# 00_SLAM_US guide

> *For building a SLAM system using RealSense cameras and RTAB-Map*
> *Recommended to ensure a clean and stable system + Ubuntu 22.04 LTS first*

## 📚 Official Documentation Recommendations (Bookmark these first!)

| Project | Official Documentation Link |
|---|---|
| ROS 2 Humble Installation | https://docs.ros.org/en/humble/Installation/Ubuntu-Install-Debians.html |
| ROS 2 Basic Tutorials | https://docs.ros.org/en/humble/Tutorials.html |
| RealSense SDK | https://github.com/IntelRealSense/librealsense |
| realsense-ros Driver | https://github.com/IntelRealSense/realsense-ros |
| RTAB-Map ROS 2 | https://github.com/introlab/rtabmap_ros |

## Installation & Basic Environment

### IntelRealSense/realsense-ros

1. **Install ROS2 Humble**
   - First, you need to install the ROS2 Humble version on your system.
   - If you are new to ROS2, you can first check out the ROS official documentation, follow the steps, and remember to choose the correct corresponding Ubuntu version (generally 22.04 corresponds to ROS2 Humble).
2. **Install the latest Intel® RealSense™ SDK 2.0**
   - This SDK is key to enabling your operating system to recognize and operate the RealSense camera.
   - For specific steps, refer to Intel's official documentation or community tutorials. Common methods are using apt or compiling from source code (if you want the latest features).
3. **Install Intel® RealSense™ ROS2 wrapper**
   - This package allows RealSense camera data to flow directly into ROS2, which is very convenient.
   - The installation method is basically downloading the source code in an environment where ROS2 is already installed and then compiling it together.

> *Small reminder: If you want the system to find the RealSense immediately upon startup, after plugging in the camera, it's best to check `lsusb` to confirm if the device is recognized by the system, or use `realsense-viewer` to see if images can be successfully acquired.*

---

## Workspace (ros2_ws) Setup

- We usually create a `ros2_ws` under `~/` as our own ROS2 workspace.
- The typical structure of a workspace is:

```
ros2_ws/
 ├ src/
 ├ install/
 └ build/
```

- You put all the required ROS packages into `ros2_ws/src`, then return to the `ros2_ws` root directory to perform the compilation operation (mentioned below as `colcon build`), which will compile these packages together.

> *Tip: Remember to `source install/setup.bash` after each compilation so you can use the packages you just compiled in the same terminal.*

---

## Required Packages: Rtabmap & Realsense

### Rtabmap

1. **Get the source code**

```
cd ~/ros2_ws/src
git clone -b humble-devel https://github.com/introlab/rtabmap_ros.git
```

   - This step will pull the `rtabmap_ros` humble-devel branch to your local machine.
   - If you are using another ROS2 version, please select the correct branch accordingly.

2. **In case of insufficient memory**
   - If your computer only has 8GB of RAM, compiling large ROS packages may use a lot of memory, causing the compilation to fail.
   - You can alleviate this by increasing the Swap partition:

```
sudo swapoff /swapfile
sudo fallocate -l 8G /swapfile  # Create an 8GB Swap file
sudo mkswap /swapfile
sudo swapon /swapfile
sudo swapon --show
```

   - After doing this, you will have an extra 8GB of swap for the system to use as "backup memory".

3. **Single-threaded compilation (saves memory)**
   - If memory is still tight, try single-threaded compilation:

```
cd ~/ros2_ws
colcon build --parallel-workers 1
```

   - Although the compilation speed will be slower, it can avoid compilation failure due to running out of memory.

### Realsense

1. **Clone realsense-ros source code**

```
cd ~/ros2_ws/src
git clone https://github.com/IntelRealSense/realsense-ros.git -b ros2-master
```

   - This pulls the `ros2-master` branch.
   - If there is another branch corresponding to your desired ROS2 version, you can switch.

2. **Install dependencies**

```
cd ~/ros2_ws
rosdep install -i --from-path src --rosdistro humble --skip-keys=librealsense2 -y
```

   - This command will help you install the dependency packages declared in the source code, filling in anything that's missing.

3. **Compile**

```
colcon build
```

   - As before, after compiling, you can `source install/setup.bash`.

---

## RealSense Driver Startup

1. **Start RealSense Node**
   - The simplest method is using the official Launch file:

```
ros2 launch realsense2_camera rs_camera.launch.py
```

   - This will publish depth, color, and IMU data (if your camera is a D435i or similar with IMU).

