MultiSense-SL Sensor
ROS Driver

Contact:

David LaRose
Director of Software Engineering
412-327-3970
dlr@carnegierobotics.com
Carnegie Robotics LLC
Ten 40th Street
Pittsburgh, PA 15201

Revision History

Rev.	Page Nos.	Description	Date	Name
1.0	Entire	Initial Pre-Release	24 Jan, 2013	S. Turner

Table of Contents

1. INSTALLATION 4
1.1. BUILDING FROM SOURCE 4
1.2. STACK LAYOUT 5
2. RUNNING 6
2.1. STARTING THE DRIVER 6
2.2. TOOLS 8
2.3. VISUALIZATION 9
3. ROS API DOCUMENTATION 10
3.1. CAMERA SUBSYSTEM 10
3.1.1. Published Topics 10
3.1.2. Static Parameters 11
3.1.3. Reconfigurable Parameters 11
3.2. LASER SUBSYSTEM 11
3.2.1. Published Topics 11
3.2.2. Static Parameters 11
3.2.3. Spindle Control Action 11
3.3. LED SUBSYSTEM 12
3.3.1. LED Control Action 12

1. INSTALLATION

1.1. BUILDING FROM SOURCE

The installation instructions are designed to work with Ubuntu 12.04 (Precise) and the ROS Fuerte release. To get started, add ROS to your apt sources by following the installation instructions on <http://www.ros.org/wiki/fuerte/Installation/Ubuntu>

Now you can install all ROS dependencies you need for the MultiSense-SL sensor:

```
sudo apt-get update

sudo apt-get install ros-fuerte-ros ros-fuerte-ros-comm ros-fuerte-laser-pipeline ros-fuerte-image-common
ros-fuerte-common-msgs ros-fuerte-diagnostics ros-fuerte-dynamic-reconfigure ros-fuerte-actionlib
ros-fuerte-vision-opencv ros-fuerte-diagnostics-monitors
ros-fuerte-geometry ros-fuerte-visualization ros-fuerte-robot-model
```

Then, install some basic tools to work with the source repository:

```
sudo apt-get install mercurial python-rosinstall build-essential cmake
```

Now you can check out the source code for the MultiSense-SL sensor. The commands below only work if the public ssh key of your host was registered with the repository.

```
echo "- hg: {local-name: multisense, uri:
'ssh://hg@bitbucket.org/hidof/cr_multisense', version: default}" >
/tmp/multisense.rosinstall

rosinstall path_to_source /opt/ros/fuerte /tmp/multisense.rosinstall
```

Now you can build the entire repository by invoking rosmake:

```
. path_to_source/setup.bash
rosmake multisense
```

1.2. STACK LAYOUT

The MultiSense stack contains nine ROS packages:

- **multisense_bringup**

This is the set of launch files and configuration files to start the ROS driver, together with a number of associated nodes for diagnostics, etc.

- **multisense_dashboard**

This is a GUI to get a full overview of the status of all components in the ROS driver.

- **multisense_description**

The URDF <ros.org/wiki/urdf> robot description XML file and associated meshes that represent the sensor head, sensor placement and kinematic structure.

- **multisense_diagnonstics**

A set of ROS nodes to generate diagnostics messages that are sent to the

multisense dashboard for visualization

- **multisense_lib**

This is the library implementing the wire protocol for communication with the MultiSense-SL sensor.

- **multisense_driver**

This is the core library that provides a callback mechanism on top of the wire protocol library. The callback mechanism allows the ROS drivers to register/deregister callback functions associated with specific message types.

- **multisense_msgs**

This is the set of ROS messages and action specifications that are used in the public ROS API. These are the messages that are specific to the MultiSense-SL device.

- **multisense_ros**

This package contains the actual ROS drivers for each of the sensors and/or actuators. There is a ROS driver for the Laser, Camera, Spindle and LED.

- **multisense_test**

This is a test package that simulates the MultiSense-SL sensor at the UDP level.

2. RUNNING

When running any of the commands in the rest of this document, make sure that you have set your environment in the terminal you're working in by running:

```
. path_to_source/setup.bash
```

2.1. STARTING THE DRIVER

Before running the ROS driver, make sure your network is configured correctly. The multisense_bringup package contains an example configuration script called:

configureNetwork.sh.

