ROS2: The evolution of Robot Operative System

Jose Luis Rivero
Open Robotics
We create open software and hardware platforms for robotics. We use those platforms to solve important problems and we help others to do the same.
The origin of ROS 1.x

First release 2007 - Ongoing
ROS: open source SDK to build robot software

ROS principles

1. Distribution
2. Abstraction
3. Introspection
4. Federation

ROS
Price: ~1100€
Weight: 1Kg
360° Laser Distance Sensor
LDS-01

Wheels
(Dynamixel actuators)

Remote controller
ps3joy

Remote controller

Documentation Status

joystick_drivers: joy | ps3joy | spacenv_node | wlimote

Package Summary

✓Released✓Continuous Integration: 1 / 1✓Documented

PlayStation 3 SIXAXIS or DUAL SHOCK 3 joystick driver. Driver for the Sony PlayStation 3 SIXAXIS, DUAL SHOCK 3 joysticks. In its current state, this driver is not compatible with PlayStation 3 wireless Bluetooth HID devices. The driver listens for a connection on the HID ports, streams data, and passes the data to the Linux input device so that it resembles a normal joystick.
ROSMASTER

register node and /joy

joy_node

/publisher

/joy

subscribe

teleop_twist_joy

/publish

/cmd_vel

receive metadata

ROS

register and naming and registration
Distribution

ROS

- Publish / subscribe messaging
- Discovery, transport, serialization
- Isolate components from each other
- Allow independent development
Raw Message Definition

```
# Reports the state of a joysticks axes and buttons
Header header
float32[] axes # the axes measurements from
int32[] buttons # the buttons measurements
```

# This expresses velocity in free space
Vector3 linear
Vector3 angular

- Well-defined interfaces
- Syntax & semantics
- Tools target generic interfaces
- Support multiple languages
360° Laser Distance Sensor LDS-01

ROS.org

Documentation Browse Software News

hls_lfcd_lds_driver

Package Summary

ROS package for LDS(HLS-LFCD2). The LDS (Laser Distance Sensor) is a sensor sending the data to Host for the simultaneous localization and mapping (SLAM). Simultaneously the detecting obstacle data can also be sent to Host. HLDS(Hitachi-LG Data Storage) is developing the technology for the moving platform sensor such as Robot Vacuum Cleaners, Home Robot, Robotics Lawn Mower Sensor, etc.

Package
Code AP
FAQ
Changelog
Change List
Reviews

Dependencies (4)
Used by (1)
Jenkins jobs (9)
Raw Message Definition

```python
# Single scan from a planar laser range-finder
#
# If you have another ranging device with different behavior (e.g. a sonar
# array), please find or create a different message, since applications
# will make fairly laser-specific assumptions about this data

Header header
# timestamp in the header is the acquisition time
# the first ray in the scan.
#
# in frame frame_id, angles are measured around
# the positive Z axis (counterclockwise, if Z is
# with zero angle being forward along the x axis

float32 angle_min  # start angle of the scan [rad]
float32 angle_max  # end angle of the scan [rad]
float32 angle_increment  # angular distance between measurements [rad]

float32 time_increment  # time between measurements [seconds] - if your 
# is moving, this will be used in interpolating 
# of 3d points

float32 scan_time  # time between scans [seconds]

float32 range_min  # minimum range value [m]
float32 range_max  # maximum range value [m]

float32[] ranges  # range data [m] (Note: values < range_min or > range_max
# intensity data [device-specific units]. If you
# device does not provide intensities, please leave
# the array empty.
```
### Topic Monitor

