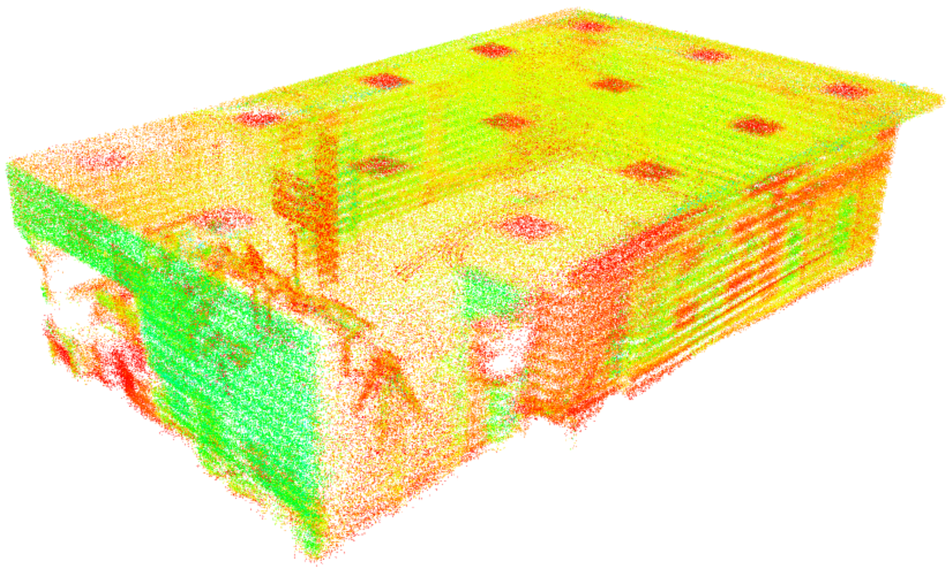
The PointCloud Tailor Guide



Ismael Bouzarhoun El Haddad
Gerard Carreras Mallol
Tobias Christopher Nick
Josep Cobos Brunet
Aleix Gaya Sarri

Adrián López Pardina
Saida Li Sillo Pérez
Aleix Teruel Menéndez
Albert Tomàs Ruiz
Oriol Torres Pla

Barcelona, Autumn 2024

Index

1. Linux

The 3 best ways to work with Linux are explained below, in our specific case, with Ubuntu 20.04 (Focal Fossa). In our personal opinion, the optimal situation would be to be able to work with a computer that has Linux installed, or a partition where it is installed, or to install it on a bootable USB that we can take with us anywhere, since this way we avoid the limitations of a virtual machine or WSL. It is true that this installation process could be a bit complex, so working with WSL is simple and enough to carry out this entire project easily.

Windows Subsystem Linux

For complete and detailed information, check the official website at the following link.
In the following link you will find a detailed tutorial regarding the use and installation of WSL.
The advantage of WSL compared to a virtual machine is that in this case Linux is not isolated, but is integrated with Windows, but a priori we can only use it through the command line, not with a graphical interface, which is not a problem for our project.

Proceed with the installation.

1. First, when the computer is turning on, press the function key that allows you to access the BIOS, this varies depending on the manufacturer. And make sure that the *Virtualization Technology* section is enabled.
2. Open the Windows terminal with *Windows+R* or *Powershell*, write the following command and accept all the permissions that are required:
   ```
   wsl --install -d Ubuntu-20.04
   ```
3. Now you can use PowerShell or Windows terminal like in Ubuntu. It will be necessary to update:
   ```
   sudo apt update
   sudo apt-get upgrade
   ```
4. It is better to restart the computer after this procedure.
5. Now, simply typing the `wsl` command you will enter the terminal of the Linux version that you have as default, to make sure it is the one you want, write `wsl -d Ubuntu-20.04` and it will be ready to work.
6. In case you are not sure which version you are in, write the following command: `hostnamectl`.

Virtual Machine

Install any virtual machine, the most recognized is *Oracle VirtualBox*.
For complete and detailed information check the following link.
In the following link there is a detailed tutorial regarding the use of *Oracle VirtualBox*.

Proceed with the installation.

