ROS Perception in 5 Days

Chapter 6: People Perception Part 3 - People Tracking

- Summary -

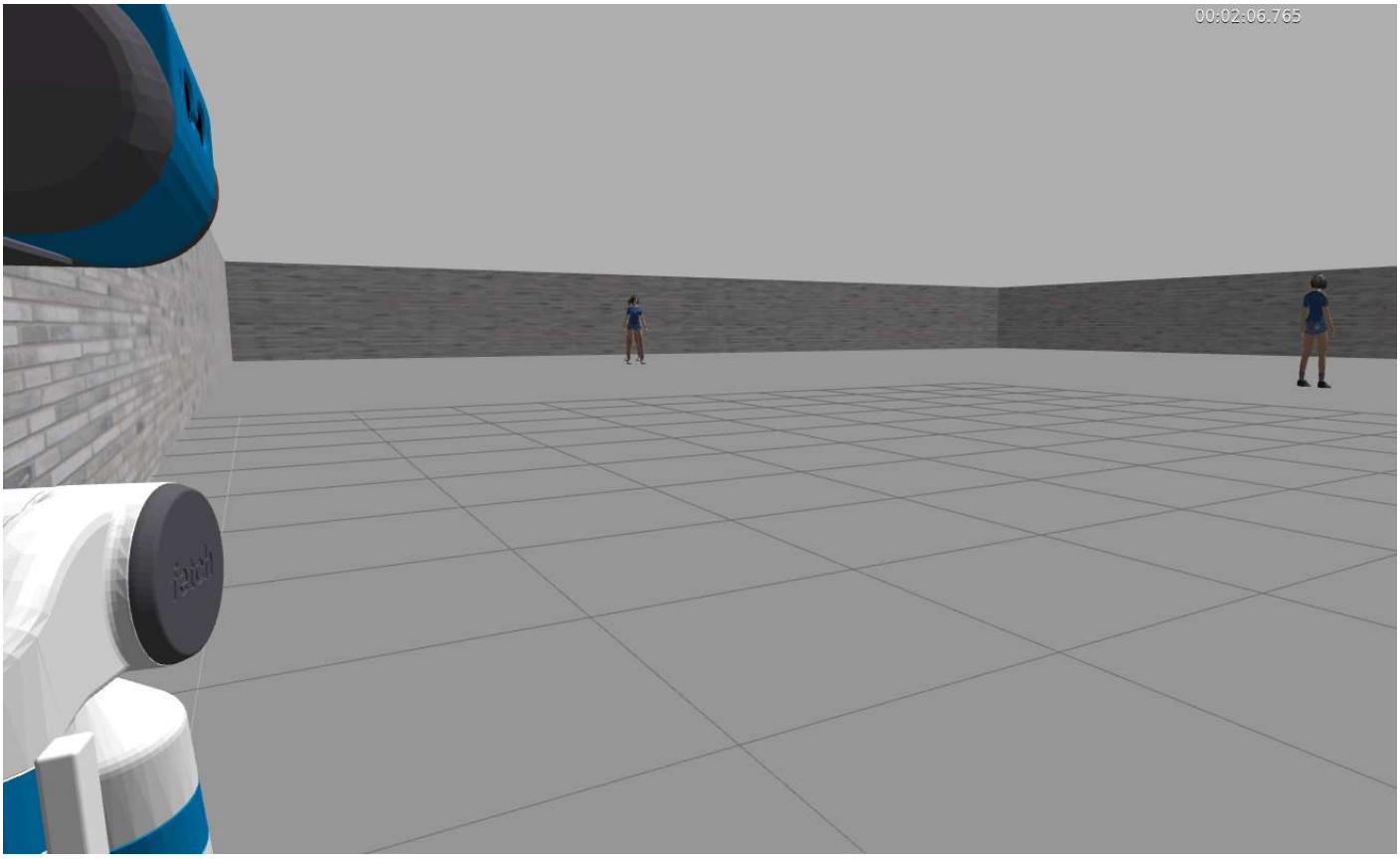
Estimated time to completion: **3 hours**

In this chapter, you will learn how to track multiple persons. Specifically, you will learn about:

- How to detect a person with lasers
- How to detect based on an upper-body detector
- How to combine everything to track three persons moving in an open space.

- End of Summary -

In this final unit about HRI, you will learn how to make a robot follow people around and detect when they approach so that you can avoid them or stop. This skill is essential for robots in public spaces, like museums, malls, etc. It's also important in HRI because a robot has to follow a person, like its instructor, and go wherever needed.