2. **Enable IMU**
   - You need to confirm in the corresponding `.launch.py` or YAML parameter file:

```
enable_gyro: true
enable_accel: true
publish_odom_tf: true
```

   - If you are writing your own parameter file, be sure to set these parameters to `true`.

> *Tip: If you only want images and don't want IMU data, you can turn off the IMU switch to save bandwidth and computing power.*

---

## RTAB-Map Odometry (Odom) Acquisition

`rtabmap_ros` can automatically calculate odometry and perform SLAM based on image/depth information, which is particularly suitable for scenarios requiring environmental map building or just wanting camera pose.

1. **Configure a new RTAB-Map Launch file**
   - You can create a new `my_rtabmap.launch.py` in the `ros2_ws/launch` directory, specifying camera topics, frame IDs, some RTAB-Map parameters, etc.
   - You can also refer to `realsense_d435i_stereo.launch.py` in `rtabmap_examples` to see how it's written.
2. **Launch this file**

   ```
   ros2 launch rtabmap_ros my_rtabmap.launch.py
   ```

   - Replace with your own filename.
3. **Visualize in RViz2**
   - Open another terminal and type `rviz2`.
   - Add the topics you are interested in under Displays:
     - `/odom`: Odometry
     - `/rtabmap/odom`: Odometry calculated by RTAB-Map
   - To check TF, you can run:

     ```
     ros2 run tf2_tools view_frames
     ros2 topic echo /tf_static
     ```

   - This can check if your Frames (e.g., map, odom, camera_link) are all properly connected.
4. **Using pose data**
   - If everything is normal, `/odom` or `/rtabmap/odom` will output the current camera pose (position + orientation) relative to the starting point (0,0,0).
   - You can subscribe to these topics in your own node for further processing, such as recording, analysis, or publishing to other modules.

---

## Data Visualization

- **RViz2**
  1. Type `rviz2` in the terminal.
  2. Inside, you can add various display types, such as:
     - **Odometry** → Subscribe to `/odom`
     - **Image** → Subscribe to `/camera/color/image_raw`
     - **Depth** → Subscribe to `/camera/depth/image_rect_raw`
     - **Map** → Subscribe to `/rtabmap/map`
     - **TF** → Subscribe to `/tf` and `/tf_static`
  3. This allows you to see real-time images, point clouds, odometry trajectories, maps, and other information simultaneously.

> *Tip: If RViz2 appears completely black, it might be due to incorrect camera image format or Frame ID configuration. Carefully check the corresponding Fixed Frame, Topic, and Frame ID.*

---

## Real-time Odometry & Pose Acquisition Steps

Here is a typical approach to get the camera and RTAB-Map working together, outputting the camera's pose in real-time.

1. **Confirm dependency packages first**
   - `rtabmap_ros`, `realsense2_camera` have been compiled without errors.
   - `realsense2_camera` can start alone without a bunch of errors.
2. **Use an integrated Launch file to start camera + RTAB-Map**
   - For example, the existing `realsense_d435i_stereo.launch.py` in `rtabmap_examples`:

     ```
     cd ~/ros2_ws
     source install/setup.bash
     ros2 launch rtabmap_examples realsense_d435i_stereo.launch.py
     ```

   - After launching, it will automatically run both the camera and RTAB-Map.
3. **Check Odom**
   - In another terminal:

     ```
     ros2 topic echo /odom
     ```

   - What you see is the current pose data.
4. **RViz2 Visualization**
   - Type `rviz2`, add relevant Displays, and you can more intuitively see your trajectory as you move around.
5. **IMU Filtering or Calibration**
   - If your camera has an IMU and you want more stable orientation, you can filter the IMU data using `imu_filter_madgwick` or similar methods.

---

## IMU Data & Calibration

1. **Why calibrate?**
   - If the IMU data drifts a lot, the pose will have significant errors over time.
   - Calibration can reduce drift, making Roll/Pitch/Yaw more reliable.
2. **First, check the IMU data quality**
   - Start the camera:

     ```
     ros2 launch realsense2_camera rs_launch.py
     ```

   - Then:

```
ros2 topic echo /camera/imu
```

- See if it shakes excessively or the values are particularly unreasonable.

3. **imu_filter_madgwick**
   - Usually, a filter like this is configured in `realsense_d435i_stereo.launch.py` to automatically process gyroscope and accelerometer data.
   - Main parameters include:

```
use_mag: false
world_frame: enu  # East-North-Up
```

   - When starting, place the camera steadily for a few seconds, waiting for the filter to acquire an initial "reference attitude".
4. **Manual IMU Calibration (Optional)**
   - If you want very high precision, you can use the `/tools` tools in the `librealsense` repository for calibration.
   - After calibration, you can write the results into the camera hardware, so you don't have to deal with it as much next time.

