

ISTANBUL TECHNICAL UNIVERSITY
ELECTRICAL & ELECTRONICS FACULTY
CONTROL & AUTOMATION ENGINEERING DEPARTMENT



GRADUATION PROJECT:
Mobile Robot Applications with ROS

SEMESTER REPORT

EMRE AY

040100675

Project Advisor: Prof. Dr. HAKAN TEMELTAŞ

Fall 2014

Contents

1. Introduction

- a. Aim of the Project**
- b. Technological Overview**
 - i. ROS**
 - ii. Low Level Processing Layer (TI C2000)**
- c. Structure of the AGV**

2. Done Applications

- a. Obtaining the Kinematic Model of AGV**
- b. C2000 Programming and Basic Movement Test**
- c. ROS Installation**
- d. Necessary Packages/Stacks**
- e. Agv Package Creation**
- f. Sharp Application**
- g. URDF of AGV**
- h. Teleoperation of AGV in Rviz**
- i. Obtaining Image from Kinect**

3. Project Work Plan for Next Semester

References

All codes written in this report are available at:

<https://github.com/emreay-/>

1. Introduction

This is the semester report of the graduation project called "*Mobile Robot Applications with ROS*" which is due to the end of the Spring 2015 semester.

a. Aim of the Project

In this project it is aimed to develop applications with ROS (Robot Operating System) for the mobile robot of the type AGV (Autonomous Ground Vehicle) in ITU Robotics Laboratory. These applications include autonomous and/or semantic navigation with sub-applications such as image processing, serial communication, kinematic modeling, and simulations and so on. Developed applications can be a base for this robot to be used as a guide robot or library robot.

b. Technological Overview

i. ROS (Robot Operating System)

As the project name indicates, Robot Operating System (ROS) forms a vital basis for the project. With a rough definition, ROS is an open-source middleware created to be used in robotics applications. It has started as a project in 2007 by Willow Garage and its early versions are released in 2009. It is not a real operating system; its name is merely an analogy for its functionalities on robots.

Basically, it creates an environment for robot programming with its tools, libraries and features and hence it reduces work load for developers by eliminating the need to write codes for the same or similar tasks on different environments again and again, i.e. reinventing the wheel.

Its main infrastructure consists of publishing of/subscribing to messages, message recording and playback, having a server/client procedure and a distributed parameter server that can be dynamically reconfigured (Willow Garage, Inc.). These features played a vital role in its popularity and wide spread.

Besides these core infrastructures, definitions of common message types (IMU data, laser sensor data, joint state data, point clouds etc.), node communication graph and its many libraries are also its powerful features.

A ROS application basically includes several program executables called *nodes* that have certain relationships and communications with each other (i.e. publisher/subscriber). There are software containers called packages and stacks which store the executable source codes, custom message declarations, launch files and so on. With these features, it is easy to create packages for different applications.

In this project, ROS will be used in high level processing layer.

ii. Low Level Processing Layer (TI C2000)

While high level processing layer (onboard computer with ROS) deals with navigation, decision making and so on; low level processing layer is the part for motor driving and signal routing and it is connected to the high level layer.

In this project, Texas Instruments C2000 will be used for low level processing layer.

c. Structure of the AGV

AGV's in the ITU Robotics Laboratory are bi-wheeled differential drive vehicles. It drives with two brushless DC motors. Current main structure can be visualized as;

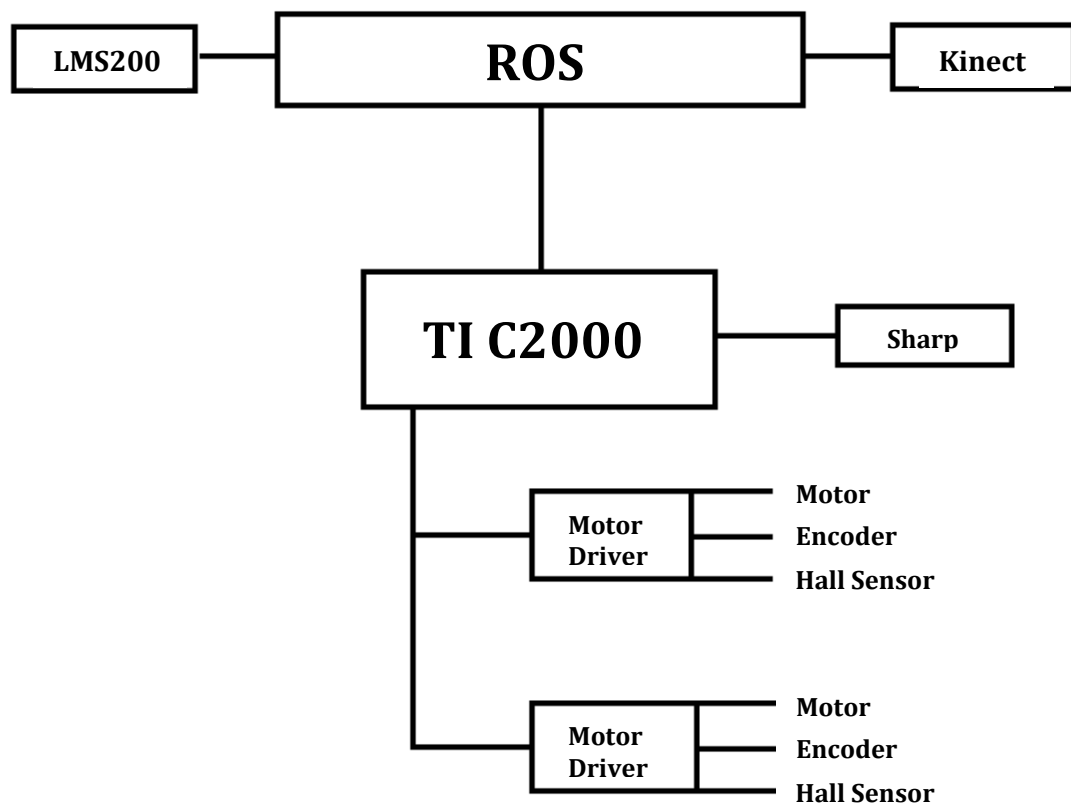


Figure 1 – Structure of AGV

It is anticipated to add an Xsense IMU to the robot.

