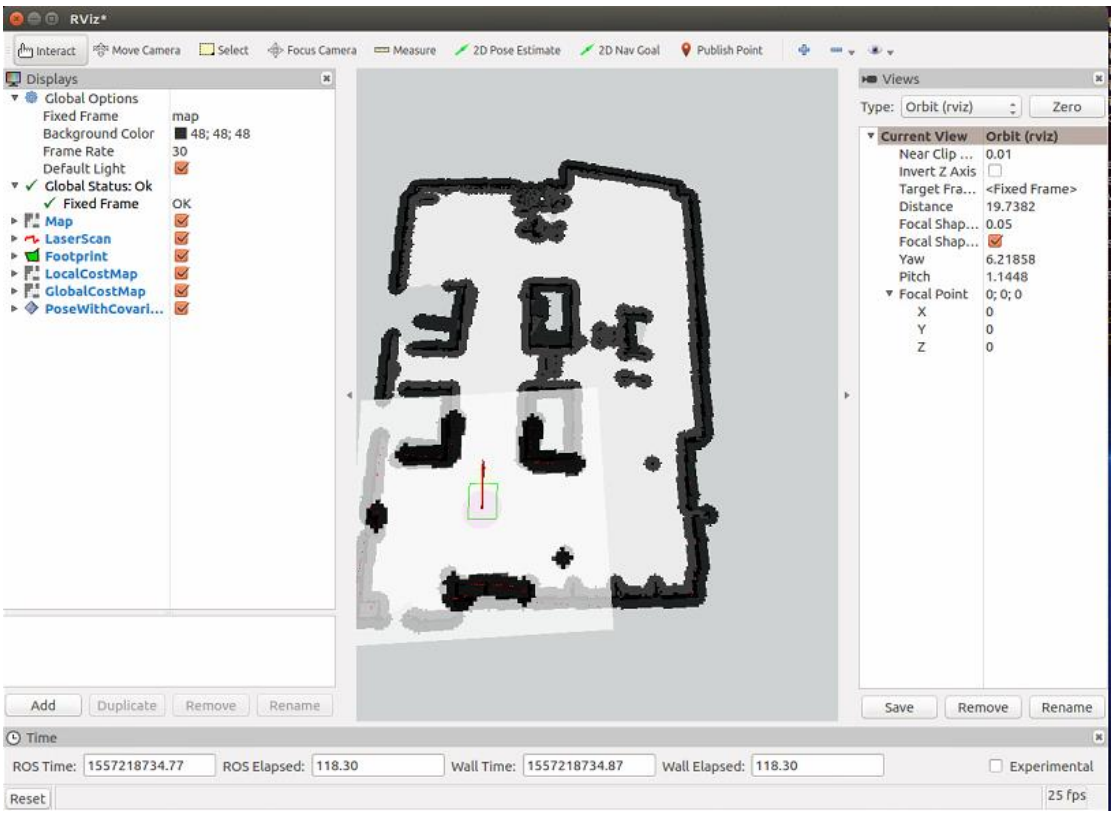
ภาพที่ ก.1 ตัวอย่างแผนที่ที่ถูกสร้าง

การใช้งานรถขนส่งสินค้าอัตโนมัติ

ขั้นตอนที่ 1 เรียกการทำงานโดยใช้คำสั่ง

$ roslaunch move_base.launch

จะปรากฏหน้าต่างของโปรแกรมอาร์วิชขึ้นมาดังภาพ



ภาพที่ ข.2 ตัวอย่างหน้าต่างอาร์วิช

ขั้นตอนที่ 2 กดปุ่มชื่อว่า 2D Nav Goal เมื่อกดที่ปุ่มแล้วลากใน Grid โดยหันทิศทางไปในทิศปลายทางที่ต้องการ จะปรากฏภาพลูกศรของปลายทางที่ต้องการให้หุ่นยนต์เคลื่อนที่ขึ้นมาในโปรแกรมอาร์วิช

ภาคผนวก ง

รายละเอียดข้อมูลโปรแกรมของระบบปฏิบัติการหุ่นยนต์สำหรับรถขนส่งสินค้าอัตโนมัติในโรงงานอุตสาหกรรม

รายละเอียดของระบบปฏิบัติการหุ่นยนต์สำหรับรถขนส่งสินค้าอัตโนมัติ ได้แบ่งข้อมูลออกเป็น 2 โฟลเดอร์ (folder)) ซึ่งข้อมูลได้อยู่ในแผ่นซีดี (CD) ที่แนบไว้ท้ายเล่ม โดยมีรายละเอียดดังนี้

1. ไฟล์ข้อมูลโปรแกรมที่สร้างระบบปฏิบัติการหุ่นยนต์สำหรับรถขนส่งสินค้าอัตโนมัติ ซึ่งเก็บไว้ใน CD:\agv_ws และ CD:\move_base_config