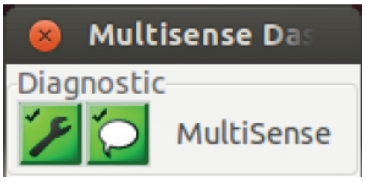
With your network interface configured, you can start the ROS driver with a single command:

```
roslaunch multisense_bringup multisense.launch
```

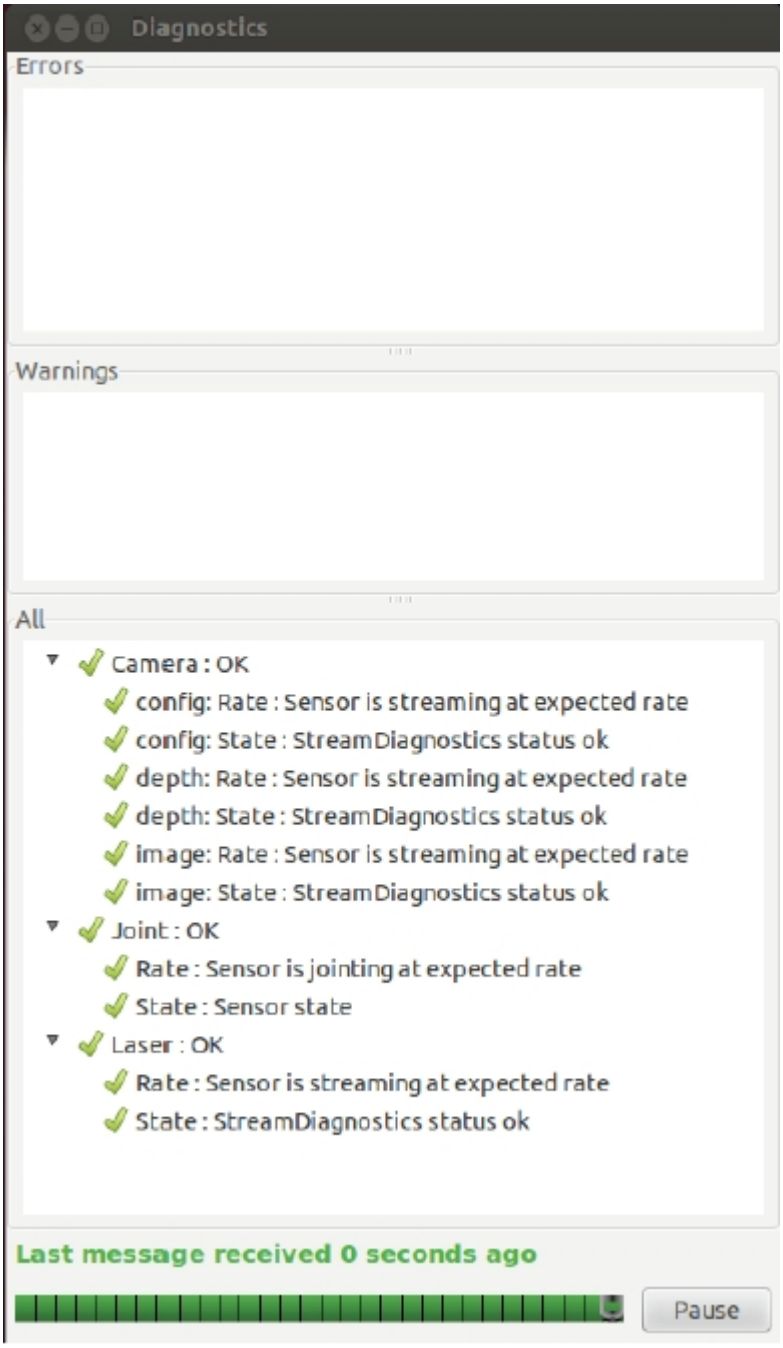
When running the driver, you should always run the dashboard too. This is your number one tool to check the status of the driver.

```
rosrun multisense_dashboard multisense_dashboard
```

The dashboard looks like this, with the left button for diagnostics, and the right button for logging messages. When everything is working well, both buttons will be green.



When you click on the left diagnostics button, you see an overview of the device sensors, and their status. There are three categories shown: camera, joint, and laser, with the camera category split up in: configuration, image, and depth. For each item in the list you get to see the current status (running, stopping, stopped, and starting), and the rate at which the corresponding data stream is coming in. Each of the items can be green (good), orange (warning), or red (error).