2. Done Applications

a. Obtaining the Kinematic Model of AGV

As mentioned above, AGV is a differential drive vehicle. Suppose $r, R, L, \omega(t), v_L(t), v_R(t), \omega_L(t), \omega_R(t), \theta(t), v_{x_1}(t), v_{y_1}(t), v_{x_0}(t)$ and $v_{y_0}(t)$ as radius of the wheels, instantaneous curvature radius of the robot trajectory, length between wheels, linear velocity of left wheel, linear velocity of the right wheel, angular velocity of the left wheel, angular velocity of the right wheel, orientation angle of the robot, linear velocity of the robot in the x-axis of the robot frame, linear velocity of the robot in the y-axis of the robot frame, linear velocity of the robot in the x-axis of the world frame and linear velocity of the robot in the y-axis of the world frame respectively.

At any instant t ;

$$\begin{aligned}\omega \left(R + \frac{L}{2} \right) &= V_R & \omega \left(R - \frac{L}{2} \right) &= V_L \\ V_L &= r \omega_L & V_R &= r \omega_R \\ \omega &= \frac{V_R - V_L}{L} & v &= \frac{v_R + v_L}{2}\end{aligned}$$

So the kinematic equation in world frames;

$$\begin{bmatrix} v_{x_0}(t) \\ v_{y_0}(t) \\ \dot{\theta}(t) \end{bmatrix} = \begin{bmatrix} \cos \theta & 0 \\ \sin \theta & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} v(t) \\ \omega(t) \end{bmatrix}$$

And in the robot frames;

$$\begin{bmatrix} v_{x_1}(t) \\ v_{y_1}(t) \\ \dot{\theta}(t) \end{bmatrix} = \begin{bmatrix} r/2 & r/2 \\ 0 & 0 \\ -r/L & r/L \end{bmatrix} \begin{bmatrix} \omega_L(t) \\ \omega_R(t) \end{bmatrix}$$

b. C2000 Programming and Basic Movement Test

C2000 series DSP's can be programmed with Code Composer Studio IDE. MATLAB supports this IDE and it is possible to program it in MATLAB/Simulink and download.

To do this, it is needed to install Code Composer Studio, necessary eZdsp driver, jtag driver and Flash API of TMS320C28335. After these installations, it is now possible to create programs in Simulink and download to the C2000 target.

A basic movement program that was previously written by ITU Robotics Laboratory staff is downloaded to the Flash and tested. The program simply sends the necessary states for the two motor drivers over CAN to achieve a rectangular trajectory for the robot.

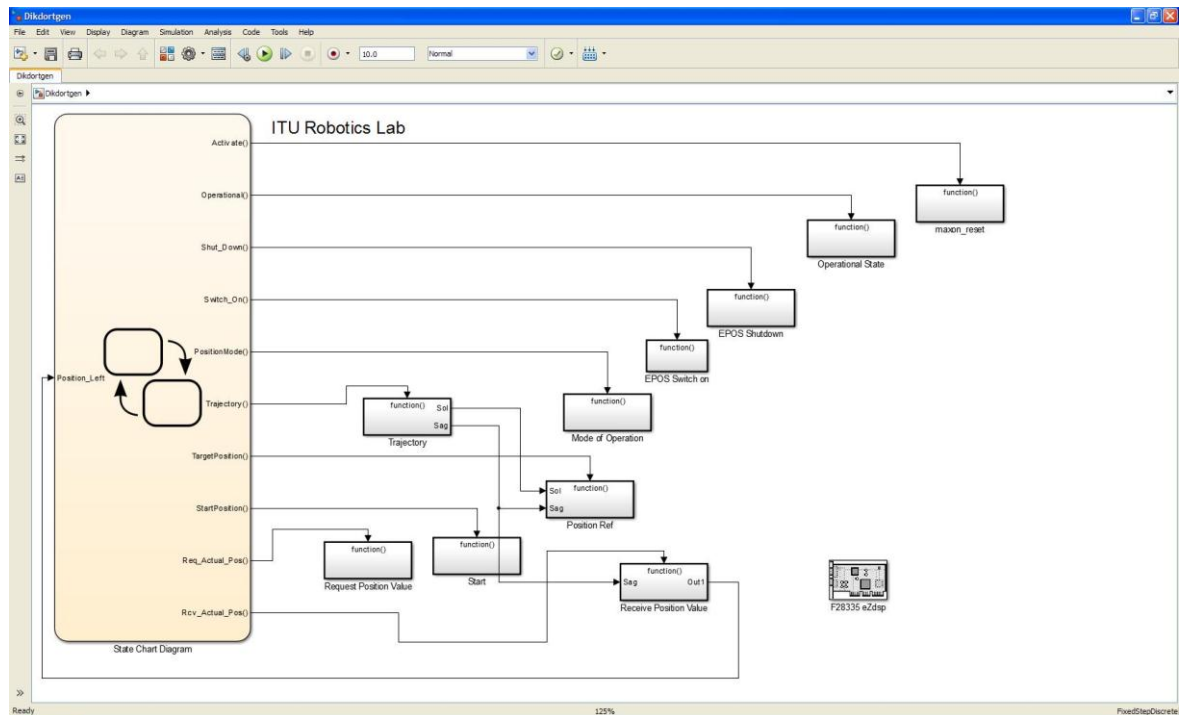


Figure 2 - Simulink Model for Movement Test

By running the program, the robot successfully moved on a rectangle trajectory.

c. ROS Installation

In this project Groovy Galapagos distribution of ROS is going to be used under Ubuntu 12.04 LTS operating system. The instructions on the official web site are used for installation (ros.org, 2013).

