ROBOTICS ENGINEERING- II

TURTLEBOT NAVIGATION SYSTEM

Submitted By,
Umamaheswaran RAMAN KUMAR
Solene GUILLAUME
BSCV2 (2015-2016)
University of Bourgogne

TABLE OF CONTENTS

I. INTRODUCTION

Turtlebot Navigation System is a web based application developed to control the turtlebot that runs on open source Robot Operating System (ROS). The main idea of the project is to create a simple user friendly interface to guide the navigation of a turtlebot for manual control and also to create dynamic paths using simple mouse clicks for autonomous navigation in known arenas.

II. DEVELOPMENT TOOLS

Listed below are the frameworks and languages used for the development of the project.

1. ROBOT OPERATING SYSTEM (ROS)

The Robot Operating System provides a flexible framework and libraries to help develop simple and robust applications for robot. It is built on a collaborative software development environment where all the packages required to run a robot are readily available. It has simple message passing system using predefined messaging standards and many other services for communication between the nodes which makes it easy for the developers to create new nodes communicating with other nodes.

2. APACHE2 WEB SERVER

The apache server is the most widely used open source server software for hosting web applications in Linux platform. It is easy to setup and is preconfigured to run an application in no time after the setup is done. The commonly used protocols are Hyper Text Transfer Protocol (HTTP), Hyper Text Transfer Protocol for Secure Socket Layer (HTTPS) and File Transfer Protocol (FTP).

3. HYPER TEXT MARKUP LANGUAGE (HTML)

The Hyper Text Markup Language is the standard markup language used to create web applications. The HTML elements are the core structuring elements for web pages. Web browsers can read HTML files and render them as web pages based on HTML tag elements.

4. PHP

PHP is a server side scripting language which is written alongside with HTML code. It is mainly used for session management, user authentication, fetching information from database and other such functions which will be protected only if run from the server side. It can be also easily installed with apache server.

5. JAVASCRIPT

JavaScript is a client side scripting language which runs in the web browser. It is one of the three core technologies of World Wide Web content production along with HTML and CSS (Cascading Style Sheet). It is supported by all modern web browsers without any external plugins. It supports both object oriented and functional programming styles which makes it easier for any programmer.

III. ROSBRIDGE SUITE

Rosbridge provides JSON (JavaScript Object Notation) API to ROS functionality for supporting non-ROS applications to interact with ROS. 'Rosbridge_suite' is a meta-package containing rosbridge, various front end packages for rosbridge including Websocket package and other additional helper packages. The package is available as debian.

Command to install:

```
sudo apt-get install ros-<rosdistro>-rosbridge-server
```

Command to launch:

```
roslaunch rosbridge_server rosbridge_websocket.launch
```

There are two parts to the rosbridge: the protocol and the implementation.

1. ROSBRIDGE PROTOCOL

The rosbridge protocol is a specification for sending JSON based commands to ROS. The only required field is the 'op' field, which specifies the operation of that message. Each 'op' then specifies its own message semantics. The rosbridge protocol is a set of 'op' codes which define a number of operations, along with the semantics for each operation. There are various operation defined in the rosbridge protocol. They are mainly classified into four types based on the functionality.

Listed below are the classifications under the rosbridge protocol and their corresponding operations.

- ROS operations
  - *advertise* -> advertise that you are publishing a topic
  - *unadvertise* -> stop advertising that you are publishing a topic
  - *subscribe* -> request to subscribe to a topic
  - *unsubscribe* -> request to unsubscribe from a topic
  - *service_response* -> response to a service
- Rosbridge status messages
  - *set_status_level* -> request to set the reporting level of the status message
  - *status* -> rosbridge status message
- Message compression/transformation
  - *fragment* -> part of a fragmented message
  - *png* -> part of a png compressed fragmented message
- Authentication
  - *auth* -> set of authentication credentials to authenticate a client connection

The JSON message structure of the subscribe operation is show below with the optional fields highlighted.

```
{              "op"              : "subscribe",
  (optional) "id"              : <string>,
             "topic"           : <string>,
  (optional) "type"            : <string>,
  (optional) "throttle_rate"   : <int>,
  (optional) "queue_length"    : <int>,
  (optional) "fragment_size"   : <int>,
  (optional) "compression"     : <string>
}
```

2. ROSBRIDGE IMPLEMENTATION

The ‘rosbridge_suite’ package is a collection of packages that implement the rosbridge protocol and provides a Websocket transport layer. The packages include:

- rosbridge_library:

  The core rosbridge package. The rosbridge_library is responsible for taking the JSON string and sending the commands to ROS and vice versa.
- rosapi:

  It makes certain ROS actions accessible via service calls that are normally reserved for ROS client libraries. This includes getting and setting parameters, getting topics list, and more.
- rosbridge_server:

  Rosbridge_server provides a Websocket connection so browsers can “talk rosbridge”. Roslibjs is a JavaScript library for the browser that can talk to ROS via rosbridge_server.

IV. ROS JAVASCRIPT LIBRARIES

1. ROSLIBJS

‘roslibjs’ is the standard ROS JavaScript library for connecting to rosbridge from the web browser through WebSockets. It provides publishing, subscribing, service calls, actionlib and other essential ROS functionalities.