6.1 ROS Package for Tracking People

There are various ways to track a person:

- **Detecting legs**: This is the most basic way to detect people. You look for laser patterns that have a U shape. As you might guess, this gives a lot of false positives, especially when there are chairs around because the chair legs are easily confused with a person's legs.
- **Detecting the upper body**: This method detects people looking for patterns that match upper body shapes. This is more robust than leg detection.
- **Detecting pedestrians**: This is a merging of some data from ground_hog, visual_odometry, and upper_body_detector.

In the end, none of them are as strong as all of them combined. So the best algorithms are the ones that combine all of the data from these detecting systems and many others and process them to have coherent detection.

You will learn how to launch a system that combines all of those systems in a multiple pedestrian simulation. Below see the source of the code that you will use:

- **PedSim_ROS**: It's a pedestrian simulator that generates simulated pedestrians that follow paths, flock, and behave as a crowd would in a busy airport, train station, mall, etc. Go to GIT for the original code.
- **Spencer People Tracking**: This package builds on previous people tracking systems, and it's the perfect candidate for tracking high numbers of people at the same time. It's also optimized for CUDA using HOG, so that the performance will be greatly improved in GPU systems such as **Jetson Nano**. Go to GIT (https://github.com/spencer-project/spencer_people_tracking) for the original code.

For this course, a combined Git was created for convenience and solved some issues in **Noetic** and **NON CUDA systems**: GIT TheConstruct pedestrian detector and sim (https://bitbucket.org/theconstructcore/pedestrian_tc/src/spencer_people_tracking/).

Create a new package named **my_people_tracker_pkg**:

▶ Execute in WebShell #1

In [ ]:

```
roscd;cd ../src
catkin_create_pkg my_people_tracker_pkg geometry_msgs rospy
roscd my_people_tracker_pkg
mkdir launch
touch launch/start_spencer_people_tracker.launch
```

start_spencer_people_tracker.launch

In [ ]:

```
<launch>


    <!-- Which RGB-D detector to use? Default setting (only upper-body) gives
    <arg name="use_pcl_detector" default="false"/>
    <arg name="use_hog_detector" default="false"/>
    <arg name="use_upper_body_detector" default="true" unless="$(arg use_pcl_d
    <arg name="use_upper_body_detector" default="false"    if="$(arg use_pcl_d

    <!-- TF frame IDs used by some detectors, and tracking components -->
    <arg name="base_footprint_frame_id" default="base_link"/>  <!-- name of th
    <arg name="world_frame_id" default="map"/>  <!-- this is the fixed trackir

    <arg name="visualization" default="true"/>
    <arg name="rviz_config_file" default="$(find spencer_people_tracking_launc



    <!-- Sensors Topics-->
    <arg name="front_laser_topic"  default="/base_scan"/>
    <arg name="point_cloud_topic" default="/head_camera/depth_registered/point
    <arg name="camera_info_topic" default="/head_camera/rgb/camera_info"/>
    <arg name="camera_namespace" default="/head_camera"/>

    <arg name="depth_image" default="/depth_registered/image_raw" />
    <arg name="rgb_image" default="/rgb/image_raw" />
    <arg name="camera_info_depth" default="/depth_registered/camera_info" />

    <!-- Optionally load a map to suppress false positives -->
    <node name="map_server" pkg="map_server" type="map_server" args="$(find fe


    <!-- Detectors -->
    <include file="$(find spencer_people_tracking_launch)/launch/detectors/frc
        <arg name="point_cloud_topic" value="$(arg point_cloud_topic)"/>
        <arg name="camera_info_topic" value="$(arg camera_info_topic)"/>
        <arg name="camera_namespace" value="$(arg camera_namespace)"/>
        <arg name="upper_body" value="$(arg use_upper_body_detector)"/>
        <arg name="pcl_detector" value="$(arg use_pcl_detector)"/>
        <arg name="hog" value="$(arg use_hog_detector)"/>
        <arg name="base_footprint_frame_id" value="$(arg base_footprint_frame_
        <arg name="depth_image" value="$(arg depth_image)" />
        <arg name="rgb_image" value="$(arg rgb_image)" />
```

```
        <arg name="camera_info_depth" value="$(arg camera_info_depth)" />
    </include>

    <include file="$(find spencer_people_tracking_launch)/launch/detectors/las
        <arg name="rear" value="false"/>
        <arg name="front_laser_topic" value="$(arg front_laser_topic)"/>

    </include>


    <!-- People tracking -->
    <include file="$(find spencer_people_tracking_launch)/launch/tracking/frei
        <arg name="rgbd" value="true"/>
        <arg name="laser_low_confidence_detections" value="true"/>
        <arg name="base_footprint_frame_id" value="$(arg base_footprint_frame_
        <arg name="world_frame_id" value="$(arg world_frame_id)"/>
    </include>

    <!-- Group tracking -->
    <include file="$(find spencer_people_tracking_launch)/launch/tracking/grou

    <!-- VISUALIZATION -->
    <node name="rviz" pkg="rviz" type="rviz" args="-d $(arg rviz_config_file)"

</launch>
```