To set computer to allow *packages.ros.org*, the following commands are ran;

```
sudo sh -c 'echo "deb http://packages.ros.org/ros/ubuntu precise
main" > /etc/apt/sources.list.d/ros-latest.list'
```

```
wget http://packages.ros.org/ros.key -O - | sudo apt-key add -
```

Command 1

To get a full installation including rqt, rviz, 2D/3D simulators;

```
sudo apt-get install ros-groovy-desktop-full
```

Command 2

Initializing rosdep, a package that makes easier to install the dependencies of packages;

```
sudo rosdep init  
rosdep update
```

Command 3

To provide automatic addition of ROS environment variables to bash sessions;

```
echo "source /opt/ros/groovy/setup.bash" >> ~/.bashrc  
source ~/.bashrc
```

Command 4

Installing rosinstall, package makes installing ROS packages easier;

```
sudo apt-get install python-roscpp
```

Command 5

It is a must to create a workspace; in this distro a *catkin workspace*, for ROS. Workspace is the special directory that would contain all user ROS packages, it is the development directory. To create a catkin workspace in the name *catkin_ws*;

```
mkdir -p ~/catkin_ws/src  
cd ~/catkin_ws/src  
catkin_init_workspace  
cd ~/catkin_ws/  
catkin_make
```

Command 6

These commands will create a directory called *catkin_ws* in the home folder that contains sub directories and a *CMakeLists.txt*.

To provide automatic addition of workspace environment variables to bash sessions;

```
echo "source ~/catkin_ws/devel/setup.bash" >> ~/.bashrc  
source ~/.bashrc
```

Command 7

d. Necessary Packages/Stacks

Installations of the certain packages that are needed in the applications are made. These include: *roscpp*, *roscpp_serial*, *roscpp_serial_arduino*, *ps3joy*, *openni_camera* and *openni_kinect*.

e. Agv Package Creation

It is needed to have a package that would contain all the necessary files (source codes, custom message files, urdf files, launch files...). Up to this point, all of the applications are developed under one created package, but it is anticipated to have a stack with organized packages at the end of this project.

ROS Groovy is Catkin based and it has a certain command to easily create a package;

```
catkin_create_pkg <package_name> [package_dependency1]  
[package_dependency2]
```

Command 8

Based on this structure, a catkin package named *agv* (with dependencies to other packages *std_msgs*, *rospy* and *roscpp*) is created by running this command under path *~/[catkin_workspace]/src*;

```
catkin_create_pkg agv std_msgs rospy roscpp
```

Command 9

Adding these dependencies is vital to have the basic functionalities as message creation. Packages may depend on different packages based on their purposes. This command creates a directory called *agv* that contains a *package.xml* file, a *CMakeLists.txt* file and two sub directories *src* and *include*. The *package.xml* file includes the name and contact info of the publisher, license information, runtime, build and build tool dependencies and hence should be edited. *CMakeLists.txt*, as it can be understood from its name, is for CMake build of the project. This file also should be edited when necessary (for example adding executables, messages...). *src* directory is where the source codes will be stored.

After the package is created, and when a source code is added or changed, the package should be built by running this command under *~/[catkin_workspace]/*;

```
catkin_make
```

Command 10

Or sometimes if necessary;

```
catkin_make --force-cmake
```

Command 11

After building the package, it should be visible to the ROS. This can be checked by running the command;

```
rospack find agv
```

Command 12

If the command returns the path to the package, it means ROS can see the package.

f. Sharp Application

A basic application is done using an analog Sharp distance sensor with an Arduino Mega. The analog values are read and converted to centimeters in Arduino and then serially sent to the ROS from USB using roserial package. A node is subscribed to the serial port data and it published a string message according to the distance value to another node that is subscribed. Also the distance is plotted with respect to time.

The following source codes are saved under `~/[catkin_workspace]/src/agv/src` and necessary changes are made in order to add executables in `~/[catkin_workspace]/src/agv/CMakeLists.txt` then the package is built again.

```
#include <ros.h>
#include <ros/time.h>
#include <sensor_msgs/Range.h>

ros::NodeHandle nh;
sensor_msgs::Range range_msg;
ros::Publisher pub_range("range_data", &range_msg);
const int analog_pin = 0;
unsigned long range_timer;

float getRange(int pin_num)
{
    int input;
    float Volt, dist;
    input = analogRead(A0); //Take ADC value
    Volt = input*0.00488; //Conver the ADC value to Volts
    if(Volt<2.8 && Volt>0.93) //Curve fitting for 4-13 cm
    {
        dist = 2.5909*Volt*Volt-14.6564*Volt+24.8075;
    }
    else if(Volt<=0.93 && Volt>0.36) //Curve fitting for 14-32 cm
    {
        dist = 48.3450*Volt*Volt-94.8984*Volt+60.8176;
    }
    else dist = 0;
    return dist;
}

char frameid[] = "/ir_ranger";

void setup()
{
    nh.initNode();
    nh.advertise(pub_range);
    range_msg.radiation_type = sensor_msgs::Range::INFRARED;
    range_msg.header.frame_id = frameid;
    range_msg.field_of_view = 0.01;
```

```

    range_msg.min_range = 0.03;
    range_msg.max_range = 0.4;
}
void loop()
{
    // publish the range value every 50 milliseconds
    // since it takes that long for the sensor to stabilize
    if ( (millis()-range_timer) > 50)
    {
        range_msg.range = getRange(analog_pin);
        range_msg.header.stamp = nh.now();
        pub_range.publish(&range_msg);
        range_timer = millis() + 50;
    }
    nh.spinOnce();
}

```

Code 1 – Arduino Code for Sharp Application

```

#include "ros/ros.h"
#include "sensor_msgs/Range.h"
#include "std_msgs/String.h"

float range = -1;
void rangeCallback(const sensor_msgs::Range::ConstPtr& range_data)
{
    range = range_data->range;
    ROS_INFO("I heard: [%f]", range);
}

int main(int argc, char **argv)
{
    ros::init(argc, argv, "sharp_listener");
    ros::NodeHandle n;
    ros::Subscriber sub = n.subscribe("range_data", 1000,
rangeCallback);
    ros::Publisher pub = n.advertise<std_msgs::String>("order",
1000);
    std_msgs::String go;
    std_msgs::String stop;
    std_msgs::String error;
    go.data = "Go!";
    stop.data = "Stop!";
    error.data = "ERROR..";

    while (ros::ok())
    {
        while(range >= 0)
        {
            if(range>4 && range<15)
            {