Listed below are the important classes that are used in the project from roslibjs.

- Ros
  Manages all connections to the server and interactions with ROS
- Topic
  Publishing or subscribing to a topic
- Message
  Message objects used for publishing and subscribing to and from topics

- ActionClient
  Action client for actionlib
- Goal
  An actionlib goal which is associated with an action server

The JavaScript code snippet to create ros node object to communicate with rosbridge.

```
var ros = new ROSLIB.Ros({
    url : 'ws://<turtlebot_ip_address>:9090'
});

ros.on('connection', function() {
    console.log('Connected to websocket server.');
});

ros.on('error', function(error) {
    console.log('Error connecting to websocket server: ', error);
});

ros.on('close', function() {
    console.log('Connection to websocket server closed.');
});
```

The JavaScript code snippet to create message object to be published.

```
var twist = new ROSLIB.Message({
    linear : {
        x : 0.5,
        y : 0,
        z : 0
    },
    angular : {
        x : 0,
        y : 0,
        z : 0
    }
});
```

The JavaScript code snippet to create topic where the message is published.

```
var cmdVel = new ROSLIB.Topic({
    ros : ros,
    name : '/cmd_vel',
    messageType : 'geometry_msgs/Twist'
});
```

The JavaScript code snippet to publish a message to a topic.

```
cmdVel.publish(twist);
```

2. NAV2D

‘nav2d’ is a JavaScript library used for loading maps on the web browser and handle the mouse clicks for 2D navigation. It provides options to specify the action server name and navigation action name. If ‘withOrientation’ is set to true, the user can also specify the orientation of the robot by clicking at the goal position and pointing into the desired direction (while holding the button pressed).

Listed below are the 3 classes in the library.

- ImageMapClientNav
  The ImageMapClientNav uses the ImageMapClient to create a map for use with the Navigator. The default topic used by ImageMapClientNav is ‘/map_metadata’.
- OccupancyGridClientNav
  The OccupancyGridClientNav uses the OccupancyGridClient to create a map for use with the Navigator. The default topic used by OccupancyGridClientNav is ‘/map’
- Navigator
  The Navigator is used to add navigation options to an object based on mouse click on the object.

The JavaScript code snippet shown below is to load a map in a HTML div element named ‘map’. The ‘OccupancyGridClientNav’ uses the ‘ros2d’ JavaScript library to get a handle to the viewer. The ‘Viewer’ class of ‘ros2d’ sets the location where the map has to be loaded and also adjusts the height and width of the viewer object. The server name is set to ‘move_base’ which is internally passed to the ‘Navigator’ and thereby all the mouse clicks on the map are converted into goals and sent to ‘move_base’ server through actionlib client.

```
var ros = new ROSLIB.Ros({
    url : 'ws://<turtlebot_ip_address>:9090'
});

console.log('Loading map.');

var viewer = new ROS2D.Viewer({
        divID : 'map',
        width : 350,
        height : 200
    });

var nav = NAV2D.OccupancyGridClientNav({
        ros : ros,
        rootObject : viewer.scene,
        viewer : viewer,
        continuous : true,
        serverName : 'move_base',
        withOrientation :true
});
```

V. ROS MESSAGES

ROS nodes communicate with each other by publishing messages to topics. Message is a simple data structure comprising of predefined fields. They can also have nested structures. The structure of the Message objects created in JavaScript follow the same structure as in ROS. The three mainly used messages in the project are '*geometry_msgs/Twist*', '*geometry_msgs/PoseWithCovarianceStamped*' and '*move_base_msgs/MoveBaseGoal*'.

1. TWIST

The 'geometry_msgs/Twist' message is used to send linear and angular velocities for the turtlebot to move. It is published to either '/cmd_vel' or '/cmd_vel_mux' topic. The structure of the message is given below.

```
geometry_msgs/Vector3 linear
  float64 x
  float64 y
  float64 z
geometry_msgs/Vector3 angular
  float64 x
  float64 y
  float64 z
```

2. POSEWITHCOVARIANCESTAMPED

The 'geometry_msgs/PoseWithCovarianceStamped' message is used to set the initial pose of the turtlebot in the map. If the initial pose of the turtlebot is not set then the robot will not be able to localize its position in the map and thereby affecting the navigation of the turtlebot when any goal is sent to move_base. The message is published to '/initializepose' topic. The structure of the message is given below.

```
std_msgs/Header header
  unit32 seq
  time stamp
  string frame_id
geometry_msgs/PoseWithCovariance pose
  geometry_msgs/Pose pose
    geometry_msgs/Point position
      float64 x
      float64 y
      float64 z
    geometry_msgs/Quaternion orientation
      float64 x
      float64 y
      float64 z
      float64 w
  float64[36] covariance
```