<table>
<thead>
<tr>
<th>Topic</th>
<th>Type</th>
<th>Bandwidth</th>
<th>Hz</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>/diagnostics</code></td>
<td>diagnostic_msgs/DiagnosticArray</td>
<td>276.41B/s</td>
<td>1.00</td>
<td>'MPU9250'</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>0</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>'Good Condition'</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>'IMU Sensor'</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>[ ]</td>
</tr>
<tr>
<td></td>
<td>std_msgs/Header</td>
<td></td>
<td></td>
<td>'DYNAMIXEL X'</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>0</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>'Torque ON'</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>'Actuator'</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>[ ]</td>
</tr>
<tr>
<td></td>
<td>diagnostic_msgs/DiagnosticStatus</td>
<td></td>
<td></td>
<td>'HLS-LFCD-LDS'</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>0</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>'Good Condition'</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>'Lidar Sensor'</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>[ ]</td>
</tr>
<tr>
<td></td>
<td>diagnostic_msgs/DiagnosticStatus</td>
<td></td>
<td></td>
<td>'Battery'</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>0</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>'Good Condition'</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>'Power System'</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>[ ]</td>
</tr>
<tr>
<td></td>
<td>diagnostic_msgs/DiagnosticStatus</td>
<td></td>
<td></td>
<td>'OpenCR Button'</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>0</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>'Pushed Nothing'</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>'Analog Button'</td>
</tr>
</tbody>
</table>
3 Introspection

ROS

- All important data on the message bus
- Support incremental system exploration
- GUIs are always external tools
- Could build apps from CLI tools
Wheels (Dynamixel actuators)

32 bits
Microcontroller

RaspBerry Pi
Wheels (Dynamixel actuators)
<table>
<thead>
<tr>
<th>Name</th>
<th>Packages</th>
</tr>
</thead>
<tbody>
<tr>
<td>rqt</td>
<td>rqt, rqt_gui</td>
</tr>
<tr>
<td>rosbag2_bag_v2</td>
<td>ros1_rosbag_store</td>
</tr>
<tr>
<td>qt_gui_core</td>
<td>qt, dotgraph, qt</td>
</tr>
<tr>
<td>gazeboros_pkgs</td>
<td>gazebo_dev, gazebo</td>
</tr>
<tr>
<td>realtime_tools</td>
<td>system_modes, system_modes</td>
</tr>
<tr>
<td>vision_opencv</td>
<td>cv_bridge, image</td>
</tr>
<tr>
<td>teleop_twist_joy</td>
<td>teleop_twist_joy</td>
</tr>
<tr>
<td>rosidl_python</td>
<td>python3 cmake, python3</td>
</tr>
<tr>
<td>ros2_object_analytics</td>
<td>object_analytics, realisense_camera</td>
</tr>
<tr>
<td>ros2_intel_realsense</td>
<td></td>
</tr>
<tr>
<td>rclcpp</td>
<td>rclcpp, rclcpp_e</td>
</tr>
<tr>
<td>launch_ros</td>
<td>launch_ros, launch</td>
</tr>
<tr>
<td>joystick_drivers</td>
<td>joy</td>
</tr>
</tbody>
</table>
4 Federation

- Let the code live where authors prefer
- Allow for independent releases
- Lower barrier for contributors
- Authors have control, branding, credit
- Requires tools for devs and CI
The next generation

First release 2018 - Ongoing
ROS 2: Goals

1. Quality of design & implementation
2. System reliability
3. Real-time control & deterministic execution
4. Validation, verification, and certification
5. Flexibility in communication
6. Support for small embedded systems
Use case:
unstable networks or high latency scenarios
## Architectural changes

### ROS 1

- **user code (nodes)**
  - **Master**
  - **C++ client library** (roscpp)
  - **Python client library** (rospy)
  - **TCPROS/UPDROS**
  - **Linux**
  - **Microprocessor**

### Towards ROS2

- **user code (nodes)**
  - **Master**
  - **rclcpp (C++)**
  - **rclpy (Python)**
  - **rcljava (Java)**
  - **ROS client library** (rcl)
  - **DDS Standard**
  - **Linux / Mac / Windows**
  - **Microprocessor**
Data-Distribution Service Standard

“The OMG Data-Distribution Service for Real-Time Systems (DDS) is the first open international middleware standard directly addressing publish-subscribe communications for real-time and embedded systems.”