1. Download the Ubuntu 20.04.6 LTS operating system ISO desktop image from the official website.
2. Download *Oracle VirtualBox* from the official website, run it and follow the steps.
3. Start the program and click *New* to create a new virtual machine that you will call Ubuntu.

4. Choose a RAM memory size taking into account that the minimum recommended for Ubuntu is 2 GB.
5. Click *Next* and *Create a virtual hard drive*, select the *VDI* hard drive file type and the storage on the physical hard drive *Dynamically Reserved*.
6. Assign a disk size of at least 30 GB.
7. Now, you have the virtual machine created. Select it, go to *Configuration*, and in *Storage* click where it says *Empty* within *Controller: IDE* and select the disk file that you have previously installed as an Ubuntu ISO image.
8. As the last step of preparing the environment, you only need to select the virtual machine and click *Start*.
9. Finally, you will start the Ubuntu installer. Follow the steps to install it and you can now use the virtual machine normally.

Bootable USB

For complete and detailed information check the following link.
To install Linux on a USB it will be necessary to have preferably a 3.1 USB of at least 32 or 64 GB for better usability and fluidity. You will also need another USB of about 8 GB to use as an installer.

Proceed with the installation.

1. Download Ubuntu 20.04.6 LTS OS Desktop ISO Image from the official website.
2. Install a bootable USB drive creation program such as Rufus or BalenaEtcher.
3. Run the chosen program, select the file that you just downloaded and select the USB drive that you want to use as the installer. Click on *Start* and let it work.
4. Once the process is finished, restart the computer and enter the BIOS using the function key predetermined by the manufacturer (F2, F10, F12...), and make sure that in the *Boot* or *Security* section the *Secure Boot* option is found. disabled, and preferably select *Legacy* or *CSM* boot mode, although you can also use *UEFI*.
5. Then, sort the boot priority by putting booting from a USB drive first, and save the changes.
6. The computer will automatically restart and enter the Ubuntu installer. Connect the USB on which you want to install Linux and choose the option *Try or install Ubuntu* and then *Install Ubuntu*.
7. Select the language and other options until you reach *Installation type* where you will click on *More options*.
8. On the screen that appears you distinguish your USB both by the MB it contains and by the name, which should be something like */dev/sdb* and delete the partitions using the minus (-) key at the bottom.
9. Now click on the plus key (+) to create a 500 MB EFI type primary partition.
10. Click on the plus key (+) again and create a logical EXT4 partition with the rest of the capacity and mount point in the root partition (/).
11. In the drop-down menu at the bottom choose your USB where to install the bootloader, and click on *Install now*.
12. Once the installation is complete, remove the installation media and the computer will restart, but do not disconnect the USB, but rather press the function key that allows you to enter the boot menu of your computer and choose the one that says Ubuntu or USB. Now you will be ready to use Ubuntu through your USB.

Before disconnecting the USB for the first time, it is recommended to follow a few steps to set up the operating system.

1. Open a terminal window and update using the following commands:
   ```
   sudo apt update
   sudo apt-get upgrade
   ```
2. Once the update is finished, it is also recommended to update the drivers of your computer's controllers using the following commands:
   ```
   ubuntu-drivers devices
   sudo ubuntu-drivers autoinstall
   sudo apt update
   ```

Finally, every time you want to start from our USB you will have to perform step 12 and thus run the operating system installed on the USB.

2. ROS 1

ROS 1 (Robot Operating System 1) is an open source software widely used in robotics. Provides a collection of tools, libraries, and conventions that enable the development and deployment of complex robotics applications. Although called an "operating system," ROS 1 actually works as a middleware layer that facilitates communication between different components of a robot, such as sensors, actuators, control systems, and planning algorithms.

ROS 1 Installation

To install it, it is recommended to use a Linux distribution such as Ubuntu, more specifically Ubuntu 20.04 (Focal Fossa). For complete and detailed information check the following link.

Proceed with the installation.