```

```

        pub.publish(stop);
    }
    else if(range>15.001)
    {
        pub.publish(go);
    }
    else
    {
        pub.publish(error);
    }
    range = -1;
}
ros::spinOnce();
}
return 0;
}

```

Code 2 - Code for Sharp Listener Node in ROS

```

#include "ros/ros.h"
#include "std_msgs/String.h"
void orderCallback(const std_msgs::String::ConstPtr& msg)
{
    ROS_INFO("%s", msg->data.c_str());
}

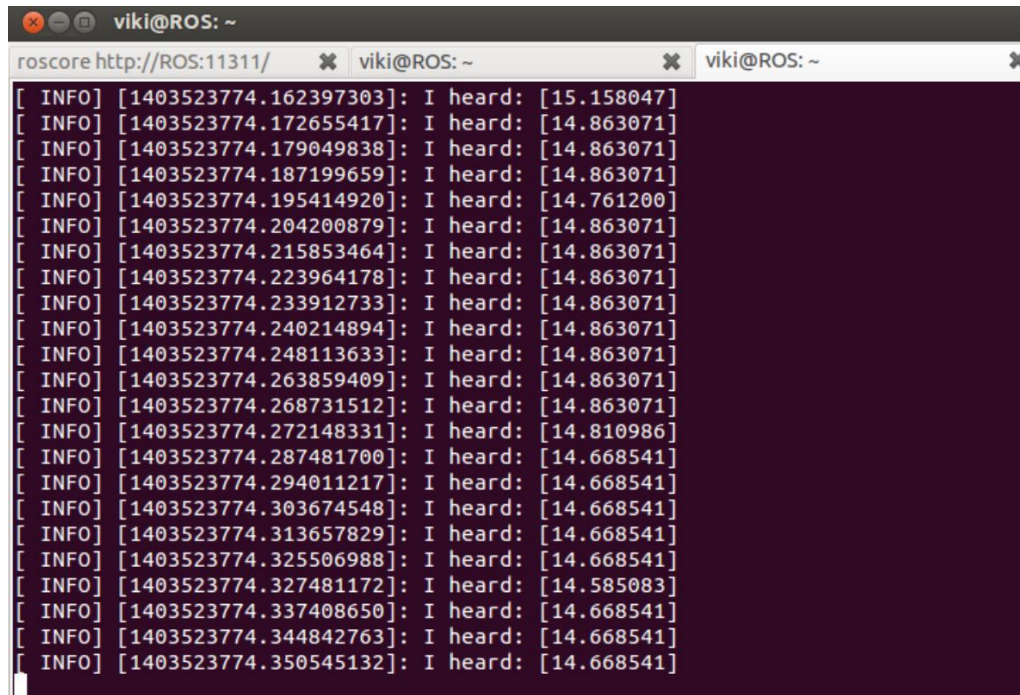
int main(int argc, char **argv)
{
    ros::init(argc, argv, "order_listener");
    ros::NodeHandle n;
    ros::Subscriber sub = n.subscribe("order", 1000,
orderCallback);
    ros::spin();
    return 0;
}

```

Code 3 - Code for Order Listener Node in ROS

Launching the nodes;

*Figure 3 - rqt Graph*

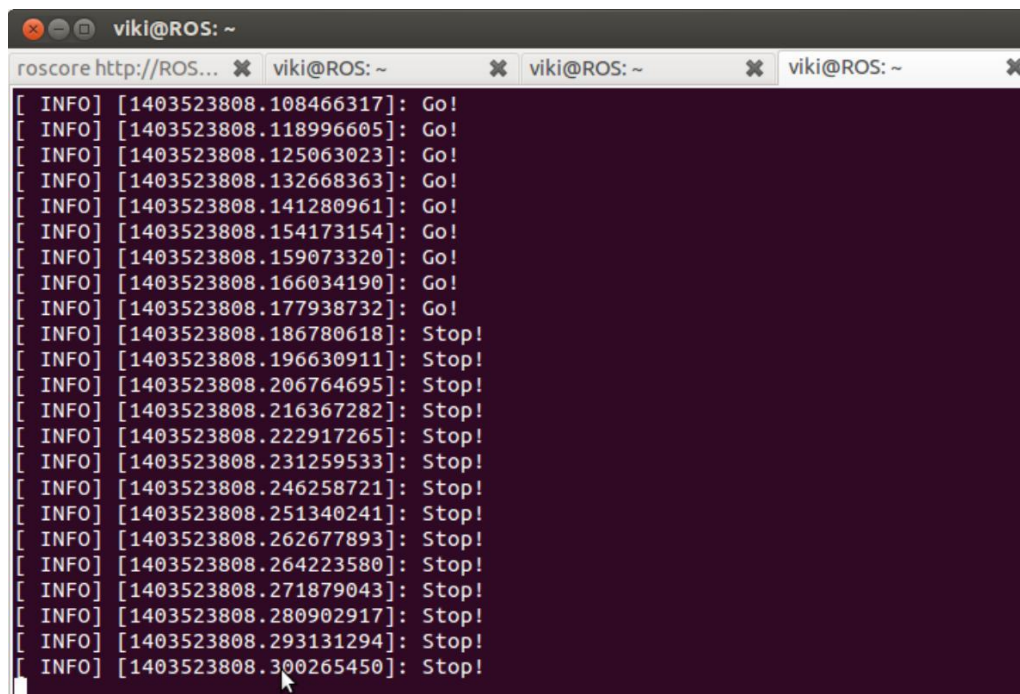


```

viki@ROS: ~
roscore http://ROS:11311/ viki@ROS: ~ viki@ROS: ~
[ INFO] [1403523774.162397303]: I heard: [15.158047]
[ INFO] [1403523774.172655417]: I heard: [14.863071]
[ INFO] [1403523774.179049838]: I heard: [14.863071]
[ INFO] [1403523774.187199659]: I heard: [14.863071]
[ INFO] [1403523774.195414920]: I heard: [14.761200]
[ INFO] [1403523774.204200879]: I heard: [14.863071]
[ INFO] [1403523774.215853464]: I heard: [14.863071]
[ INFO] [1403523774.223964178]: I heard: [14.863071]
[ INFO] [1403523774.233912733]: I heard: [14.863071]
[ INFO] [1403523774.240214894]: I heard: [14.863071]
[ INFO] [1403523774.248113633]: I heard: [14.863071]
[ INFO] [1403523774.263859409]: I heard: [14.863071]
[ INFO] [1403523774.268731512]: I heard: [14.863071]
[ INFO] [1403523774.272148331]: I heard: [14.810986]
[ INFO] [1403523774.287481700]: I heard: [14.668541]
[ INFO] [1403523774.294011217]: I heard: [14.668541]
[ INFO] [1403523774.303674548]: I heard: [14.668541]
[ INFO] [1403523774.313657829]: I heard: [14.668541]
[ INFO] [1403523774.325506988]: I heard: [14.668541]
[ INFO] [1403523774.327481172]: I heard: [14.585083]
[ INFO] [1403523774.337408650]: I heard: [14.668541]
[ INFO] [1403523774.344842763]: I heard: [14.668541]
[ INFO] [1403523774.350545132]: I heard: [14.668541]

