IntelRealSense / **realsense-ros**

Intel(R) RealSense(TM) ROS Wrapper for D400 series, SR300 Camera and T265 Tracking Module

wiki.ros.org/realsense

Apache-2.0 License

1.1k stars **964** forks

Star Watch

Code Issues 71 Pull requests 18 Discussions Actions Projects Wiki Security

development

doronhi 2.2.23 ··· 10 days ago **994**

View code

ROS Wrapper for Intel® RealSense™ Devices

These are packages for using Intel RealSense cameras (D400 series SR300 camera and T265 Tracking Module) with ROS.

This version supports Kinetic, Melodic and Noetic distributions.

For running in ROS2 environment please switch to the ros2 branch.

LibRealSense supported version: v2.43.0 (see realsense2_camera release notes)

Installation Instructions

README.md

Step 1: Install the ROS distribution

- **Install ROS Kinetic, on Ubuntu 16.04, ROS Melodic on Ubuntu 18.04 or ROS Noetic on Ubuntu 20.04.**

Windows

Step 1: Install the ROS distribution

- **Install ROS Melodic or later on Windows 10**

There are 2 sources to install realsense2_camera from:

- ## Method 1: The ROS distribution:

  Ubuntu

  realsense2_camera is available as a debian package of ROS distribution. It can be installed by typing:

  ```
  export ROS_VER=kinetic
  ```

  or

  ```
  export ROS_VER=melodic
  ```

or

```
export ROS_VER=noetic
```

Then install the ros packages using the environment variable created above:

```
sudo apt-get install ros-$ROS_VER-realsense2-camera
```

This will install both realsense2_camera and its dependents, including librealsense2 library.

Notice:

- The version of librealsense2 is almost always behind the one availeable in RealSense™ official repository.
- librealsense2 is not built to use native v4l2 driver but the less stable RS-USB protocol. That is because the last is more general and operational on a larger variety of platforms.
- realsense2_description is available as a separate debian package of ROS distribution. It includes the 3D-models of the devices and is necessary for running launch files that include these models (i.e. rs_d435_camera_with_model.launch). It can be installed by typing: `sudo apt-get install ros-$ROS_VER-realsense2-description`

Windows

Chocolatey distribution Coming soon

- ## Method 2: The RealSense™ distribution:

> This option is demonstrated in the .travis.yml file. It basically summerize the elaborate instructions in the following 2 steps:

Step 1: Install the latest Intel® RealSense™ SDK 2.0

Ubuntu

- Install from Debian Package
  - In that case treat yourself as a developer. Make sure you follow the instructions to also install librealsense2-dev and librealsense-dkms packages.

Windows Install using vcpkg

```
`vcpkg install realsense2:x64-windows`
```

OR

- **Build from sources by downloading the latest Intel® RealSense™ SDK 2.0 and follow the instructions under Linux Installation**

Step 2: Install Intel® RealSense™ ROS from Sources

- Create a catkin workspace *Ubuntu*

```
mkdir -p ~/catkin_ws/src
cd ~/catkin_ws/src/
```

Windows

```
mkdir c:\catkin_ws\src
cd c:\catkin_ws\src
```

- Clone the latest Intel® RealSense™ ROS from here into 'catkin_ws/src/'

```
git clone https://github.com/IntelRealSense/realsense-ros.git
cd realsense-ros/
git checkout `git tag | sort -V | grep -P "^2.\d+\.\d+" | tail -1`
cd ..
```

- Make sure all dependent packages are installed. You can check .travis.yml file for reference.
- Specifically, make sure that the ros package *ddynamic_reconfigure* is installed. If *ddynamic_reconfigure* cannot be installed using APT or if you are using *Windows* you may clone it into your workspace 'catkin_ws/src/' from here

```
catkin_init_workspace
cd ..
catkin_make clean
catkin_make -DCATKIN_ENABLE_TESTING=False -DCMAKE_BUILD_TYPE=Release
catkin_make install
```

Ubuntu

```
echo "source ~/catkin_ws/devel/setup.bash" >> ~/.bashrc
source ~/.bashrc
```

Windows

```
devel\setup.bat
```

Usage Instructions

Start the camera node

To start the camera node in ROS:

```
roslaunch realsense2_camera rs_camera.launch
```

This will stream all camera sensors and publish on the appropriate ROS topics.