Although this example is with **three persons**, this system can come with hundreds of them. We are just limiting the number for performance reasons.

Let's comment on the main elements that you will need to apply to any robot:

In [ ]:
```
<arg name="use_pcl_detector" default="false"/>
<arg name="use_hog_detector" default="false"/>
<arg name="use_upper_body_detector" default="true" unless="$(arg use_pcl_detec
<arg name="use_upper_body_detector" default="false"    if="$(arg use_pcl_detec
```

Here we set the detection systems we will use. In this basic example, we will only use **leg detection and upper-body detectors**. Note that the **HOG_Detector** is perfect for CUDA-enabled systems.

In [ ]:
```
<arg name="base_footprint_frame_id" default="base_link"/>  <!-- name of the ro
<arg name="world_frame_id" default="map"/>  <!-- this is the fixed tracking fr
```

Here we set the **base_frame** normally, the **base_footprint** that has a lot of robots that are **navigation enabled**. In this case, it's **base_link**. As for the **world_frame**, we set a **map** here because this system uses a previously generated navigation map to improve the detections.

In this case, the Fetch robot is started with navigation and a map of the environment. You will see that when opening RViz.

In [ ]:
```
<!-- Sensors Topics-->
<arg name="front_laser_topic"  default="/base_scan"/>
<arg name="point_cloud_topic" default="/head_camera/depth_registered/points"/>
<arg name="camera_info_topic" default="/head_camera/rgb/camera_info"/>
<arg name="camera_namespace" default="/head_camera"/>

<arg name="depth_image" default="/depth_registered/image_raw" />
<arg name="rgb_image" default="/rgb/image_raw" />
<arg name="camera_info_depth" default="/depth_registered/camera_info" />
```

This is probably the most important section, where we set the **topics** for our sensors. You need a robot that has an RGB-D sensor, for example, **Xtion** or **Kinect**. You will also need at least **ONE front laser**.

In [ ]:
```
<include file="$(find spencer_people_tracking_launch)/launch/detectors/front
    <arg name="point_cloud_topic" value="$(arg point_cloud_topic)"/>
    <arg name="camera_info_topic" value="$(arg camera_info_topic)"/>
    <arg name="camera_namespace" value="$(arg camera_namespace)"/>
    <arg name="upper_body" value="$(arg use_upper_body_detector)"/>
    <arg name="pcl_detector" value="$(arg use_pcl_detector)"/>
    <arg name="hog" value="$(arg use_hog_detector)"/>
    <arg name="base_footprint_frame_id" value="$(arg base_footprint_frame_id)"
    <arg name="depth_image" value="$(arg depth_image)" />
    <arg name="rgb_image" value="$(arg rgb_image)" />
    <arg name="camera_info_depth" value="$(arg camera_info_depth)" />
</include>
```

This is the first detector, the **upper-body detector**. It looks for patterns that look like an upper-body of a human being. This works quite well at close range and in good visibility.

In [ ]:
```
<include file="$(find spencer_people_tracking_launch)/launch/detectors/laser_d
    <arg name="rear" value="false"/>
    <arg name="front_laser_topic" value="$(arg front_laser_topic)"/>

</include>
```

And this is the **laser_detector**. It's an old method for detecting a person, detecting two people closed by cylinder patterns with a 2D laser scan. It's very robust, especially when cameras don't work well, such as poor lighting conditions.

This **Spencer People Tracking System** is really rich, and it could be its own course. For our purposes, we narrowly define its scope for this **basic perception course**.

If you are interested in learning more about the Spencer People Tracking Systems and its features, go to documentation (https://github.com/spencer-project/spencer_people_tracking/blob/master/README.md).