3. MOVEBASEGOAL

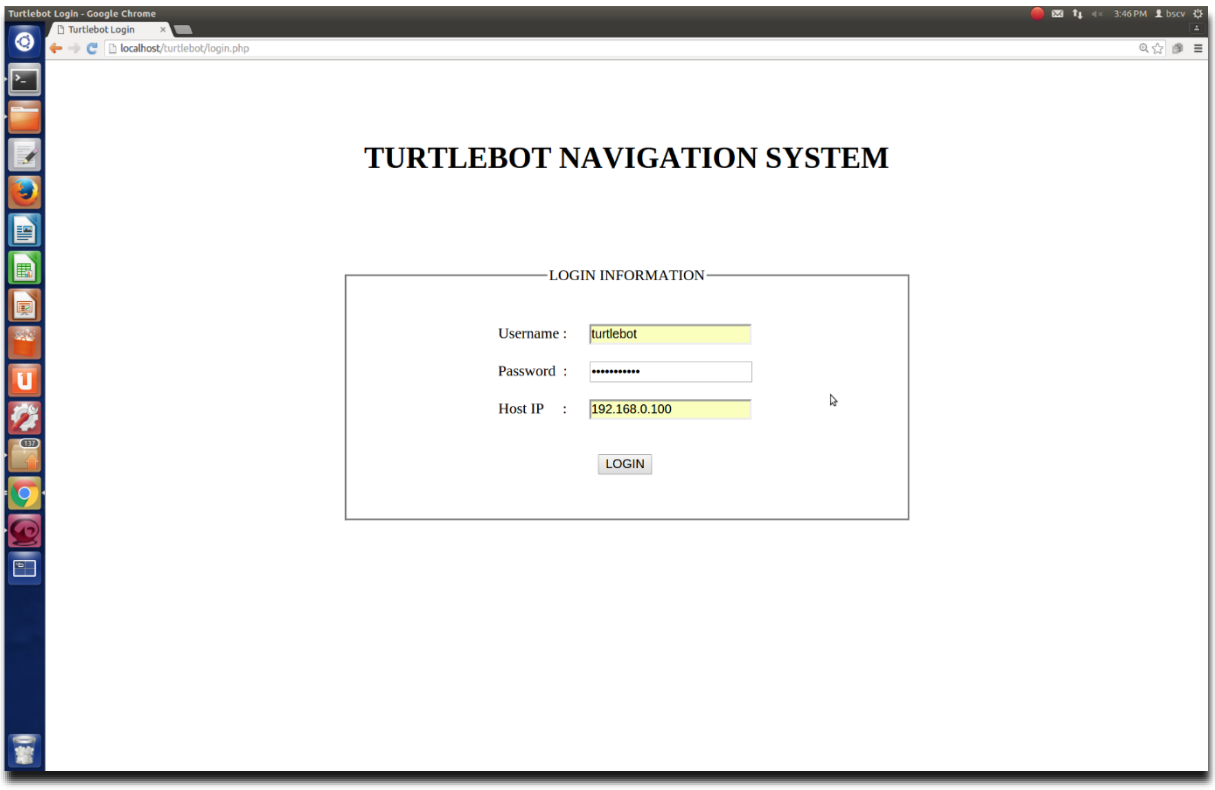
The ‘move_base_msgs/MoveBaseGoal’ message is used to set the goal for the turtlebot in the map. It is sent using the actionlib client ‘SimpleActionClient’. The structure of the message is given below.

```
geometry_msgs/PoseStamped target_pose
  std_msgs/Header header
    unit32 seq
    time stamp
    string frame_id
  geometry_msgs/Pose pose
    geometry_msgs/Point position
      float64 x
      float64 y
      float64 z
    geometry_msgs/Quaternion orientation
      float64 x
      float64 y
      float64 z
      float64 w
```

VI. FUNCTIONALITIES

Listed below are the screenshots of the web application and the functionalities explained.

1. LOGIN


Turtlebot Login - Google Chrome
Turtlebot Login
localhost/turtlebot/login.php
LOGIN