```

Figure 4 - Output of Sharp Listener Node



```

viki@ROS: ~
roscore http://ROS... viki@ROS: ~ viki@ROS: ~ viki@ROS: ~
[ INFO] [1403523808.108466317]: Go!
[ INFO] [1403523808.118996605]: Go!
[ INFO] [1403523808.125063023]: Go!
[ INFO] [1403523808.132668363]: Go!
[ INFO] [1403523808.141280961]: Go!
[ INFO] [1403523808.154173154]: Go!
[ INFO] [1403523808.159073320]: Go!
[ INFO] [1403523808.166034190]: Go!
[ INFO] [1403523808.177938732]: Go!
[ INFO] [1403523808.186780618]: Stop!
[ INFO] [1403523808.196630911]: Stop!
[ INFO] [1403523808.206764695]: Stop!
[ INFO] [1403523808.216367282]: Stop!
[ INFO] [1403523808.222917265]: Stop!
[ INFO] [1403523808.231259533]: Stop!
[ INFO] [1403523808.246258721]: Stop!
[ INFO] [1403523808.251340241]: Stop!
[ INFO] [1403523808.262677893]: Stop!
[ INFO] [1403523808.264223580]: Stop!
[ INFO] [1403523808.271879043]: Stop!
[ INFO] [1403523808.280902917]: Stop!
[ INFO] [1403523808.293131294]: Stop!
[ INFO] [1403523808.300265450]: Stop!

```

Figure 5 - Output of Order Listener Node

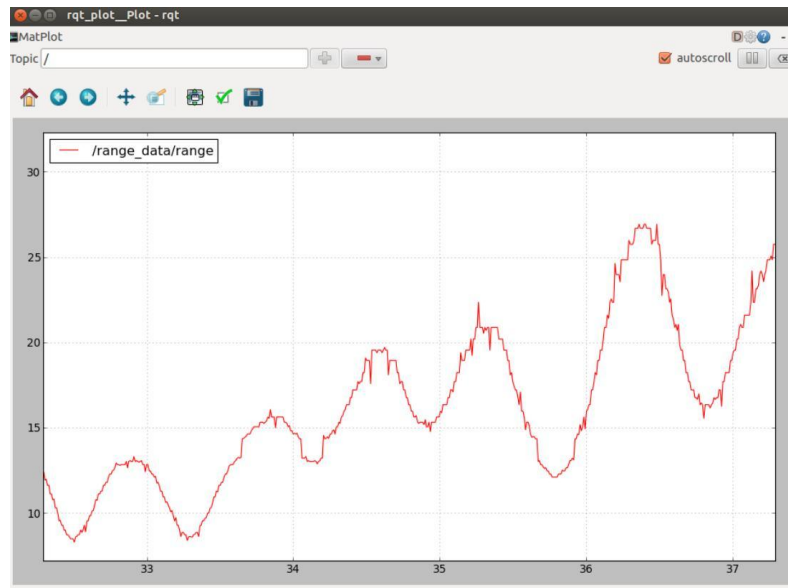


Figure 6 - rqt Plot of Distance vs. Time

g. URDF of AGV

Universal Robot Description File, or shortly URDF is an xml based file format that is used for describing joint frames, links, and geometry. In order to have a simple simulation of the AGV in the rviz environment, a URDF is created.

```
<robot name="AGV">
  <link name="base_link">
    <visual>
      <geometry>
        <box size="0.82 .49 .2"/>
      </geometry>
      <origin rpy="0 0 0" xyz="0 0 0.01"/>
      <material name="gray">
        <color rgba="0 0 0 0.6"/>
      </material>
    </visual>
  </link>
  <link name="wheel_left">
    <visual>
      <geometry>
        <cylinder length="0.05" radius="0.1"/>
      </geometry>
      <origin rpy="1.57079633 0 0" xyz="0 -0.22 0.01"/>
      <material name="black">
        <color rgba="1 0 0 1"/>
      </material>
    </visual>
  </link>
</robot>
```

```

</link>
<link name="wheel_right">
  <visual>
    <geometry>
      <cylinder length="0.05" radius="0.1"/>
    </geometry>
    <origin rpy="1.57079633 0 0" xyz="0 0.22 0.01"/>
    <material name="black"/>
  </visual>
</link>
<joint name="base_to_wheel_left" type="continuous">
  <parent link="base_link"/>
  <child link="wheel_left"/>
  <origin xyz="0 -0.22 0.01"/>
  <axis xyz="1 0 0"/>
</joint>
<joint name="base_to_wheel_right" type="continuous">
  <parent link="base_link"/>
  <child link="wheel_right"/>
  <origin xyz="0 0.22 0.01"/>
  <axis xyz="1 0 0"/>
</joint>
</robot>

```

Code 4 – URDF Code of AGV

h. Teleoperation of AGV in Rviz

Since a urdf is created, teleoperation of AGV can be simulated on rviz. Teleoperation is a vital functionality since AGV should be taken to the desired areas. It is a heavy robot and cannot be carried.