1. You have to make sure that you have the *Restricted*, *Universe* and *Multiverse* repositories enabled. You can check it by running the *Ubuntu Software* application with the following command (if you have the Ubuntu interface):
   ```
   sudo software-properties-gtk
   ```
   It can also be verified in the terminal with the following command (if you do not have the Ubuntu user interface):
   ```
   grep ^deb /etc/apt/sources.list
   ```
   And to enable any of them from here (if not already enabled), for example *Restricted*:
   ```
   sudo add-apt-repository restricted
   ```
2. Configure the list of sources to accept ROS software:
   ```
   sudo sh -c 'echo "deb http://packages.ros.org/ros/ubuntu $(lsb_release -sc) main" > /etc/apt/sources.list.d/ros-latest.list'
   ```
3. Configure the keys with the following commands:
   ```
   sudo apt install curl
   curl -s https://raw.githubusercontent.com/ros/rosdistro/master/ros.asc | sudo apt-key add -
   ```
4. Update the system and install the full desktop version:
   ```
   sudo apt-get update
   sudo apt install ros-noetic-desktop-full
   ```
5. Every time you start the terminal you have to configure the development environment, and in order to forget about this, execute the following command that writes a line in the bash configuration and thus launch it automatically and load the configurations:
   ```
   echo source /opt/ros/noetic/setup.bash >> ~/.bashrc
   source ~/.bashrc
   ```
6. Install some dependencies to develop ROS packages:
   ```
   sudo apt install python3-rosdep python3-rosinstall python3-rosinstall-generator python3-wstool build-essential
   ```
7. Install and initialize *rosdep*, which allows you to easily install system dependencies for the source code you want to compile and is necessary for some main ROS components:
   ```
   sudo apt install python3-rosdep
   sudo rosdep init
   rosdep update
   ```

ROS 1 Graphic Tools

With this you will have ROS 1 installed. Now you can install some graphical interfaces like:

1. The best option is *Rviz*, for 3D visualization of robots and sensor data (LiDAR...):
   sudo apt-get install ros-noetic-rviz
   Initialize it with the command: rviz
2. *RQT* is also a good choice for real-time monitoring and debugging tasks:
   sudo apt-get install ros-noetic-rqt ros-noetic-rqt-common-plugins
   Initialize it with the command: rqt_graph
3. *Gazebo* is good for a physical simulation environment:
   sudo apt-get install ros-noetic-gazebo-ros-pkgs
   Initialize it with the command: gazebo

ROS 1 Elements

Once you have installed the ROS1 software, we are going to detail some of the most important elements of which it is made up and that it is necessary to know before starting to work:

- Node: Independent processing unit in ROS. It runs a specific task (as if it were an independent program).
- Topic: Communication channel that allows the exchange of messages between nodes. A node can publish (publishers) on a topic, while other nodes can subscribe (subscribers) to receive those messages.
- Service: Unlike topics, these allow a node to request a task directly from another node and receive a response. They are useful for performing specific actions.
- Master: Core that manages communication between nodes. It keeps track of which nodes exist, which topics they publish and subscribe to, and which services are available. It acts as a system coordinator and without it the nodes cannot communicate.
- Message: Data structure that contains information that nodes exchange with each other (*std_msgs*, *sensor_msgs*, etc). They allow communication between nodes through topics, services or actions.
- Bags: Tool to record and playback ROS messages. They store messages in real time that can be replayed later for analysis or simulations.
- Launch files: Files that allow launching multiple nodes with specific parameters. They simplify the management of complex configurations and the launch of entire systems.
- Packages: Basic unit of organization in ROS. They contain nodes, scripts, configuration files, launch files, and other dependencies necessary for a robotic application.
- Catkin: ROS build system, based on CMake that allows you to compile packages and manage dependencies between them, facilitating the process by ensuring that all packages and dependencies are built correctly.