TURTLEBOT NAVIGATION SYSTEM

LOGIN INFORMATION

Username : turtlebot

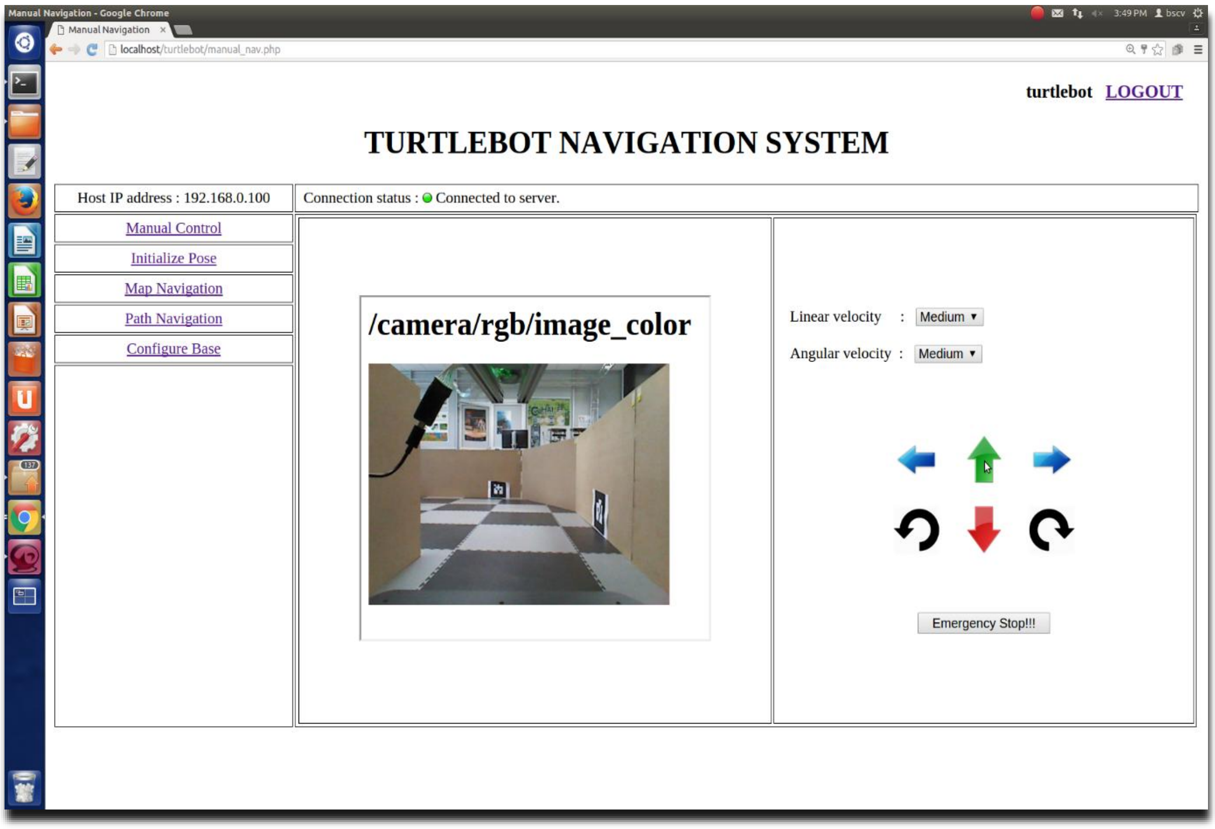
Password : •••••••••••

Host IP : 192.168.0.100

Fig1: Login screen

The user is allowed to access the application only after the username and password is authenticated by the server. If the authentication succeeds a session is created, the ip address is stored in a cookie variable and the user is redirected to the application's main page, else if the authentication fails the user is requested to enter valid credentials.

2. MANUAL NAVIGATION


TURTLEBOT NAVIGATION SYSTEM

Host IP address : 192.168.0.100

Connection status : ● Connected to server.

Linear velocity : Medium

Angular velocity : Medium

Fig2: Manual Navigation screen

Fig2 shows the initial screen displayed to the user after he/she login to the application. It display the ip address of the turtlebot along with the connection status of the connection with the turtlebot's Websocket server. A new connection to the Websocket server can be established by refreshing the page.

The user is allowed to control the turtlebot with mouse press on the arrows displayed on the screen. Every arrow corresponds to a different twist message being published to the 'cmd_vel_mux/input/navi' ROS topic based on the direction of the arrow. For example, if the user clicks on the forward arrow the twist message generated will have only the linear 'x' value set to a positive decimal number and all the remaining values are set to zero and is published to the 'cmd_vel_mux/input/navi' topic. The user can also change the linear and angular velocities by setting the velocity range to high, medium or low from the dropdown menu.

The middle frame shows the video streamed from the kinect camera connected to the turtlebot. It is also placed in all other navigation screens to help the user know the location of the turtlebot for better navigation.