In this application a Sony Play Station 3 joystick is used for the teleoperation of the virtual model of the AGV in rviz. There is a useful package called *ps3joy* developed by Willow Garage. By installing and using *ps3joy* package, the PS3 joystick is matched and connected via Bluetooth to the computer. With the launch file in the package, a publisher node of the joystick values is established in ROS.

For the teleoperation of the model, it is needed to have a joint state publisher, an odometry transform broadcaster together with a robot state publisher. Since it is desired to take left and right analog joystick values as left and right wheel angular velocities, joint state publisher node is subscribed to the joystick node and the analog joystick values are passed to the joint state velocity values with multiplying with scalars. After that, the joint states are published.

Odometry transform broadcaster is necessary to define the transformations between the frames continuously. It is subscribed to the joint state publisher node

and it takes the right and left wheel angular velocity values from there. After receiving angular velocities, it calculates the translation and orientation of the robot according to the world frame using kinematic equations and it publishes the transformations.

```
#include<string>
#include<ros/ros.h>
#include<sensor_msgs/JointState.h>
#include<sensor_msgs/Joy.h>
#include<tf/transform_broadcaster.h>
//definition of the class TeleopAgv
class TeleopAgv
{
public:
TeleopAgv(); //constructor
~TeleopAgv(); //destructor
private:
//callback function for joystick
void joyCallback(const sensor_msgs::Joy::ConstPtr& joy);
//node handle object
ros::NodeHandle nh_;

int linearL_, linearR_;
double l_scale_L, l_scale_R;
ros::Publisher js_pub_; //publisher object
ros::Subscriber joy_sub_; //subscriber object
}; //end of class definition

//class constructor definition
TeleopAgv::TeleopAgv():
    linearL_(1),
    linearR_(2)
{
    ROS_INFO("Calling the constructor of class
TeleopAgv");
    //getting necessary parameters from parameter server
    nh_.getParam("axis_linear_L", linearL_);
    nh_.getParam("axis_linear_R", linearR_);
    nh_.getParam("scale_linear_L", l_scale_L);
    nh_.getParam("scale_linear_R", l_scale_R);
    //advertising to the topic joint_state_pub
    js_pub_ =
nh_.advertise<sensor_msgs::JointState>("joint_state_pub", 1);
    //subscribing to the topic joy
    joy_sub_ =
```



```

nh_.subscribe<sensor_msgs::Joy>("joy",1000,
&TeleopAgv::joyCallback, this);
}

//class destructor definition
TeleopAgv::~TeleopAgv()
{
    ROS_INFO("Calling the destructor of class
TeleopAgv");
}
//joystick callback function definition
void TeleopAgv::joyCallback(const
sensor_msgs::Joy::ConstPtr& joy)
{
    //joint state object
    sensor_msgs::JointState joint_state;
    //set header time stamp
    joint_state.header.stamp = ros::Time::now();
    //resize the number of names that would be given to
    the joint states
    joint_state.name.resize(3);
    //resize the number of (angular) velocity values of
    the joint states
    joint_state.velocity.resize(3);
    //set the name of the first joint
    joint_state.name[0] ="base";
    //set the name of the second joint as left
    joint_state.name[1] ="left_wheel";
    //set the name of the third joint as right
    joint_state.name[2] ="right_wheel";
    //set the velocity of left wheel from the left
    analog joystick value
    joint_state.velocity[1] = l_scale_L*joy-
>axes[linearL_];
    //set the velocity of left wheel from the left
    analog joystick value
    joint_state.velocity[2] = l_scale_R*joy-
>axes[linearR_];
    //publish joint states
    js_pub_.publish(joint_state);
}

//main function
int main(int argc, char** argv)

```

```

{
    //initialize the node
    ros::init(argc, argv, "jsPublisher");
    //construct a TeleopAgv object
    TeleopAgv teleop_agv;
    //spin
    ros::spin();
} //end of main

```

Code 5 - Joint State Publisher Code

```

#include<string>
#include<ros/ros.h>
#include<sensor_msgs/JointState.h>
#include<sensor_msgs/Joy.h>
#include<tf/transform_broadcaster.h>
#include<boost/bind.hpp>
#include<boost/ref.hpp>
//physical dimensions
const float wheelRadius = 0.1;
const float length = 0.8;
//callback function for subscribed joint states
void jsCallback(ros::NodeHandle &node_handle, const
sensor_msgs::JointState::ConstPtr& jointState)
{
    double wL, wR;
    wL = jointState->velocity[1];
    wR = jointState->velocity[2];
    node_handle.setParam("wLeft",wL);
    node_handle.setParam("wRight",wR);
}
//broadcastTF function definition
void broadcastTF(tf::TransformBroadcaster *tf_broadcaster,
                std::string device_frame,
                double x, double y, double z,
                double theta)
{
    // broadcast Transform from vehicle to device
    geometry_msgs::TransformStamped odom_trans;
    odom_trans.header.stamp = ros::Time::now();
    odom_trans.header.frame_id = "odom";
    odom_trans.child_frame_id = device_frame;
    odom_trans.transform.translation.x = x;
    odom_trans.transform.translation.y = y;
    odom_trans.transform.translation.z = z;
    odom_trans.transform.rotation =
    tf::createQuaternionMsgFromYaw(theta);
}