ROS 1 Commands

To learn more information about the nodes, topics or services involved, we can execute the following commands:

- To see the list of active nodes: rosnode list

- To see information about a node: rosnode info */node_name*
- To end the execution of a node: rosnode kill */node_name*
- To see the list involved topics: rostopic list
- To see information about a topic: rostopic info */topic_name*
- To see what is published on a topic in real time: rostopic echo */topic_name*
- To see the available services: rosservice list
- To see information about a service: rosservice info */servicie_name*
- To run an available service: rosservice call */service_name*
- To see the parameter server parameter list: rosparam list
- To get the value of a parameter: rosparam get */param_name*
- To set the value of a parameter: rosparam set */param_name value*
- To see information about a bag: rosbag info */bag_name*

To learn how to run a program in ROS we have the following commands:

- To run the ROS Master: roscore
- Execute only one node: rosrun *pkg_name node_name*
- Execute a launcher file: roslaunch *pkg_name lauch_file*.launch

Finally, in the following link you will find a series of tutorials from the official website to familiarize yourself with the ROS environment.

We also suggest you watch the video tutorials in the playlist at the following link.

3. Project Development

The goal of this project is to explore an extension of the HP SitePrint robot features by adding the ability to check the accuracy of an already built building against specifications by scanning the construction elements using a LiDAR-enabled sensor.

To this end, we developed a program that seeks the correct operation of algorithms capable of efficiently aligning consecutive point clouds detected by the LiDAR sensor integrated into the robot based on edges and other geometries of interest, in real-time and with relatively modest hardware, typical of embedded platforms with low memory and processor.

Creating the Workspace

Proceed to create our own workspace. We will do it using the *Catkin* tool.

1. We start by creating our workspace which we will call *catkin_ws* in the root directory (it can be anywhere else), and within it a folder called *src* where we will save our packages, do it with the following command:
```
mkdir ~/catkin_ws/src/
```
2. Before performing the next step, it is important to make sure that we are at the root of our directory (not inside *src*), and then compile it. We will see that the *build* and *devel* folders are created, as well as other files necessary for ROS to work:
```
cd ~/catkin_ws/
catkin_make
```
3. Now, configure the environment to be able to use our workspace in ROS, and load the configuration for the changes to take effect. To do this, write the following commands that modify the *~/.bashrc* file so that this is done automatically every time we start a new terminal:
```
echo source ~/catkin_ws/devel/setup.bash >> ~/.bashrc
source ~/.bashrc
```

Creating the Package

With this we would have our own workspace ready to work on. But we can go further and create a package with the final point cloud alignment program.

1. Go to the *src* directory of our workspace and create a package that contains the ros*cpp* (C++), *std_msgs*, *tf*, *pcl_conversions* and *pcl_ros* dependencies that will be necessary for the program we are going to create:
```
cd ~/catkin_ws/src/
catkin_create_pkg point_cloud_tailor roscpp std_msgs std_srvs sensor_msgs nav_msgs geometry_msgs pcl_conversions pcl_ros
```
2. A folder with the name *point_cloud_tailor* will have been created, enter it and create another folder called *src* (if it has not been done automatically):
```
cd ~/catkin_ws/src/point_cloud_tailor
mkdir src
```

Creating the Program

Now, it is time to explain how to create a program, in this case we will only detail the one that contains the main node and that solves most of the requested needs, since the others are carried out in the same way:

1. Create the *alignment.cpp* file in the *src* directory of our package that contains a ROS node that will publish the accumulation of the aligned clouds of all the consecutive point clouds detected by de LiDAR:
   ```
   cd ~/catkin_ws/src/point_cloud_tailor/src
   touch alignment.cpp
   ```
2. Moving one directory back, modify *CMakeLists.txt* so that the program is then compiled and its dependencies updated, using the command:
   ```
   nano CMakeLists.txt
   ```
   Add the executables and the dependencies for the libraries at the end of the file:
   ```
   ## Add the program executable
   add_executable(alignment_node src/alignment.cpp)
   ## Link the ROS necessary libraries
   target_link_libraries(alignment_node ${catkin_LIBRARIES})
   ## Message dependences
   add_dependencies(alignment_node ${${PROJECT_NAME}_EXPORTED_TARGETS}
   ${catkin_EXPORTED_TARGETS})
   ```
3. Return to the root of our workspace and compile:
   ```
   cd ~/catkin_ws/
   catkin_make
   ```

In order to run the program and check the correct operation of the different test versions that we are implementing, we do the following in different terminal windows:

1. Execute the ROS Master:
   ```
   roscore
   ```
2. Execute the program nodes:
   ```
   rosrun point_cloud_tailor alignment_node
   ```
3. Launch the RViz interface to visualize the result:
   ```
   rviz
   ```
   Once inside RViz, check that we have *map* as the value of *Fixed Frame* in the *Global Options*. Then, add a *PointCloud2* type viewer and in the *Topic* select the *aligned_cloud*.
4. Finally, publish the dataset bag into the topic:
   ```
   cd ~/catkin_ws/rosbags/
   rosbag play livox_room.bag
   ```

System Operation

In order to validate the result of the system we use bag files that contain a set of data obtained by the LiDAR sensor. The bag file format is very efficient for both recording and replaying messages in ROS, as it works similar to how ROS nodes subscribe and publish via ROS topics. With these files we can collect the point clouds captured by the LiDAR sensor at a given moment, and then reproduce them to simulate that we are doing it in real time.

If you analyze the system and the program code you will be able to detect that from the input node (in this case in the form of a bag file) the point clouds collected by the LiDAR sensor are published in the alignment node, which tries to align them by using the *ICP* (Iterative Closest Point) algorithm, thus obtaining an aligned cloud that we can show in RViz and that can be saved in *.pcd* format.

Also published in the transformations topic are the transformations carried out in each step of the *ICP* algorithm, which the trajectory node takes advantage of to estimate the path carried out by the robot over time. We know this working method as *SLAM* (Simultaneous Localization And Mapping).

In addition, we have incorporated more functionalities that will be seen later, such as filtering by height to delete the ceiling of the room and the transformation from *.pcd* to *.bmp* format.

Adjustable Parameters

This program has a lot of versatility in order to be able to adapt it to multiple environments. Among the most notable parameters you can modify are:

1. *MAX_HEIGHT*:
   Determines the maximum height at which we are going to filter the point cloud, it is mainly used to delete the ceiling. Now it is initialized by default to 1.7 m to ignore the ceiling of the room, and if you write the argument *show_ceiling:=true* as explained above, the value becomes 3 m. Both values can be adapted to the situations that best suit them.
2. *Leaf_size*:
   Parameter of the *downsamplePointCloud()* function. Determines that the density of the point cloud is reduced, leaving only one representative point in a cube of size equal to the assigned leaf size. Which implies that there will be no points less than the distance that has been assigned to the leaf size. Currently the value is initialized at 0.2 m, which is an appropriate value for interiors of which we have many consecutive point clouds detected.
3. *Radius*:
   Parameter of the *filterRedundantPoints()* function. Determines that any new point that is within this radius of another point already detected will be ignored. Now it is initialized by default to 5 cm.
4. Any parameter integrated into the *ICP* algorithm, modifying them allows you to adapt to each scenario in search of the most optimal alignment possible. Some of them are the maximum number of iterations, the maximum correspondence distance or the convergence threshold.

4. Project Execution

Required Libraries

In order to carry out our project, it is important to have the necessary libraries to work with point clouds, in this case *PCL*:

1. Install these libraries in Ubuntu:
   ```
   sudo apt-get install libpcl-dev
   ```
2. Install the libraries package in ROS:
   ```
   sudo apt-get install ros-noetic-pcl-ros
   ```
3. Verify that it has been installed correctly:
   ```
   rospack find pcl_ros
   ```

In addition, it is necessary to install the *Eigen* library, that is a mathematical library developed to work with geometric transformations and mathematical optimization efficiently:

1. Install the library:
   ```
   sudo apt install libeigen3-dev
   ```
2. Verify that it has been installed correctly:
   ```
   ls /usr/include/eigen3
   ```

Cloning the Project

First of all, you have to clone in your workspace the GitHub repository that contains the final project, to later execute it.

