1 Results

About the wireless plugin, there was an evaluation error, according to the comment seen here: http://answers.gazebosim.org/question/13021/plugin-for-wirelesstransmitter-and-wirelessreciever-sensors/. In this blog, the Wireless plugin happens to be within Gazebo7, however this doesn't mean we have to use Gazebo7.

The nice thing would be to have a *prebuilt* WirelessPlugin (a file with *.so* extension), ready to use in the Gazebo simulation. To do so:

- Find a robot in Kinetic with a Wirelessplugin, perform the installation via sudo-apt-install-kinetic-'name of the robot'
- Restore the old situation of the simulation so far, but with the new robot

Notes: Remember when you start a new clean catkin-ws, the file *bashrc* should be completed as follows. Add the following lines to the file:

- /opt/ros/ros-distro/setup.bash
- tildasymbol/catkin-ws/devel/setup.bash
- export ROS_PACKAGE_PATH=/home/yourname/catkin_ws/src:ROS_PACKAGE_PATH

Now I was trying to see if the AGV robot (wiki.ros.org/agvs) which is not an official installation of Hydro, can be compiled into the Kinetic release. The Gazebo is working, but the control not. The gmapping and amcl are working, so now try to make the Husky working too.

The big challenge is, and it would be nice to have, find a robot with Wifii to be installed directly into Kinetic and be simulated here.

2 To do's

- Install *gazebo ros packages* or similar and check it on a Kinetic robot (try with the Husky distro found for Kinetic) and install the corresponding packages —— search for something like "How to install gazebo-ros-pkgs for Kinetic"
- How to have a direct .so file for Wirelessplugins? Probably need to write one myself
- **Pose the question on Wiki.Ros.questions !!**
- Add an antenna model receiver onto the Husky URDF description, passages are on Section 4.1.
- Look into the Transmitter propagation model and understand the basic principle of it. The variable **ModelStrdDevs, represents the st.dev of the Gaussian propagation model**

3 Achieved

- Plugins are neither installed with the ROS distro (Kinetic, Indigo, ..) nor with the robot itself. They are under the folder /devel/lib and they come with the *gazebo ros packages*. It is possible that the .so file came from a catkin-make of some packages, since according to wiki.ros.org/catkin/Tutorials/using-a-workspace the catkin-make writes libraries in to the **/devel** folder. **Plugin shared objects found: They come with the gazebo-ros-pkg catkin-make installation, following the tutorial**. If we see inside how the gazebo-ros-pkgs are made we see .cpp and .h files, then by building them with catkin-make. Additional note is, according to the official Gazebo plugin tutorials the available plugins in the folder **gazebo-plugins** are found also under the path: /opt/ros/kinetic/include/gazebo-plugins, in form of header files .h

- On the Ubuntu 14.04 the Indigo system has been completely reinstalled
- The gazebo-ros-pkgs successfully build and the .so files are placed under the /devel folder. Since the Wirelessplugins was missing, I cloned the Gazebo master release which contains the wireless sensors .cpp and .h files, add them into the gazebo-plugin source folder, built them with catkin-make to check if the .so files libraries are added into the **simone-ws/devel/lib**, however **no Wireless.so file has been automatically created**. This implies some modifications into the .cpp and .h single plugin files maybe
- Husky packages have been restored in the Kinetic distro, the simulation is again at the last point in time left on the 20.Aug.
- The antenna receiver is now on the Husky robot, under the additional folder

4 Notes on Wifii Plugin implementation

4.1 Notes on Plugins tutorials

A small document with a review of the xacro and macro properties of the Husky is available in the Ubuntu PC of KFS; the receiver model has to be moved onto the husky robot, and the plugin should be wrapped into the robot as well. Maybe a physical model is actually not needed, but just insert the sensor tag i.e. sensor tag symbol, with the corresponding properties.

Adding a wireless receiver as an small cylinder antenna, modifying the Husky URDF. xacro files. As first according to the Clearpath robotics following the Customized Husky config, the **husky-customization** folder has been cloned. This contains two subfolders, the husky-custom-description and the husky-custom-gazebo, and since we want to add an URDF element onto the Husky, we directly use the first subfolder. The file **custom-description.urdf.xacro** has been modified with the Receiver link and joint of the Wireless Receiver (simple white cylinder anternna). In order to make it work, the *description.launch* file include file is added with the custom.urdf.xacro file.

Now the WirelessReceiver has to be fired up and added as a sensor tag into the custom.urdf.xacro file. At this time I am first following the **http://gazebosim.org/tutorials?tut=ros-plugins** tutorial. The aim is to make a shared object file (.so) to add into the custom.xacro file of the Husky.