3. INITIALIZE POSE

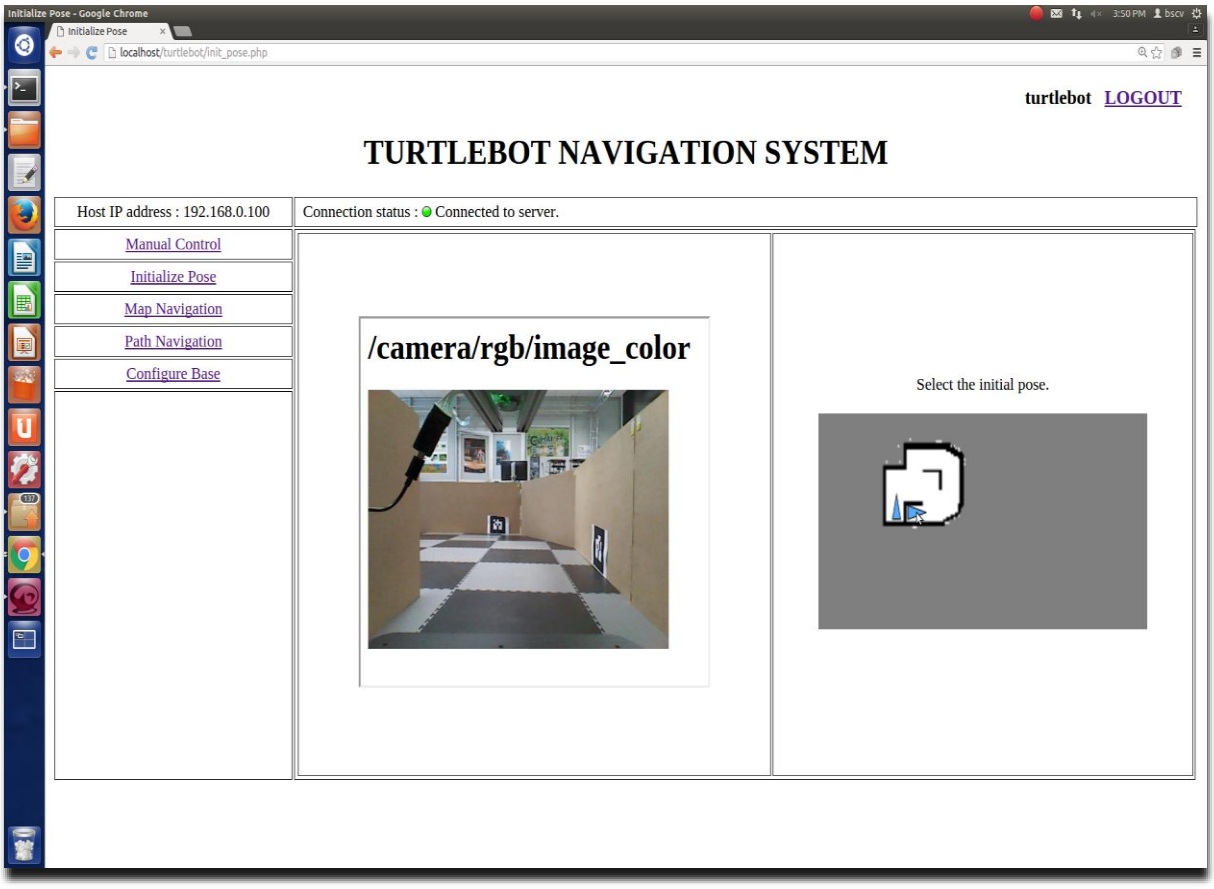

Initialize Pose - Google Chrome
Initialize Pose
localhost/turtlebot/init_pose.php
turtlebot LOGOUT
TURTLEBOT NAVIGATION SYSTEM
Host IP address : 192.168.0.100
Connection status : Connected to server.
Manual Control
Initialize Pose
Map Navigation
Path Navigation
Configure Base
/camera/rgb/image_color
Select the initial pose.

Fig3: Initialize Pose screen

Fig3 shows the screen where the user can set the change the pose of the turtlebot. It displays the current map being set in the map server along with the default position of the turtlebot set when the user is logged in. Initially there is only one blue arrow showing the position of the turtlebot along with the orientation. When the user does a mouse press event on the map to change the position of the turtlebot, a new arrow appears indicating the request for change in position and he/she can change the orientation by dragging and rotating the mouse when it is still in the pressed state.

The mouse release event triggers the publishing of a ‘PoseWithCovarianceStamped’ message, containing the corresponding position and orientation, to the ‘/initializepose’ topic. After the turtlebot is initialized to the new position selected the arrow indicating the previous position disappears. These actions can be repeated any number of times to set the correct position and orientation of the turtlebot because this helps the turtlebot to localize properly in the given map thereby giving better navigation results when doing a path planning.

If the map appears to be small the user can zoom in (Ctrl + ‘+’) the whole page to get a best fit of the screen with the application and therefore increasing the size of both video stream and the map. This increases the zoom for all the pages in the application because it changes the default zoom of the browser.