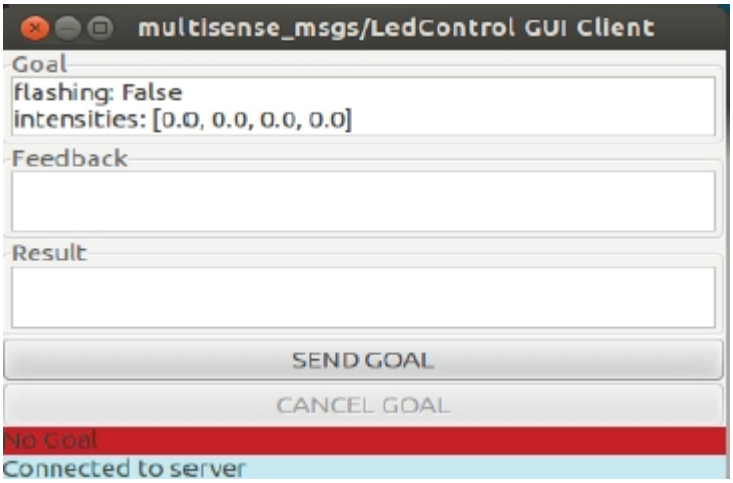


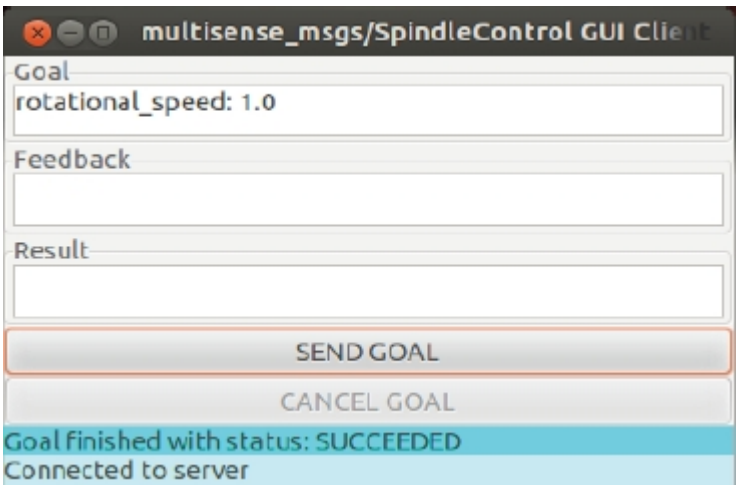
2.2. TOOLS

To control the LEDs and the spindle rotational speed, the ROS driver provides two action interfaces. These interfaces are designed to be controlled by other ROS nodes, but there is also a generic actionlib GUI tool that allows you to communicate with the action interface. You can bring up the GUI by running:

```
roslaunch multisense_bringup control.launch
```

This will bring up the two windows below, one to control the LEDs and one to control the spindle rotational speed. Simply modify the numbers in the goal field, and click on SEND GOAL.