4.2 The single Receiver and Transmitter .cpp and .h files

Taken the file $gazebo - ros - lasercpp$ it is visible that a node is created, which advertises (i.e. publishes) data on a pre-created message, in this case the sensor-msgs:LaserScan. It is very likely that the Transmitter plugin should do the same.

The class reference for the WirelessTransmitter and Receiver plugins are found under **http://osrf-distributions.s3.amazonaws.com/gazebo/api/2.2.1/classgazebo_1_1sensors_1_1WirelessTransmitter.**

4.2.1 WirelessTransmitter

Description of the Transmitter by function block

- The constant NObstacle has been placed to **0**, no ostacles present between the transmitter and the receiver
- On **Line 39** the WirelessTransmitter class is wrapped with a Transceiver class, what is the role of the Transceiver? The **GetTopic** function is defined in the Transceiver.cpp on line 46

- **void WirelessTransmitter::Load(const std::string &_worldName)** there is a reference to a worldModel. As in the Section 4.2.3 it checks for the transceiver element.

  On line 72 the publisher **pub** is used to publish the propagation model data, publishing data on the msgs::PropagationGrid, gets the topic with Gettopic() function define in the Transceiver model, on **line 127** it publishes the PropagationGrid msg object

- **void WirelessTransmitter::Init()** the testRay is used to check collision between obstacles in the **GetSignalStrenght** function.

- **bool WirelessTransmitter::UpdateImpl()** this is where the propagation model is defined and where the signal *strenght* is calculated. An object **msg** is defined with the type *PropagationGrid*, this will later be published. **Need to understand the propagation model**. On line **127** the **pub** publishes the propagation model via the **msg** tag, but wouldn't be the msg empty?

- The functions **GetESSID** and **GetFreq** are self-explaining

- **double WirelessTransmitter::GetSignalStrength(const math::Pose &_receiver,const double rxGain)**, this function takes as input the **receiver** via its reference & and the **rxGain i.e. the receiver gain**, constructs a 3-Dm vector from the sensor position to the end of the receiver. On **Line 178** a *distance* is calculated as the max (std::max) between 1 (?) and the referencePose.pos.Distance(_receiver.pos), I think the distance between the sensor and the receiver, then the **wavelenght** is calculated, then it returns **rxPower**

4.2.2 WirelessReceiver

Description of the Receiver by function blocks

- The function **void WirelessReceiver::Load(const std::string &_worldName)**, with **pub** publishes the propagation model and the corresponding node with message type **msgs::WirelessNodes**, gets the **transceiver element** like in Section 4.2.3

- **bool WirelessReceiver::UpdateImpl(bool)** is also present in Section 4.2.1 for the Transmitter, defines a **msg** object of the type **msgs::WirelessNodes**, defines **rxPower, this is the return of GetSignalStrenght function** and a **txFreq, transmitter frequency acquired in Line 113**, get the sensors in the SDF file via GetSensors(), line **108** checks for "wireless_transmitter", in Lines **114 and 115** take via the functions GetFreq and GetSignalStrenght the transmitter and **freq** and **power**, what does this do **msgs::WirelessNode *wirelessNode = msg.add_node()?**

4.2.3 WirelessTransceiver

Description of the Transceiver by functions blocks

- The **GetTopic** function is defined in the Transceiver.cpp on line 46 and is used in the Transmitter and Receiver models to get the topic name from the SDF files

- In the function of line 56 **void WirelessTransceiver::Load(const std::string &_worldName)** the $reference - pose$ takes in the sensor reference pose (think defined under the sensor SDF tag 'pose') while the *pose* the sensor pose (Have no clue which is the difference). Then the $parent - entity$ helps preventing the dynamic cast. Formula : ref pose = pose + parent entity. On line 67 checks for a missing Transceiver element, this is present in the ⟨transceiver⟩ tag of the MLR-Arena model. Gets the transceiver element with **sdf::ElementPtr transceiverElem = this-¿sdf-¿GetElement("transceiver")** The **Lines 73 and 74** gets the **gain** and **power** elements.

- The function **GetPower** and **GetGain** return the power and gain previously acquired

4.3 Steps taken and tentatives to fire up the WirelessPlugin

These are the tentatives to fire up the WirelessTransmitter and WirelessReceiver plugin:

4.4 Question - Gazebo answers

Formulation of the question of the Gazebo ROS. The main idea is to fire up a Transmitter plugin on the Husky robot, which broadcast some sort of wireless signals over the simulation or directly to a fix Receiver, that receives the signal or listens to it over the simulation

The main guiding questions:

- Where to place the .cpp and .h files to fire up the plugin and create the library file?
- How does the communication between the two tags works?
- On which topic should the plugin publish to?