1. Using the terminal, go to the *src* directory of your workspace and access the Github repository: *https://github.com/albert-tomas/PAE-HP*. And clone it into your *Catkin* workspace using the following command:
   ```
   cd ~/catkin_ws/src/
   git clone https://github.com/albert-tomas/PAE-HP.git
   ```
2. Then, you must go to the root of your workspace (*catkin_ws*), making a jump back, and compile again with the command:
   ```
   cd ~/catkin_ws/
   catkin_make
   ```
3. At this time, as always before starting to run ROS in a terminal, it would be necessary to update the ROS environment with your workspace, but if you have previously modified your *~/.bashrc* file adding the configuration of your workspace it will not be necessary, since when opening a new terminal it will do it by default.
   If you have not done, do this:
   ```
   source ~/catkin_ws/devel/setup.bash
   ```

Execution Steps

Now you have everything ready to start working, you just have to follow the steps that we detail below:

1. Running the ROS Master is not necessary if you are going to run the program with a launch file, so in a terminal window, you can easily run the program like this:
   ```
   roslaunch point_cloud_tailor alignment.launch
   ```
   What you are doing is to execute the *alignment.lauch* file found in the *point_cloud_tailor* package. This will run all the nodes by default, and the ceiling will not be shown, also it will launch an RViz window with all parameters initialized.
2. Then, in another terminal window, you have to place yourself in the same directory where the dataset is located, and decompress the bag file.
   Finally, publish the point clouds to the topic using:
   ```
   cd ~/catkin_ws/src/PAE-HP/rosbags/
   rosbag play livox_room.bag
   ```

Additional Features

Additionally, we have added several ROS services that perform different functions that we found useful to validate the correct operation of the algorithm.

1. In another terminal window you can call a service at any time to save the final aligned point cloud result in *.pcd* format in the directory where the program is running:
   ```
   rosservice call /save_pcd
   ```
2. For this other service we have applied *SLAM* in order to show the trajectory made by the robot with the LiDAR sensor:
   ```
   rosservice call /trajectory
   ```
3. If you want to see the complete detection of the room, including the ceiling, add the argument *show_ceiling:=true*, this will change the maximum height value to 3 m:
   ```
   roslaunch point_cloud_tailor alignment.launch show_ceiling:=true
   ```
4. In case you do not want to use or publish on RViz the final aligned cloud, you can add the argument *publish_cloud:=false*, if you do not write anything the program will run by default de visualization:
   ```
   roslaunch point_cloud_tailor alignment.launch publish_cloud:=false
   ```

To end, we have created a simple program *2D_map.cpp*, that you can run whenever you want, capable of reading the final point cloud result in *.pcd* format, which after properly filtering it and collapsing its vertical axis, we obtain a 2D image in *.bmp* format in order to check the quality of the construction maps:

1. If it is not compiled, compile the program using the following command that explains its dependencies:
   ```
   g++ -o 2D_map 2D_map.cpp -I/usr/include/pcl-1.10 -I/usr/include/eigen3 -lpcl_common
   -lpcl_io -lpcl_filters
   ```
2. Then, you can run the program by default (without showing the ceiling of the room) like this:
   ```
   ./2D_map cloud_name.pcd
   ```
3. If you want to show the ceiling or you are working with taller walls you can add a number argument to define the desired height:
   ```
   ./2D_map cloud_name.pcd [MAX_HEIGHT]
   ```

5. <u>Results</u>

In this section we show you as a result the complete image of the accumulated cloud obtained after aligning all the point clouds of the dataset that has been provided to us.