6.2 Launch the People Tracker

Launch the people tracker, and let's see how it performs:

▶ Execute in WebShell #1

In [ ]:
```
roslaunch my_people_tracker_pkg start_spencer_people_tracker.launch
```

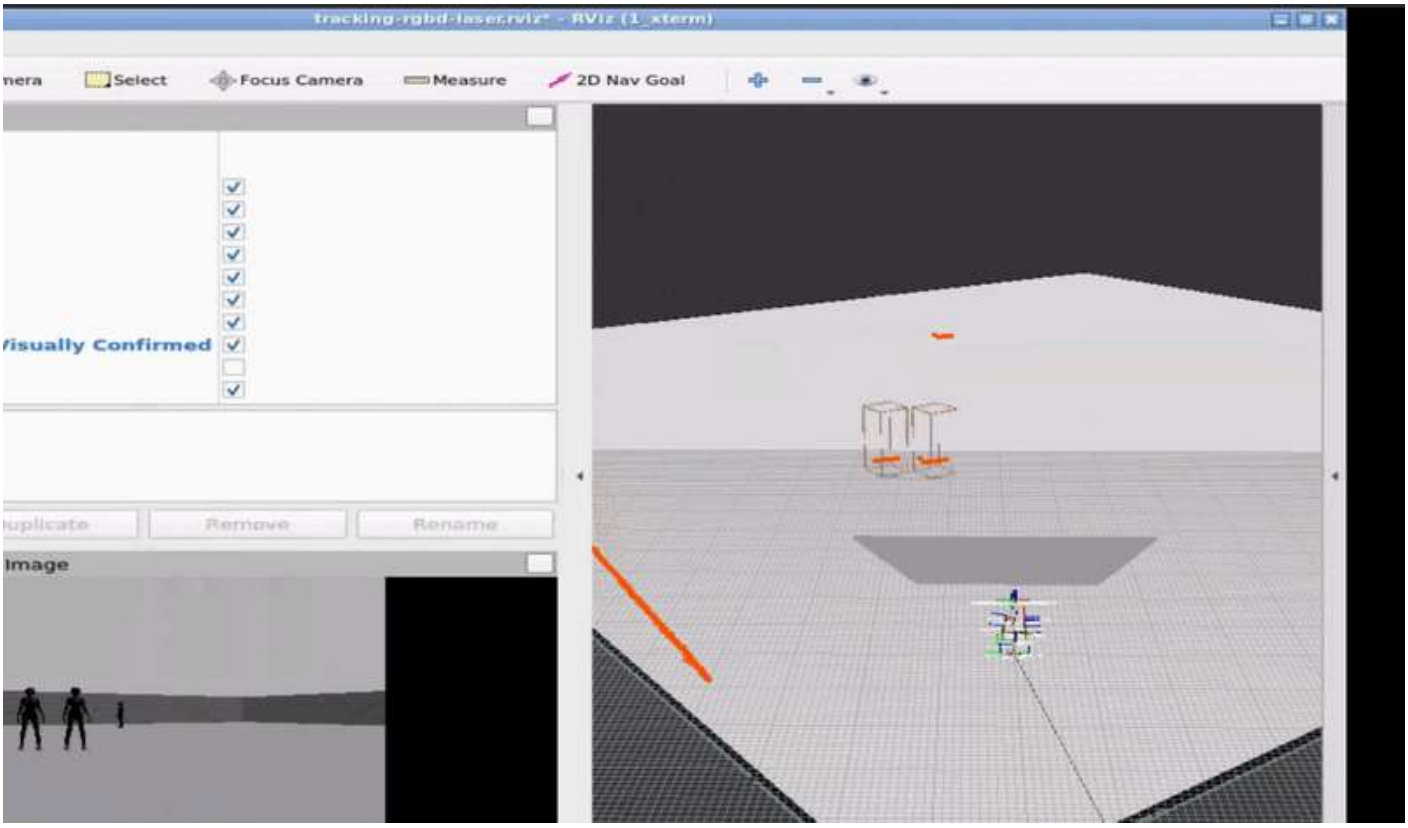
When executing this program, RViz will appear automatically set for people tracking in the **Graphical Interface**. Click on the Graphical Interface icon:

You can move the **Fetch robot** using basic keyboard commands to see how it tracks people.

▶ Execute in WebShell #2

In [ ]:
```
rosrun teleop_twist_keyboard teleop_twist_keyboard.py
```

You should then see something similar to the image below:



Congratulations! You have finished the main topics for this course. You can now proceed to the final project to apply all that you have learned.