Other stream resolutions and frame rates can optionally be provided as parameters to the 'rs_camera.launch' file.

Published Topics

The published topics differ according to the device and parameters. After running the above command with D435i attached, the following list of topics will be available (This is a partial list. For full one type `rostopic list`):

- /camera/color/camera_info
- /camera/color/image_raw
- /camera/depth/camera_info
- /camera/depth/image_rect_raw
- /camera/extrinsics/depth_to_color
- /camera/extrinsics/depth_to_infra1
- /camera/extrinsics/depth_to_infra2
- /camera/infra1/camera_info
- /camera/infra1/image_rect_raw
- /camera/infra2/camera_info
- /camera/infra2/image_rect_raw
- /camera/gyro/imu_info
- /camera/gyro/sample
- /camera/accel/imu_info
- /camera/accel/sample
- /diagnostics

> Using an L515 device the list differs a little by adding a 4-bit confidence grade (pulished as a mono8 image):
>
> - /camera/confidence/camera_info
> - /camera/confidence/image_rect_raw
>
> It also replaces the 2 infrared topics with the single available one:
>
> - /camera/infra/camera_info

- /camera/infra/image_raw

The "/camera" prefix is the default and can be changed. Check the rs_multiple_devices.launch file for an example. If using D435 or D415, the gyro and accel topics wont be available. Likewise, other topics will be available when using T265 (see below).

Available services:

- reset : Cause a hardware reset of the device. Usage: `rosservice call /camera/realsense2_camera/reset`

Launch parameters

The following parameters are available by the wrapper:

- **serial_no**: will attach to the device with the given serial number (*serial_no*) number. Default, attach to available RealSense device in random.
- **usb_port_id**: will attach to the device with the given USB port (*usb_port_id*). i.e 4-1, 4-2 etc. Default, ignore USB port when choosing a device.
- **device_type**: will attach to a device whose name includes the given *device_type* regular expression pattern. Default, ignore device type. For example, device_type:=d435 will match d435 and d435i. device_type=d435(?!i) will match d435 but not d435i.
- **rosbag_filename**: Will publish topics from rosbag file.
- **initial_reset**: On occasions the device was not closed properly and due to firmware issues needs to reset. If set to true, the device will reset prior to usage.
- **align_depth**: If set to true, will publish additional topics with the all the images aligned to the depth image.
  The topics are of the form: `/camera/aligned_depth_to_color/image_raw` etc.
- **filters**: any of the following options, separated by commas:

- `colorizer` : will color the depth image. On the depth topic an RGB image will be published, instead of the 16bit depth values .

- `pointcloud` : will add a pointcloud topic `/camera/depth/color/points` .

  - The texture of the pointcloud can be modified in rqt_reconfigure (see below) or using the parameters: `pointcloud_texture_stream` and `pointcloud_texture_index` . Run rqt_reconfigure to see available values for these parameters.
  - The depth FOV and the texture FOV are not similar. By default, pointcloud is limited to the section of depth containing the texture. You can have a full depth to pointcloud, coloring the regions beyond the texture with zeros, by setting `allow_no_texture_points` to true.
  - pointcloud is of an unordered format by default. This can be changed by setting `ordered_pc` to true.

- `hdr_merge` : Allows depth image to be created by merging the information from 2 consecutive frames, taken with different exposure and gain values. The way to set exposure and gain values for each sequence in runtime is by first selecting the sequence id, using rqt_reconfigure `stereo_module/sequence_id` parameter and then modifying the `stereo_module/gain` , and `stereo_module/exposure` .
  To view the effect on the infrared image for each sequence id use the `sequence_id_filter/sequence_id` parameter.
  To initialize these parameters in start time use the following parameters:
  `stereo_module/exposure/1` , `stereo_module/gain/1` , `stereo_module/exposure/2` , `stereo_module/gain/2`
  * For in-depth review of the subject please read the accompanying white paper.