---

## 3D Reconstruction Ideas & Follow-up Points

If you plan to do 3D reconstruction later, especially combined with ultrasound images, here is a possible approach:

1. **Prerequisites**
   - Ultrasound images (DICOM or other formats) + timestamps.
   - Synchronously acquired camera poses (or robot poses).
   - Basic denoising or enhancement has been performed on the ultrasound frames.
2. **Image Preprocessing**
   - Ultrasound often has much noise, such as Speckle, refraction, shadows, etc., requiring some filtering or enhancement methods.
   - Image segmentation (automatic, semi-automatic, or manual) may also be needed to outline the tissue/region you care about.
3. **Synchronize Timestamps**
   - Generally, the acquisition time of the ultrasound image needs to be matched with the pose time of the camera (or sensor).
   - Each image frame has a corresponding pose so it can be correctly placed in 3D space.
4. **3D Point Cloud or Voxel Fusion**
   - First, map each 2D ultrasound frame to a three-dimensional coordinate, then multiply by the corresponding pose transformation matrix (Translation + Rotation).
   - Finally, merge all frames into a complete point cloud/voxel grid.
5. **Output**

- Usually, the fused 3D data is exported in point cloud formats like `.pcd`, `.ply`, or surface reconstruction (Meshing) is continued.

---

## Yaw Deviation & Coordinate System Explanation

During actual testing, you might see a deviation of about 90° in the `Yaw` value at startup. For example:

```
[INFO] [1738926558.893188446] [position_logger]: Time: 2025-02-07 11:09:18.703, Position: x=-0.0 cm, y=-0.0 cm, z=-0.0 cm, Roll: -0.0°, Pitch: 1.3°, Yaw: -92.1°
```

- If you want the initial `Yaw=0°`, you can add an offset correction in the code, using the starting pose as the baseline.
- You can also compare the actual rotation amount using `tf2_ros tf2_echo map camera_link` to see if it matches your expectation.

---

## About TF Transforms & Coordinate Systems

1. **View TF**

   ```
   ros2 run tf2_ros tf2_echo map camera_link
   ```

   Will output a 4×4 matrix, as well as translation, quaternion, RPY, and other information, for example:

   ```
   Translation: [-0.009, -0.024, -0.005]
   Rotation (Quaternion): [0.009, 0.008, -0.741, 0.671]
   Rotation (RPY, degree): [-0.007, 1.383, -95.640]
   ```

   - This indicates the approximate pose relationship between the camera coordinate system and `map`.
2. **TF Transform Tree**
   - Generally, you will see a chain like `map -> odom -> camera_link`.
   - `map`: Global reference (world coordinates)
   - `odom`: Odometry coordinates
   - `camera_link`: Camera body coordinates, attached under odom
3. **Axis Directions**
   - Realsense and ROS typically define: `X` forward/backward, `Y` left/right, `Z` up/down (but sometimes there are slight differences).
   - It's best to refer to the corresponding URDF or official documentation. The best way is to physically move the camera and see if the data for the corresponding axis increases or decreases, which makes it clear.

## Configuration & Parameter Supplements

- **Decimation filter**
  - In Realsense parameters, if you want to reduce the resolution of the depth image to reduce noise, you can enable:

    ```
    {'name': 'decimation_filter.enable', 'default': 'true', 'description': 'enable_decimation_filter'},
    ```

  - This makes the depth data more "sparse" but also cleaner.
- **Reconnect Timeout**
  - Can be set to around 5 seconds. If the USB temporarily malfunctions, it won't cause the node to crash directly.
- **wait_imu_to_init**
  - Setting it to `true` allows the system to wait for the IMU to be ready before starting large movements; otherwise, shaking at the beginning can easily cause initialization offset.
- **Mode Comparison**
  - `stereo`: Left and right infrared views are used for depth calculation. Fast speed and good accuracy, a commonly used mode.
  - `RGBD`: Color plus depth, more precise but sometimes requires more processing.
  - `Infra`: Uses the infrared stream for ranging, fast speed but potentially slightly lower accuracy, depending on your needs.

> *Final reminder, remember to do plenty of testing. You can first run repeatedly in a small scene of known dimensions, see how much the odometry differs from actual measurements, and then slowly adjust the parameters.*