4. MAP NAVIGATION

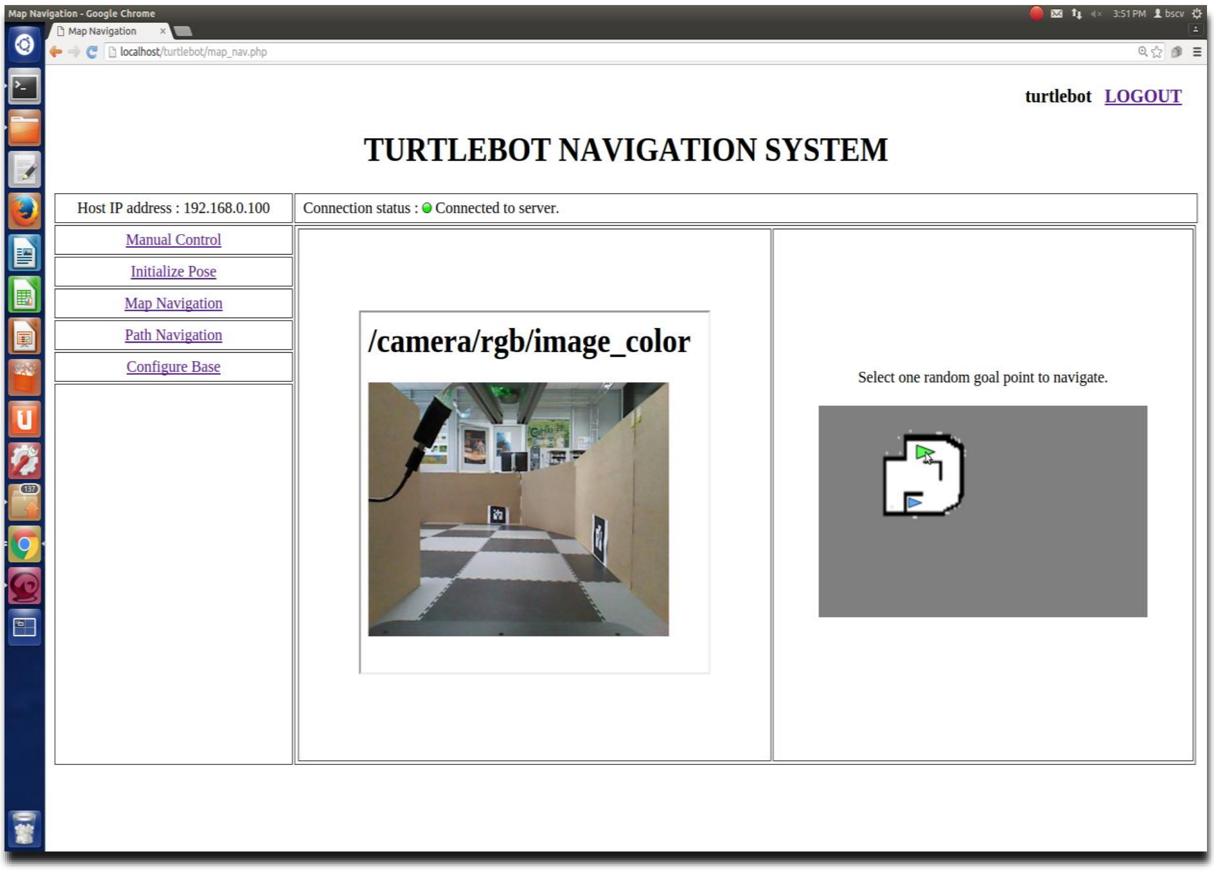

Map Navigation - Google Chrome
Map Navigation
localhost/turtlebot/map_nav.php
turtlebot LOGOUT
TURTLEBOT NAVIGATION SYSTEM
Host IP address : 192.168.0.100
Connection status : Connected to server.
Manual Control
Initialize Pose
Map Navigation
Path Navigation
Configure Base
/camera/rgb/image_color
Select one random goal point to navigate.