- The following filters have detailed descriptions in : https://github.com/IntelRealSense/librealsense/blob/master/doc/post-processing-filters.md

  - `disparity` - convert depth to disparity before applying other filters and back.
  - `spatial` - filter the depth image spatially.
  - `temporal` - filter the depth image temporally.
  - `hole_filling` - apply hole-filling filter.
  - `decimation` - reduces depth scene complexity.

- **enable_sync**: gathers closest frames of different sensors, infra red, color and depth, to be sent with the same timetag. This happens automatically when such filters as pointcloud are enabled.

- **<*stream_type*>_width**, **<*stream_type*>_height**, **<*stream_type*>_fps**: <stream_type> can be any of *infra, color, fisheye, depth, gyro, accel, pose*. Sets the required format of the device. If the specified combination of parameters is not available by the device, the stream will be replaced with the default for that stream. Setting a value to 0, will choose the first format in the inner list. (i.e. consistent between runs but not defined).
  *Note: for gyro accel and pose, only _fps option is meaningful.

- **enable_<*stream_name*>**: Choose whether to enable a specified stream or not. Default is true for images and false for orientation streams. <stream_name> can be any of *infra1, infra2, color, depth, fisheye, fisheye1, fisheye2, gyro, accel, pose*.

- **tf_prefix**: By default all frame's ids have the same prefix - `camera_` . This allows changing it per camera.

- **<*stream_name*>_frame_id**, **<*stream_name*>_optical_frame_id**, **aligned_depth_to_<*stream_name*>_frame_id**: Specify the different frame_id for the different frames. Especially important when using multiple cameras.

- **base_frame_id**: defines the frame_id all static transformations refers to.

- **odom_frame_id**: defines the origin coordinate system in ROS convention (X-Forward, Y-Left, Z-Up). pose topic defines the pose relative to that system.

- **All the rest of the frame_ids can be found in the template launch file: nodelet.launch.xml**

- **unite_imu_method**: The D435i and T265 cameras have built in IMU components which produce 2 unrelated streams: *gyro* - which shows angular velocity and *accel* which shows linear acceleration. Each with it's own frequency. By default, 2 corresponding topics are available, each with only the relevant fields of the message sensor_msgs::Imu are filled out. Setting *unite_imu_method* creates a new topic, *imu*, that replaces the default *gyro* and *accel* topics. The *imu* topic is published at the rate of the gyro. All the fields of the Imu message under the *imu* topic are filled out.

  - **linear_interpolation**: Every gyro message is attached by the an accel message interpolated to the gyro's timestamp.

  - **copy**: Every gyro message is attached by the last accel message.

- **clip_distance**: remove from the depth image all values above a given value (meters). Disable by giving negative value (default)

- **linear_accel_cov**, **angular_velocity_cov**: sets the variance given to the Imu readings. For the T265, these values are being modified by the inner confidence value.

- **hold_back_imu_for_frames**: Images processing takes time. Therefor there is a time gap between the moment the image arrives at the wrapper and the moment the image is published to the ROS environment. During this time, Imu messages keep on arriving and a situation is created where an image with earlier timestamp is published after Imu message with later timestamp. If that is a problem, setting *hold_back_imu_for_frames* to *true* will hold the Imu messages back while processing the images and then publish them all in a burst, thus keeping the order of publication as the order of arrival. Note that in either case, the timestamp in each message's header reflects the time of it's origin.

- **topic_odom_in**: For T265, add wheel odometry information through this topic. The code refers only to the *twist.linear* field in the message.

- **calib_odom_file**: For the T265 to include odometry input, it must be given a configuration file. Explanations can be found here. The calibration is done in ROS coordinates system.

- **publish_tf**: boolean, publish or not TF at all. Defaults to True.

- **tf_publish_rate**: double, positive values mean dynamic transform publication with specified rate, all other values mean static transform publication. Defaults to 0

- **publish_odom_tf**: If True (default) publish TF from odom_frame to pose_frame.

- **infra_rgb**: When set to True (default: False), it configures the infrared camera to stream in RGB (color) mode, thus enabling the use of a RGB image in the same frame as the depth image, potentially avoiding frame transformation related errors. When this feature is required, you are additionally required to also enable `enable_infra:=true` for the infrared stream to be enabled.

