**4.2 ROS Programming**

**4.2.1 ROS Installation and Init**

Pre-compiled repositories for ROS distributions are officially supported and supplied for Ubuntu. The HLPL would be ROS running on Ubuntu 12.04 LTS on a notebook computer. ROS Groovy distribution is supported on Ubuntu 12.04 LTS so its complete packages are installed according to the directives at ROS Wiki Website.

After installation a workspace is needed to be created. Workspace is the container folder where all the packages and their relative files are stored. Catkin build system has a command to easily create a workspace for ROS.

There are several packages needed to be installed that are not included in the ROS core packages. These are usually specific work or hardware packages. Since the sensors to be installed are decided, their relative software driver packages for ROS are installed including ps3joy for Play Station 3 joystick, sicktoolbox\_wrapper for laser sensors and xsens\_driver for IMU.

To contain the applications that are going to be written, a ros package is needed to be created. Catkin also provides an easy package creation with a command and its several arguments. Packages usually depend on other packages in order to use their functionalities. So a package named *agv* is created with dependencies to *roscpp* and *rospy* packages.

**4.2.2 Teleoperation**

Teleoperation is a vital functionality for manual data collection or to move the robot to the application areas. It is needed to be able to move ITU-AGVs manually as desired in order to collect laser data or images for use in algorithm development on Simultaneous Localization and Mapping (SLAM), loop closure or similar applications.

**4.2.2.1 Simulation**

Before directly passing to work on ITU-AGVs, the teleoperation is desired to be applied on a simulation environment of ROS. To achieve this, a visual model has to be created. Rviz visualization environment supports an xml based format for basic 3D robot representation called *Unified Robot Description Format* (\*.urdf). It is a simple parser with a simple syntax. So basically, a box with to wheels in the dimensions and placement of the ITU-AGVs can be created by a urdf file as follows;

|  |
| --- |
| <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>  <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> |

So basically the links and joints are geometrically defined with their positions, dimensions together with their relationship with each other. So their hierarchy tree can be understood by the system and it is possible to visually see the tree as in Figure ~\ref{fig:urdfTree}. When the created urdf file is opened in Rviz, a simple box and two wheels with the given parameters can be seen as in Figure ~\ref{fig:agv-urdf}. This visualization is a simple and quick solution. But it is possible to integrate more realistic 3D models. Rviz supports Collada file format (\*.dae) which is an open xml schema, so the 3D models created in various CAD software such as Google SketchUp can be converted to Collada file and implemented to Rviz~\cite{Martinez-Ros}. So a 3D model of ITU-AGVs are created using Google SketchUp and it is converted to the Collada file (Figure ~\ref{fig:agv-dae}).

It is wanted to realize the tele-operation using the Play Station 3 joystick. It is desired to control with both analog buttons and digital buttons. In the analog mode, the vertical values of left and right analog buttons will be the angular velocity references (in rpm) of left and right wheels. In digital mode, cross buttons will drive the motors on constant angular velocities. The up and down direction buttons will move the robot forwards and backwards, left and right buttons will drive the motors in the opposite directions resulting in a clockwise and a counter clockwise rotation around the central point. Lastly it is desired to only one mode at a time, hence while *R2* button on the joystick is pushed digital mode would be active and otherwise analog mode would be active.

Using the installed ps3joy package, communication with PS3 joystick over Bluetooth is handled and the button values are published to ROS environment. A node that subscribed to the joystick topic is created. This node gets the joystick button values, and passes the values of the necessary buttons as the relevant joint’s velocity with multiplying it with a scalar, then publishes all the joint states. Another node is written so that it is subscribed to the topic that joint states are published. When the joint states data is received this node passes the velocity data to the parameter server in the callback function. Then in the main loop, it calculates the odometry of the robot and publishes the odometry information.

After building the nodes and launching them the 3D model of ITU-AGVs have successfully moved with using the PS3 joystick (Figure ~\ref{fig:agv-teleop-sim})

**4.2.2.2 Teleop of ITU**

Since the teleoperation is successfully made on the simulation, it is convenient to realize the teleoperation of ITU-AGVs. A similar but modified approach is made. A node is created so that it would subscribe to the joystick topic and every time the joystick data is received it takes the needed button values. If the R2 button is pressed, it configures several variables depending on the values of digital cross buttons. If R2 button is not pressed, it configures the same variables depending on the analog button values. In the main loop, the node publishes an array of the variables which are configured in the callback function. This array is in the form that has been specified in ~\cite{subsec:comm with hlpl}. In order to send the commands to ITU-AGVs, another node is subscribed to the topic in which the array is published and it sends the array to LLPL of ITU-AGVs over a serial COM port. After the nodes are built, teleoperation of ITU-AGVs is successfully achieved.