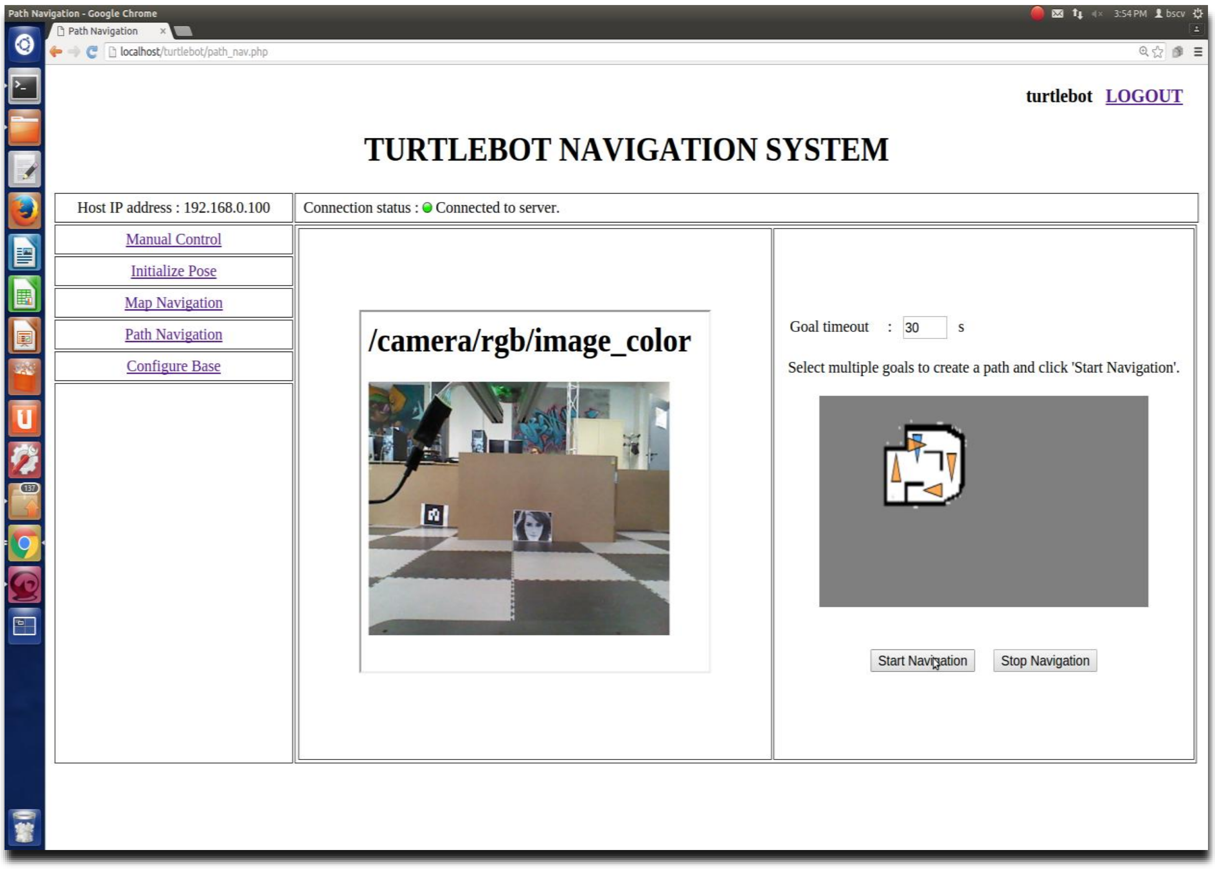
Fig4: Map Navigation screen

Fig4 shows the screen where the user can set a goal pose for the turtlebot to navigate using the ‘move_base’. It displays the current map being set in the map server along with the current position of the turtlebot shown by a blue arrow. When the user does a mouse press event on the map, a new green arrow appears indicating the request for setting a goal position and he/she can change the orientation by dragging and rotating the mouse when it is still in the pressed state.

The mouse release event triggers a ‘MoveBaseGoal’ message, containing the corresponding position and orientation, being sent to the actionlib client and the goal position is indicated using a saffron coloured arrow. The ‘move_base’ server starts planning its path to the new goal set based on the global and local cost map. If the goal set is in the gray region then the goal is ignored else the turtlebot starts navigating towards the goal. The navigation of the turtlebot is indicated by the continuous change in position and orientation of the blue arrow on the map.

During navigation the turtlebot tries to avoid obstacles that were not present earlier in the map because the ‘move_base’ local cost map calculation also depends on the input data from rplidar’s laser scan. Ones the turtlebot reaches the destination/goal the arrow indicating the goal disappears and the map is left with only the blue arrow indicating the current pose of the turtlebot.

5. PATH NAVIGATION


Path Navigation - Google Chrome

localhost/turtlebot/path_nav.php

turtlebot LOGOUT

TURTLEBOT NAVIGATION SYSTEM

Host IP address : 192.168.0.100

Connection status : ● Connected to server.

Goal timeout : 30 s

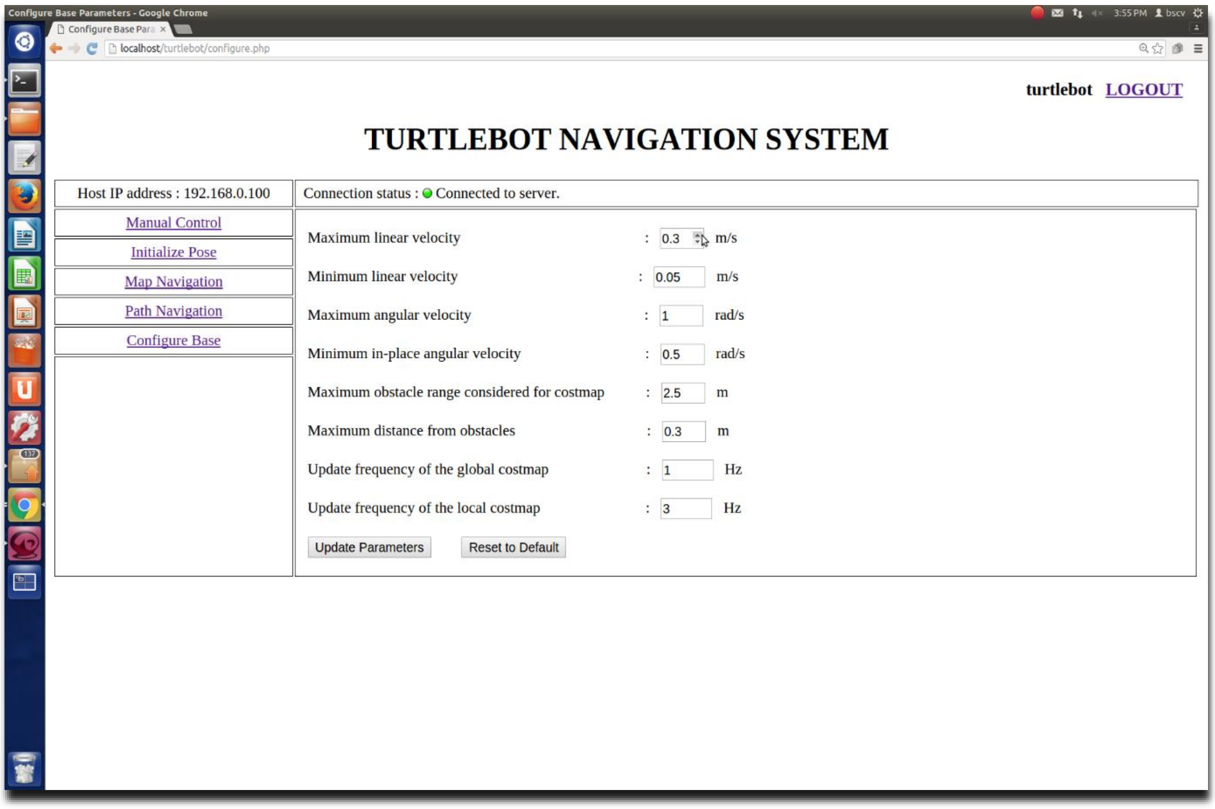
Select multiple goals to create a path and click 'Start Navigation'.