  - **NOTE** The configuration required for `enable_infra` is independent of `enable_depth`

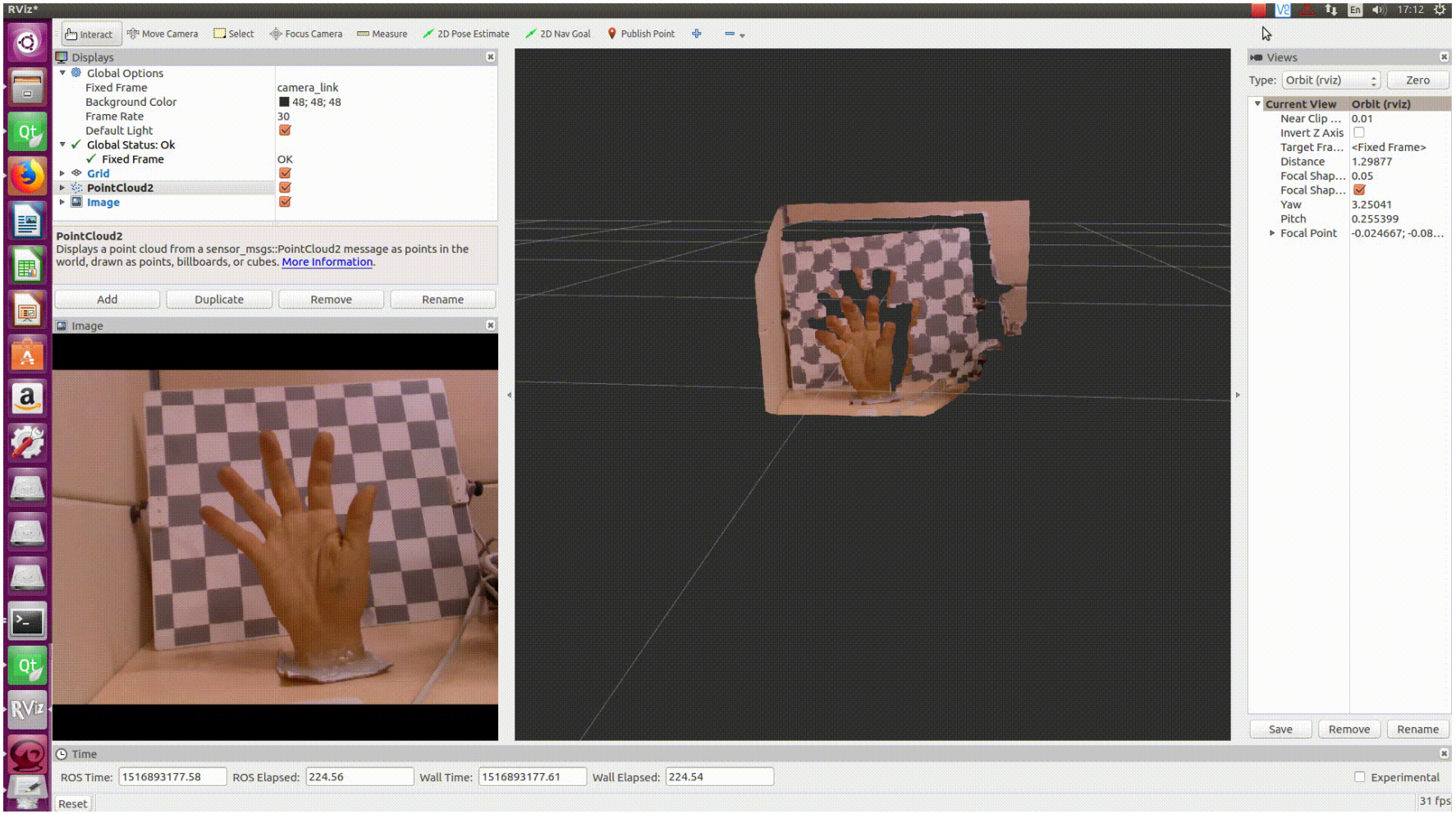
- **NOTE** To enable the Infrared stream, you should enable `enable_infra:=true` NOT `enable_infra1:=true` nor `enable_infra2:=true`
- **NOTE** This feature is only supported by Realsense sensors with RGB streams available from the `infra` cameras, which can be checked by observing the output of `rs-enumerate-devices`