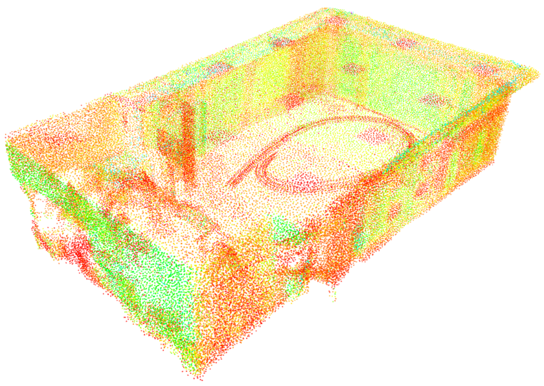


Figure 1. Detected room with ceiling

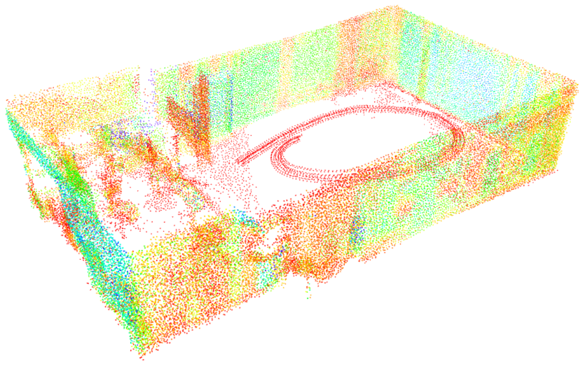


Figure 2. Detected room without ceiling

Note that by default we have deleted the ceiling of the detected room because it is an apparently unnecessary feature in the development of the functions of the HP SitePrint robot, in this way we also achieve better performance in the execution of the algorithm, making it more efficient.

Next, you can see the estimation of the trajectory carried out by the robot at the time of data collection using the LiDAR sensor.

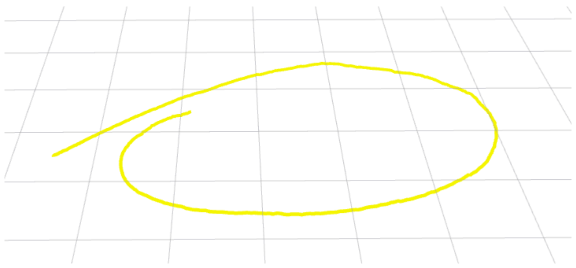


Figure 3. Trajectory estimation

Below, you can see the result of running the *.pcd* to *.bmp* transformation program, you will be able to obtain a 2D map of the detected room.

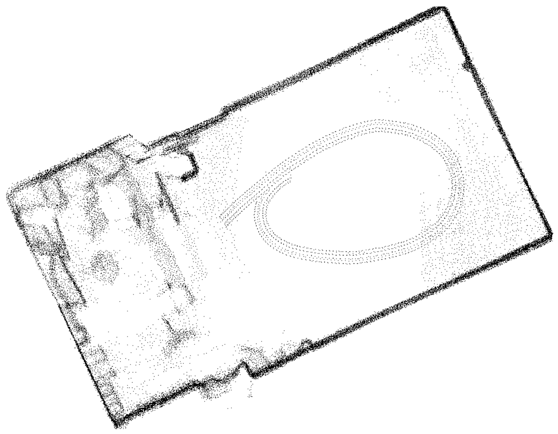


Figure 4. Aligned point cloud 2D map

Here is the flow chart that follows the complete program and how the nodes and topics interact with each other to achieve the final result. It is very helpful in understanding how the system works.

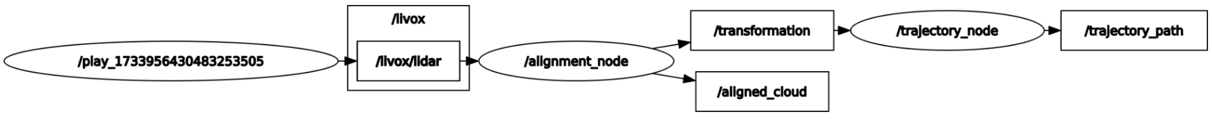


Figure 5. Alignment program flow graph

The dataset contains an amount of 395 point clouds detected with an average of 20,000 points per cloud. And at the end, the final aligned cloud has an amount of 40,751 points.

The execution time of the program running on an Intel Core i7 processor takes 41 seconds, so we are working at 9.63 Hz, compared to the 39.4 seconds and 10 Hz spent by the input dataset, we can proudly say that the algorithm works efficiently in real time.

The *ICP* algorithm takes an average of 93.24 ms and less than 25 iterations to converge properly.

As we also filter the final aligned and accumulated cloud to discard those points that are within a 5 cm radius of another point already detected, we can affirm that approximately (since some points are new due to the movement of the robot) 93.20% of the points align efficiently.

Finally, with the final program we obtain an average *RMSE* (Root Mean Square Error) of 3.54 cm, which, since it is the detection of the interior of a room and is within the range of 2 to 5 cm, is a correct value for this type of detection.