```

```

        //broadcasting the transform
        tf_broadcaster->sendTransform(odom_trans);
    }
    //main function
    int main(int argc, char** argv)
    {
        double x = 0.0;
        double y = 0.0;
        double th = 0.0;
        double wLeft, wRight, vLeft, vRight, vx, wth, dt,
        delta_x, delta_y, delta_th;
        //node initialize
        ros::init(argc, argv, "odom_transform");
        //node handle object
        ros::NodeHandle nh_;
        //tf broadcaster object
        tf::TransformBroadcaster broadcaster;
        //subscriber object
        ros::Subscriber js_sub_;
        //subscribe to joint_state_pub topic
        js_sub_ =
        nh_.subscribe<sensor_msgs::JointState>("joint_state_pub",
        100, boost::bind(jsCallback,boost::ref(nh_), _1));
        //time variables
        ros::Time current_time, prev_time;
        prev_time = ros::Time::now();
        ros::Rate loop_rate(30); //set loop rate to 30 Hz
        while (ros::ok())
        {
            ros::spinOnce(); //check new messages
            current_time = ros::Time::now(); //set current time
            //get left wheel angular velocity value from
            parameter server
            nh_.getParam("wLeft",wLeft);
            //get right wheel angular velocity value from
            parameter server
            nh_.getParam("wRight",wRight);
            //linear velocity of left wheel
            vLeft = wheelRadius*wLeft;
            //linear velocity of righth wheel
            vRight = wheelRadius*wRight;
            //linear velocity in ROBOT frame
            vx = (vRight+vLeft)/2;
            //angular velocity of the ROBOT
            wth = (vRight-vLeft)/length;

```

```

//time differential
dt = (current_time - prev_time).toSec();
//change in x axis in world frame
delta_x = vx*cos(th)*dt;
//change in y axis in world frame
delta_y = vx*sin(th)*dt;
//change in the orientation
delta_th = wth*dt;
x += delta_x;
y += delta_y;
th += delta_th;
//broadcast transforms
broadcastTF(&broadcaster, "base_link", x, y, 0, th);
broadcastTF(&broadcaster, "wheel_left", x, y, 0, th);
broadcastTF(&broadcaster, "wheel_right", x, y, 0, th);
//set previous time
prev_time = current_time;
//rest until loop rate is done
loop_rate.sleep();
} //end of while
return 0;
} //end of main

```

Code 6 – Odometry Transform Braodcaster Code

```

<launch>
  <arg name="gui" default="False" />
  <param name="robot_description" textfile="$(find
agv)/urdf/agv.urdf" />
  <param name="use_gui" value="$(arg gui)"/>
  <!-- JOY NODE -->
  <node respawn="true" pkg="joy"
    type="joy_node" name="joy_node" >
    <param name="dev" type="string"
value="/dev/input/js0" />
    <param name="deadzone" value="0.12" />
  </node>
  <!-- ROBOT STATE PUBLISHER NODE -->
  <node name="robot_state_publisher"
pkg="robot_state_publisher" type="state_publisher" />
  <!-- RVIZ -->
  <node name="rviz" pkg="rviz" type="rviz" args="-d $(find
agv)/urdf.rviz" />
  <!-- PS3 JOYSTICK PARAMETERS -->
  <param name="axis_linear_L" value="1" type="int"/>
  <param name="axis_linear_R" value="3" type="int"/>
  <!-- SCALAR PARAMETERS -->

```

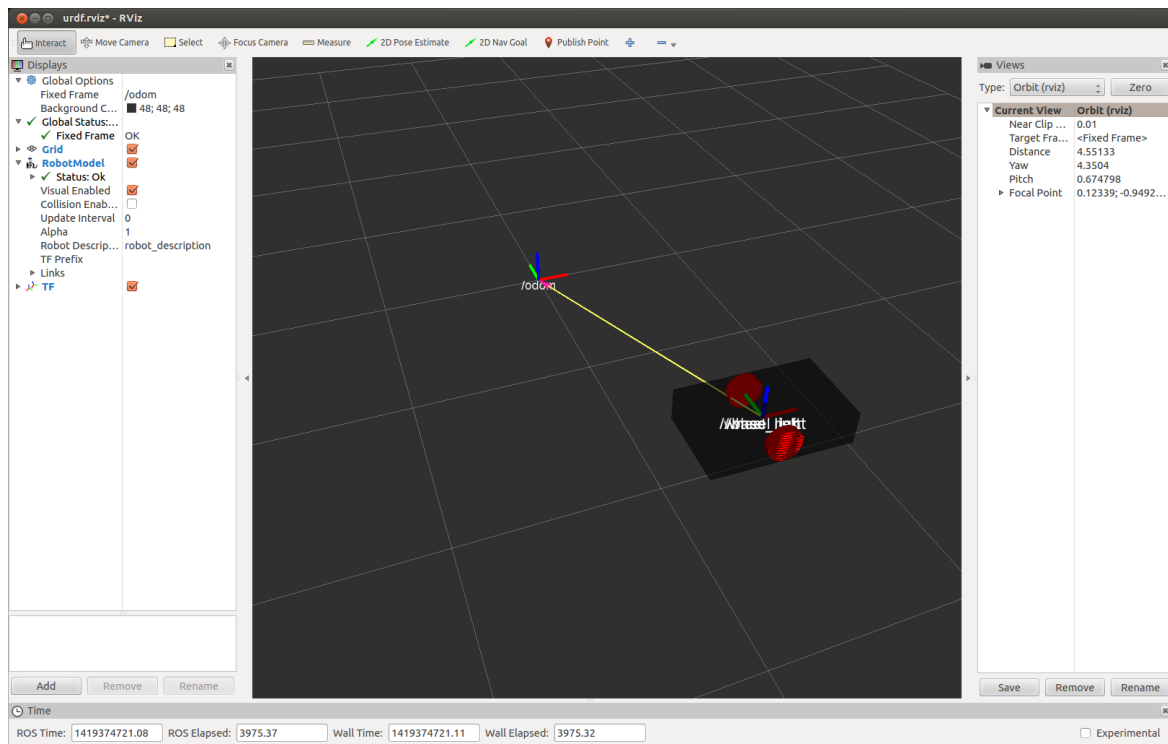
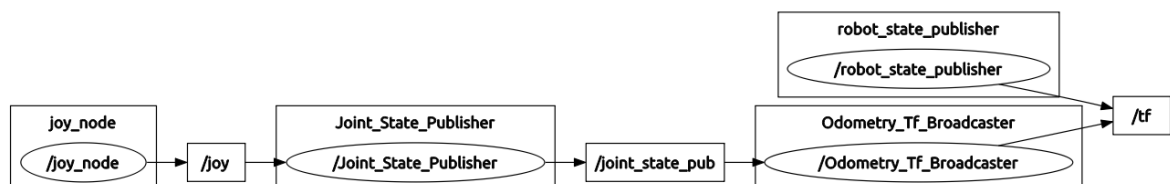
```

<param name="scale_linear_L" value="9" type="double"/>
<param name="scale_linear_R" value="9" type="double"/>
<!-- LEFT&RIGHT WHEEL ANG VELOCITY PARAMETERS -->
<param name="wLeft" value="0" type="double"/>
<param name="wRight" value="0" type="double"/>
<!-- JOINT STATE PUBLISHER NODE -->
<node pkg="agv" type="jsPublisher"
name="Joint_State_Publisher" ></node>
<!-- TF BROADCASTER NODE -->
<node pkg="agv" type="odomTF"
name="Odometry_Tf_Broadcaster" ></node>
</launch>