Point Cloud

Here is an example of how to start the camera node and make it publish the point cloud using the pointcloud option.

```
roslaunch realsense2_camera rs_camera.launch filters:=pointcloud
```

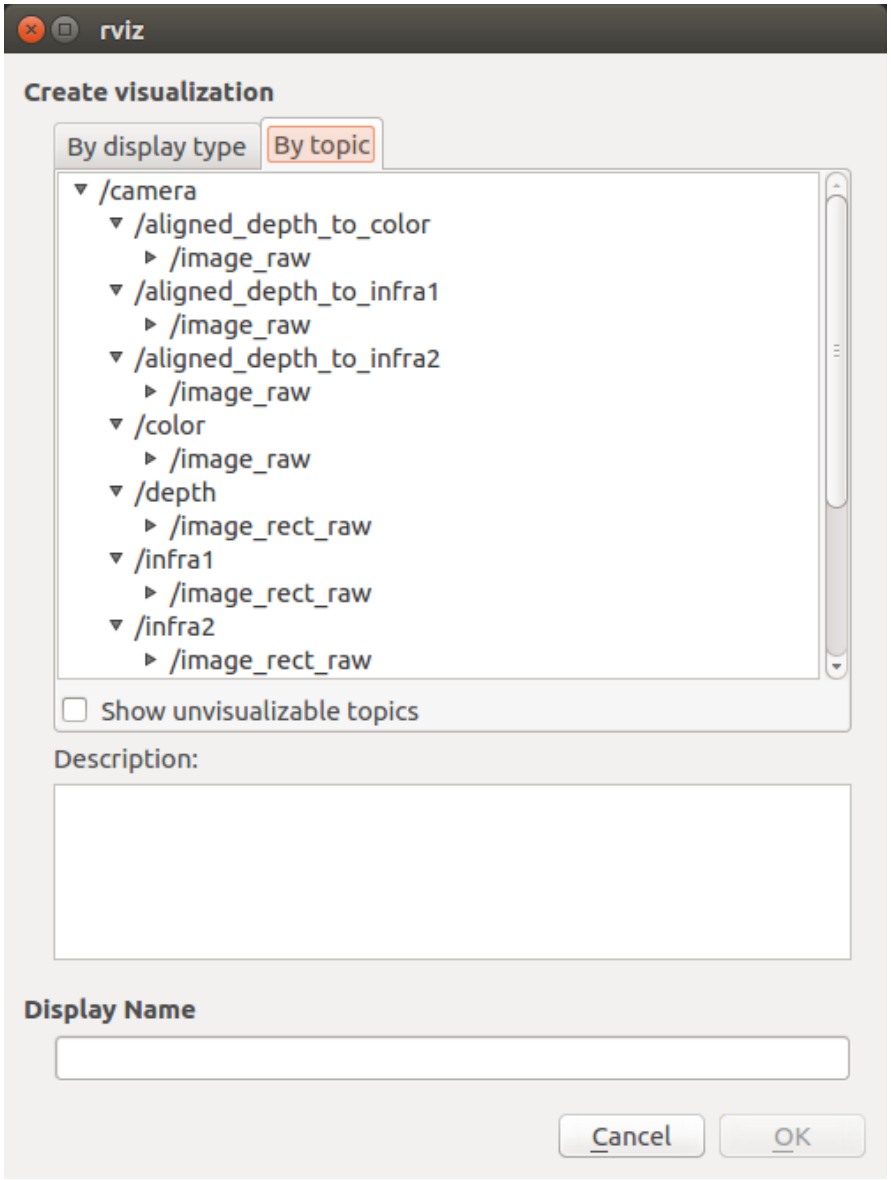
Then open rviz to watch the pointcloud:



Aligned Depth Frames

Here is an example of how to start the camera node and make it publish the aligned depth stream to other available streams such as color or infra-red.