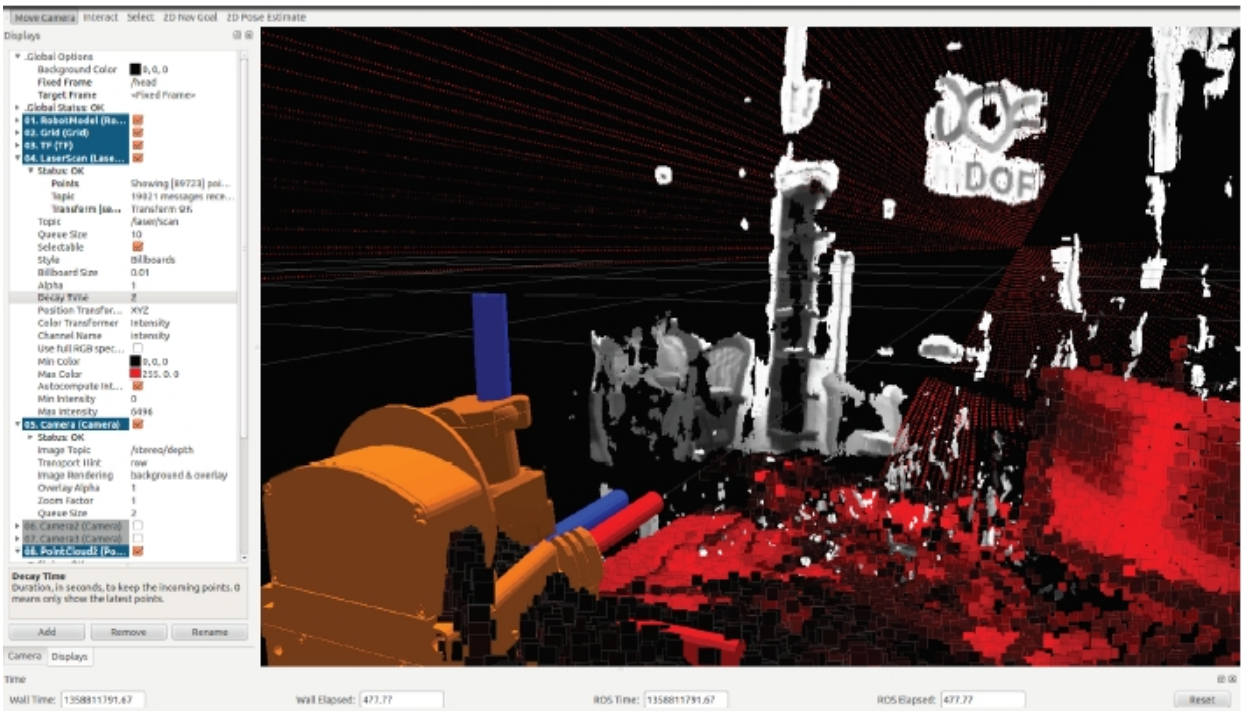


2.3. VISUALIZATION

ROS comes with an amazing 3D visualization tool called RViz. You can start RViz by running:

```
rosrun rviz rviz
```

Once you get to the the RViz window, click on file -> Open Config, and select the file called rviz_config in the multisense_bringup package. This will load the correct configuration to visualize all sensors on the Multisense-SL sensor.



3. ROS API DOCUMENTATION

The ROS API for the Multisense-SL sensor, provided by the ros_driver executable, is split into distinct subsystems for each piece of hardware. Specifically, the camera, laser, spindle joint, and LEDs are all placed in a namespace exposing an interface to that piece of hardware. Each of these subsystems are documented below.

Streams from the sensor are initiated on an "on-demand" basis. That is, on subscription of a given data stream (such as /camera/left/image_rect), the driver will initialize the stream on the sensor itself. Thus, the driver uses very little bandwidth and CPU when no one has subscribed to any of the ROS topics.

3.1. CAMERA SUBSYSTEM

3.1.1. Published Topics

Depth Camera

/camera/depth/camera_info (sensor_msgs/CameraInfo)

Camera projection matrix and metadata.

/camera/depth/depth (sensor_msgs/Image)

Depth image. Uses canonical representation (float32 meters). Depth images

Replace disparity images as the standard. (See REP_118)

Subtopics /compressed & /theora are for use with image_transport.

/camera/points2 (sensor_msgs/PointCloud2)

Stereo point cloud. Each point contains 4 fields (x, y, z, rgb) See the pcl for more information.

Left Camera

/camera/left/camera_info (sensor_msgs/CameraInfo)

Camera projection matrix and metadata.

/camera/left/image_rect (sensor_msgs/Image)

Rectified images from left camera in mono8 format

Subtopics /compressed & /theora are for use with image_transport

Right Camera

/camera/right/camera_info (sensor_msgs/CameraInfo)

Camera projection matrix and metadata.

/camera/right/image_rect (sensor_msgs/Image)

Rectified images from right camera in mono8 format

Subtopics /compressed & /theora are for use with image_transport

3.1.2. Static Parameters

/stereo/frame_id (string)

Used to populate the frame_id field in the camera message headers. Used by tf.

3.1.3. Reconfigurable Parameters

These camera parameters employ dynamic_reconfigure, and can be modified as runtime via "rosrun dynamic_reconfigure reconfigure_gui"

/stereo

exposure_time (float 32)

Time (in seconds) for camera exposure.

Gain (float 32)

Camera gain.

Fps (float 32)

Frames per second.

3.2. LASER SUBSYSTEM

3.2.1. Published Topics

/laser/scan (sensor_msgs/LaserScan)

Raw laser scan from device.

/joint_states (sensor_msgs/JointState

Position and velocity of the laser joint, named "hokuyo joint". This is used by the robot_state_publisher to publish /tf messages.

3.2.2. Static Parameters

/laser/frame_id (string)

used to populate the frame_id field in the laser scan header. Used by tf.

3.2.3. Spindle Control Action

The laser spindle is controlled using the action interface specified below.

/laser_joint/spindle_control (multisense_msgs/SpindleControlAction)

goal (multisense_msgs/SpindleControlActionGoal)

rotational_speed (float32)

The desired speed of the laser spindle (in RPM)

3.3. LED SUBSYSTEM

3.3.1. LED Control Action

The LED strobing and brightness is controlled using the action interface specified below

/led_control (multisense_msgs/LEDControlAction)

goal (multisense_msgs/LedControlActionGoal)

flashing (bool)

True -> LEDs strobe, staying on only during the camera exposure

False -> LEDs stay continuously on while the cameras are streaming

intensities (float32[4])

intensity setting for each LED, where 0.0 is off, and 1.0 is full brightness