“... DDS features fine and extensive control of QoS parameters, including reliability, bandwidth, delivery deadlines, and resource limits. ...”
Architectural changes

ROS 1

- user code (nodes)
- Master
- C++ client library (roscpp)
- Python client library (rospy)
- TCPROS/UPDROS
- Linux
- Microprocessor

Towards ROS2

- user code (nodes)
- Master
- rclcpp (C++)
- rclpy (Python)
- rcljava (Java)
- ROS client library (rcl)
- ROS abstract middleware (rmw)
- Cyclone DDS
- FastRTPS
- Eclipse iceoryx
- Linux / Mac / Windows
- Microprocessor
Use case: manage groups of robots
ROS MASTER

Where? ....
“Effective data communication between publishers and subscribers requires dynamic and reliable discovery of publisher/subscriber endpoints in the system, which DDS currently supports via a standardized approach called the Simple Discovery Protocol (SDP)”[1]

Use case: small systems as first-class ROS systems
<table>
<thead>
<tr>
<th></th>
<th>8/16-bit MCU</th>
<th>32-bit MCU</th>
<th>ARM A-class smartphone without screen</th>
<th>SFF x86 laptop without screen</th>
</tr>
</thead>
<tbody>
<tr>
<td>Example Chip</td>
<td>Atmel AVR</td>
<td>STM32</td>
<td>Samsung Exynos</td>
<td>Intel Core i5</td>
</tr>
<tr>
<td>Example System</td>
<td>Arduino Leonardo</td>
<td>Pixhawk PX4</td>
<td>ODROID</td>
<td>Intel NUC</td>
</tr>
<tr>
<td>MIPS</td>
<td>10's</td>
<td>100's</td>
<td>1000's</td>
<td>10000's</td>
</tr>
<tr>
<td>RAM</td>
<td>1-32 KB</td>
<td>4-256 KB</td>
<td>a few GB (off-chip)</td>
<td>2-16 GB (SODIMM)</td>
</tr>
<tr>
<td>Max power</td>
<td>10's of mW</td>
<td>100's of mW</td>
<td>1000's of mW</td>
<td>10000's of mW</td>
</tr>
<tr>
<td>Comms peripherals</td>
<td>UART, USB FS, ...</td>
<td>USB HS, Ethernet</td>
<td>Gigabit Ethernet</td>
<td>USB SS, PCIe</td>
</tr>
</tbody>
</table>

"normal" ROS2

Future work

https://roscon.ros.org/2015/presentations/ros2_on_small_embedded_systems.pdf
Wheels (Dynamixel actuators)

32 bits Microcontroller

RasPBerry Pi
32 bits Microcontroller

Rosserial protocol

battery state
IMU values

Velocity and servo commands

ROS

Publish

/battery_state
/imu
/cmd_vel
### ROS2

- **user code (nodes)**
- **rclcpp (C++)**
- **rclpy (Python)**
- **rcldjava (Java)**
- **ROS client library (rcl)**
- **ROS abstract middleware (rmw)**
- **Connext DDS**
- **FastRTPS**
- **Eclipse iceoryx**
- **Linux / Mac / Windows**
- **Microprocessor**

### Micro-ROS

- **user code (nodes)**
- **rclcpp (C++)**
- **ROS client library (rcl)**
- **Predictable Execution**
- **ROS abstract middleware (rmw)**
- **XRCE-DDS**
  - (eXtremely Resource Constrained Environments)
- **NuttX (RTOS)**
- **Microcontroller**
32 bits Microcontroller

ROS

battery state
IMU values

velocity and servo commands

rosserial protocol

publish

/battery_state

/imu

/cmd_vel

Raspberry Pi
32 bits
Microcontroller

Raspberry Pi

turtlebot3_core

publish

/battery_state
/imu
/cmd_vel
More and more features ...
ROS 2: other features

### Real time capabilities
- ROS2 design facilitates to implement real-time compliant
- Not only Micro-ROS, also other approaches
- Not only NuttX but FreeRTOS, VxWorks or QNX

### Security
- ROS1 has no security by design feature not a bug ;)
- ROS2 integrates security from DDS
- Features are: authentication, access control, cryptographic support, etc.

More: lifecycle for nodes, run multiples nodes in a single process, deterministic starting sequence, OpenEmbedded/Yocto support, etc.
Thanks! Questions?