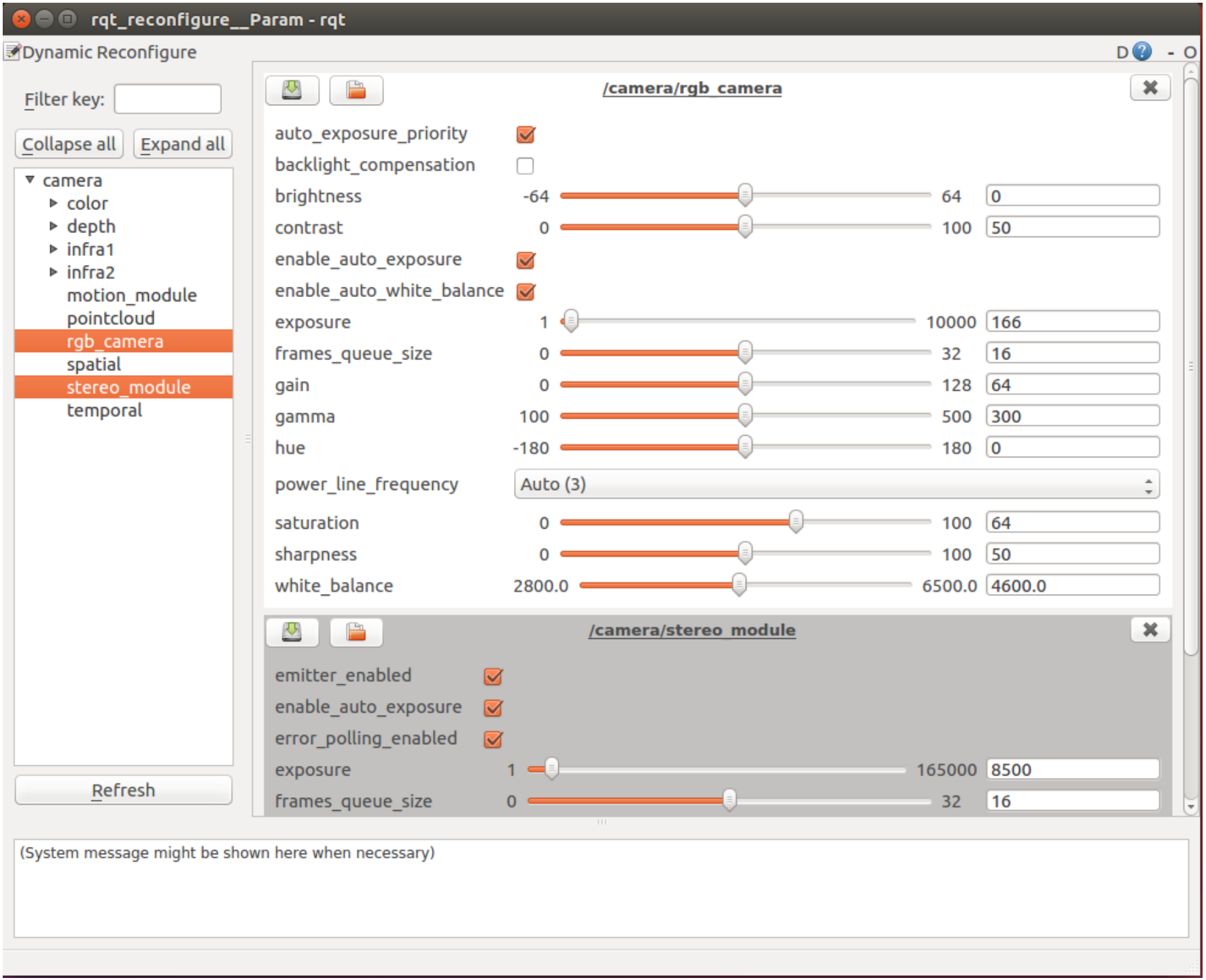
```
roslaunch realsense2_camera rs_camera.launch align_depth:=true
```