```

Code 7 - Launch File

Launching these nodes, it is possible to move the model robot in the rviz using the analog joysticks of the PS3 controller that is connected via Bluetooth.

**Figure 7 - Robot Teleoperation in rviz****Figure 8 - rqt Graph**

i. Obtaining Image from Kinect

As previously indicated, `openni_camera` and `openni_launch` packages are installed in order to identify Kinect to Ubuntu and publish the images in ROS. After connecting the Kinect to the computer via USB, the following commands will open a Kinect node that publishes images in ROS;

```
roslaunch oppenni_launch oppenni.launch
```

Command 13

To visualize the images (RGB, point clouds etc.), `rviz` can be used;

```
roslaunch oppenni_launch oppenni.launch
```

Command 14

After the `rviz` is opened, `camera_link` can be selected as fixed frame from the global options. Then, to see RGB image from Kinect, a Camera should be added from *Add→Camera* and the image topic can be selected as `/camera/rgb/image_color`. To see the point clouds, it has to be added from *Add→PointCloud2* and the topic can be selected as `/camera/depth_registered/points`.

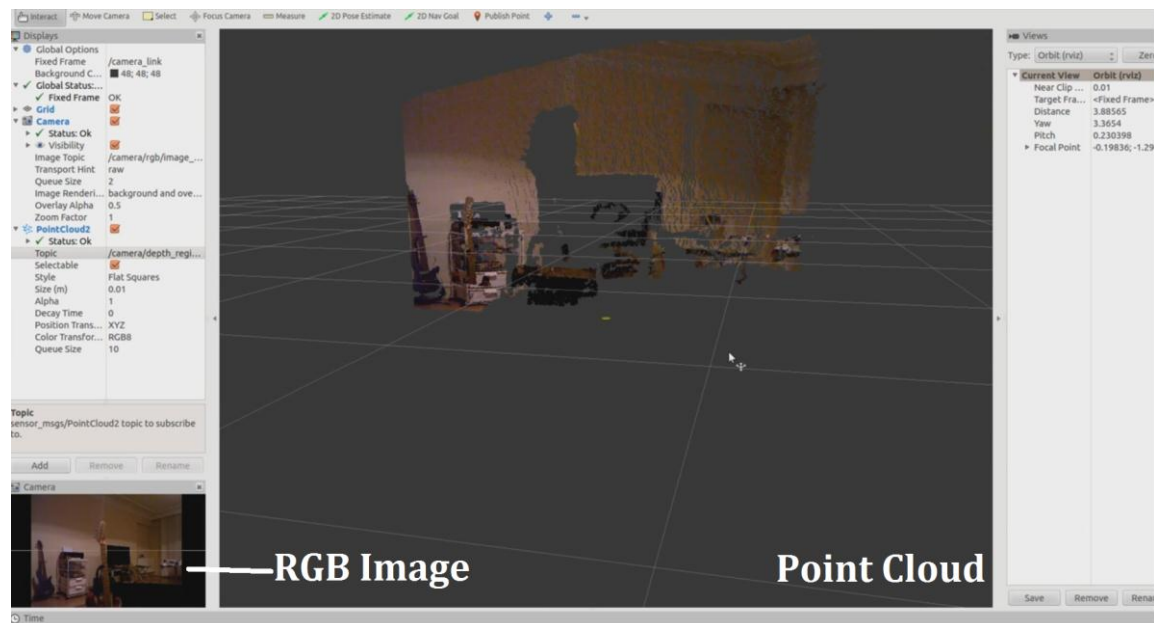


Figure 9 - Kinect Images in rviz

3. Project Work Plan for Next Semester

For the next semester the following works are planned;

- Providing the communication between ROS and C2000
- Programming C2000 suitable for the commands that will be send from ROS

- Laser sensor data reading
- Teleoperation of AGV
- Trajectory tracking
- Comparing the tracked trajectories of the real results vs. simulation
- Navigation applications development
- Simulation of the navigation
- Semantic navigation applications development

Together with these main topics;

- Testing and troubleshooting in each phase
- Writing short reports
- Creating a ROS stack for AGV's and similar mobile robots
- Finalizing project documentation
- Finalizing code version

References

ros.org. (2013, April 10). *Ubuntu install of ROS Groovy*. Retrieved December 15, 2014, from ROS Web Site: <http://wiki.ros.org/groovy/Installation/Ubuntu>

Willow Garage, Inc. (n.d.). *Core Components*. Retrieved December 23, 2014, from ROS Web Site: <http://www.ros.org/core-components/>