Fig5: Path Navigation screen

Fig5 shows the screen where the user can select multiple goals to create a path through which the turtlebot navigates autonomously. It displays the current map being set in the map server along with the current position of the turtlebot shown by a blue arrow and the different goals points of the path chosen by the user shown in saffron coloured arrows. When the user does a mouse press event on the map a new green arrow appears indicating the request for adding a goal position to the current path and he/she can change the orientation by dragging and rotating the mouse when it is still in the pressed state.

The mouse release event triggers a ‘MoveBaseGoal’ message, containing the corresponding position and orientation, being added to the current set of goals in the path and a new saffron coloured arrow is added to the existing set of arrows in the map. When the user clicks on the ‘Start Navigation’ button the first goal is sent to the ‘move_base’ server and the turtlebot starts moving towards the first goal. The navigation of the turtlebot is indicated by the continuous change in position and orientation of the blue arrow on the map. A timer is set which keeps sending new goals to the server from the path every 30 seconds. If the current goal is reached before the time then the turtlebot waits until it receives its next goal and if the previous goal is not reached then the goal is skipped and the turtlebot starts navigating to the new goal. The timer interval can be changed to take values between 15 seconds to 300 seconds. When the user clicks the ‘Stop Navigation’ button the turtlebot reaches the current goal position and stops navigating further.

6. CONFIGURE BASE PARAMETERS


Configure Base Parameters - Google Chrome

localhost/turtlebot/configure.php

turtlebot LOGOUT

TURTLEBOT NAVIGATION SYSTEM

Host IP address : 192.168.0.100

Connection status : ● Connected to server.

Maximum linear velocity	:	0.3	m/s
Minimum linear velocity	:	0.05	m/s
Maximum angular velocity	:	1	rad/s
Minimum in-place angular velocity	:	0.5	rad/s
Maximum obstacle range considered for costmap	:	2.5	m
Maximum distance from obstacles	:	0.3	m
Update frequency of the global costmap	:	1	Hz
Update frequency of the local costmap	:	3	Hz

Fig6: Configure Base Parameters screen

Fig6 shows the screen where the user can see few of the important parameters of the 'move_base' server that is currently available. All the parameter fields are editable and so if the user wishes to change the parameters he/she can modify the values of the field and click on 'Update Parameters' button. When this button is clicked the values in the field are validated for correctness. If the values are not within a particular range then the parameters are not updated and an error message is shown with the acceptable range of the parameter. The parameters are sent to parameter server and update happens only when the validation is successful.

The user can reset the parameters back to the default values by clicking the 'Reset to Default' button. The parameters updated are saved only for that particular session and if the user logout and login again then the parameters are set to the default values. The newly updated parameters are used by the 'move_base' server for further path planning.

VII <u>PROJECT LAUNCH FILE</u>

The project is dependent on multiple packages and its nodes. So every time the application is started all the launch files and nodes should be started from the turtlebot notebook. In order to simplify the task a single launch file is created including all the dependent launches and nodes. The launch file for the project is given below and the comments stating the purpose of including each node.

```
<launch>

    <!-- Launching turtlebot base and rplidar -->
    <include file="$(find turtlebot_le2i)
                     /launch/remap_rplidar_minimal.launch" />

    <!-- Launching the kinect camera -->
    <include file="$(find rbx1_bringup)/launch/openni_driver.launch" />

    <!-- Launching amcl for localization in the map-->
    <include file="$(find rbx1_nav)/launch/tb_demo_amcl.launch" >
        <!-- Set the value of the map -->
        <arg name="map" value="class_arena_map2.yaml" />
    </include>

    <!-- Launching the robot_pose_publisher node to get the transform -->
    <node name="robot_pose_publisher" pkg="robot_pose_publisher"
                                      type="robot_pose_publisher"/>

    <!-- Launching websocket for allowing access to non_ROS programs -->
    <include file="$(find rosbridge_server)
                     /launch/rosbridge_websocket.launch" />

    <!-- Launching web_video_server node for video stream through http-->
    <node name="web_video_server" pkg="web_video_server"
                                  type="web_video_server"/>

</launch>
```

VIII FUTURE ENHANCEMENTS

- Provide single user the access to connect to multiple turtlebots in the same session so the user can control the navigation of all the turtlebots simultaneously
- Allow the user to choose the map, he/she wants to work on, from the frontend based on the arena
- Creating new maps from the front end for new arenas and the maps created can be dynamically loaded to the application on the fly
- Enable zoom in, zoom out and rotation options for the map

IX IMPORTANT LINKS

Link to project source code repository:
https://github.com/umaatgithub/TurtlebotNavigationSystem-Web.git

Link to project demonstration video:
https://youtu.be/XH-CUbHvbfQ

X **REFERENCES**:

http://www.ros.org/

http://wiki.ros.org/rosbridge_suite

https://github.com/RobotWebTools/rosbridge_suite/blob/groovy-devel/ROSBRIDGE_PROTOCOL.md

http://wiki.ros.org/roslibjs

https://help.ubuntu.com/lts/serverguide/httpd.html

https://en.wikipedia.org/wiki/HTML

http://php.net/