Set Camera Controls Using Dynamic Reconfigure Params

The following command allow to change camera control values using [http://wiki.ros.org/rqt_reconfigure].

```
rosrun rqt_reconfigure rqt_reconfigure
```


rqt_reconfigure__Param - rqt
Dynamic Reconfigure
Filter key:
Collapse all
Expand all
camera
color
depth
infra1
infra2
motion_module
pointcloud
rgb_camera
spatial
stereo_module
temporal
Refresh
/camera/rgb_camera
auto_exposure_priority
backlight_compensation
brightness -64 64 0
contrast 0 100 50
enable_auto_exposure
enable_auto_white_balance
exposure 1 10000 166
frames_queue_size 0 32 16
gain 0 128 64
gamma 100 500 300
hue -180 180 0
power_line_frequency Auto (3)
saturation 0 100 64
sharpness 0 100 50
white_balance 2800.0 6500.0 4600.0
/camera/stereo_module
emitter_enabled
enable_auto_exposure
error_polling_enabled
exposure 1 165000 8500
frames_queue_size 0 32 16
(System message might be shown here when necessary)

Work with multiple cameras

Important Notice: Launching multiple T265 cameras is currently not supported. This will be addressed in a later version.

Here is an example of how to start the camera node and streaming with two cameras using the rs_multiple_devices.launch.

```
roslaunch realsense2_camera rs_multiple_devices.launch serial_no_camera1:=<serial number of the first came
```

The camera serial number should be provided to `serial_no_camera1` and `serial_no_camera2` parameters. One way to get the serial number is from the rs-enumerate-devices tool.

```
rs-enumerate-devices | grep Serial
```

Another way of obtaining the serial number is connecting the camera alone, running

```
roslaunch realsense2_camera rs_camera.launch
```

and looking for the serial number in the log printed to screen under "[INFO][...]Device Serial No:".

Another way to use multiple cameras is running each from a different terminal. Make sure you set a different namespace for each camera using the "camera" argument:

```
roslaunch realsense2_camera rs_camera.launch camera:=cam_1 serial_no:=<serial number of the first camera>
roslaunch realsense2_camera rs_camera.launch camera:=cam_2 serial_no:=<serial number of the second camera>
...
```

Using T265

Start the camera node

To start the camera node in ROS:

```
roslaunch realsense2_camera rs_t265.launch
```

This will stream all camera sensors and publish on the appropriate ROS topics.

The T265 sets its usb unique ID during initialization and without this parameter it wont be found. Once running it will publish, among others, the following topics:

- /camera/odom/sample
- /camera/accel/sample
- /camera/gyro/sample
- /camera/fisheye1/image_raw
- /camera/fisheye2/image_raw

To visualize the pose output and frames in RViz, start:

```
roslaunch realsense2_camera demo_t265.launch
```

About Frame ID

The wrapper publishes static transformations(TFs). The Frame Ids are divided into 3 groups:

- ROS convention frames: follow the format of <tf_prefix>_<_stream>"_frame" for example: camera_depth_frame, camera_infra1_frame, etc.
- Original frame coordinate system: with the suffix of <_optical_frame>. For example: camera_infra1_optical_frame. Check the device documentation for specific coordinate system for each stream.

- base_link: For example: camera_link. A reference frame for the device. In D400 series and SR300 it is the depth frame. In T265, the pose frame.

realsense2_description package:

For viewing included models, a separate package is included. For example:

```
roslaunch realsense2_description view_d415_model.launch
```

Unit tests:

Unit-tests are based on bag files saved on S3 server. These can be downloaded using the following commands:

```
cd catkin_ws
wget "http://realsense-hw-public.s3.amazonaws.com/rs-tests/TestData/outdoors.bag" -P "records/"
wget "http://realsense-hw-public.s3-eu-west-1.amazonaws.com/rs-tests/D435i_Depth_and_IMU_Stands_still.bag"
```

Then, unit-tests can be run using the following command (use either python or python3):

```
python src/realsense/realsense2_camera/scripts/rs2_test.py --all
```

Packages using RealSense ROS Camera

Title	Links
ROS Object Analytics	github / ROS Wiki

Known Issues

- This ROS node does not currently support ROS Lunar Loggerhead.
- This ROS node currently does not support running multiple T265 cameras at once. This will be addressed in a future update.

License

Releases 61

ROS2-Foxy Wrapper for Intel® RealSense™ Devices (build 3.1.5) Latest
10 days ago

+ 60 releases

Packages

No packages published

Contributors 77

+ 66 contributors

Languages

- **C++** 76.5%
- **Python** 21.4%
- **CMake** 2.1%