SSRR Rescue Camp Exercises

Date: Monday, October 21, 2013
Location: Linköping, Sweden
Prior knowledge: Linux terminal basic commands, compilation using CMake, basic ROS tools
Goal: get used to already existing registration tools and learn the basic steps to design a custom solution of a given problem.

Installation

Mapping ROS stack

Assuming that you are on Ubuntu 12.04 and have ROS Fuerte or Groovy installed:

```
$ sudo apt-get install ros-fuerte-ethzasl-icp-mapping
```

If you can't find the debian package you can to compile the ROS package from source by following the instruction there: http://wiki.ros.org/ethzasl_icp_mapping

Paraview

Paraview is a free and multiplatform 3D viewer maintained by Kitware (the same company supporting CMake). It is mainly used to display simulation outputs from clusters and allow a high level of interaction with the data. In our case, we will used it to display debug information of on ICP registration. You can install it using apt-get but we suggest to download the binaries from there web site for a more recent version.

Download: http://www.paraview.org/paraview/resources/software.php

Unarchive the repository, open a terminal and move to its root. You should be able to launch Paraview with:

```
$ YOUR_PARAVIEW_PATH/bin/paraview
```

Extra: If you want to launch it from any location, you can append its path to your $PATH environment variable with the *export* command. You can add the export command to your bashrc script to not re-export the path in every new console you open.

Data sets

For the exercises, we will use two types of file format: *.vtk and *.bag. The VTK format will be used for the case where we want to register only two point clouds together where a single VTK file represents one point cloud. The bag format will simulate a stream of laser inputs aggregated in a PointCloud2 type of message.

Download link (87 MB):
http://robotics.ethz.ch/~asl-datasets/SSRR_summerSchool/SSRR_ICP_exercises.tar.gz

Extra: The VTK files used are part of a 3D data sets called *Challenging data sets for point cloud registration algorithms*[1], which covers multiple types of environments with ground truth information in the order of millimeters. More information about the recording setup can be found on the data set web page[2].

Exercise 1: registration of two scans

Your goal is to register pointcloud1.vtk (considered as our reading point cloud) to pointcloud0.vtk (considered as our reference point cloud) using the executable *pmicp*. Here are the proposed steps to do so in a structured way.

Step 1: Observations

Before applying any registration pipeline to point clouds, you should always have a look at the provided data and come up with some conclusion about the strategy that should be used for the ICP solution. Contextual information about the application and the type of environment is essential to understand well the problem. In the case of this exercise, contextual information can be found here:
http://projects.asl.ethz.ch/datasets/doku.php?id=laserregistration:apartment:home

1. Open pointcloud0.vtk in Paraview. Don't forget to click on the *Apply* button to view it.
2. Discuss with your colleagues about the quality of the data.
   Hints:
   a. How many points were scanned?
   b. Where is the center of the point cloud?

---

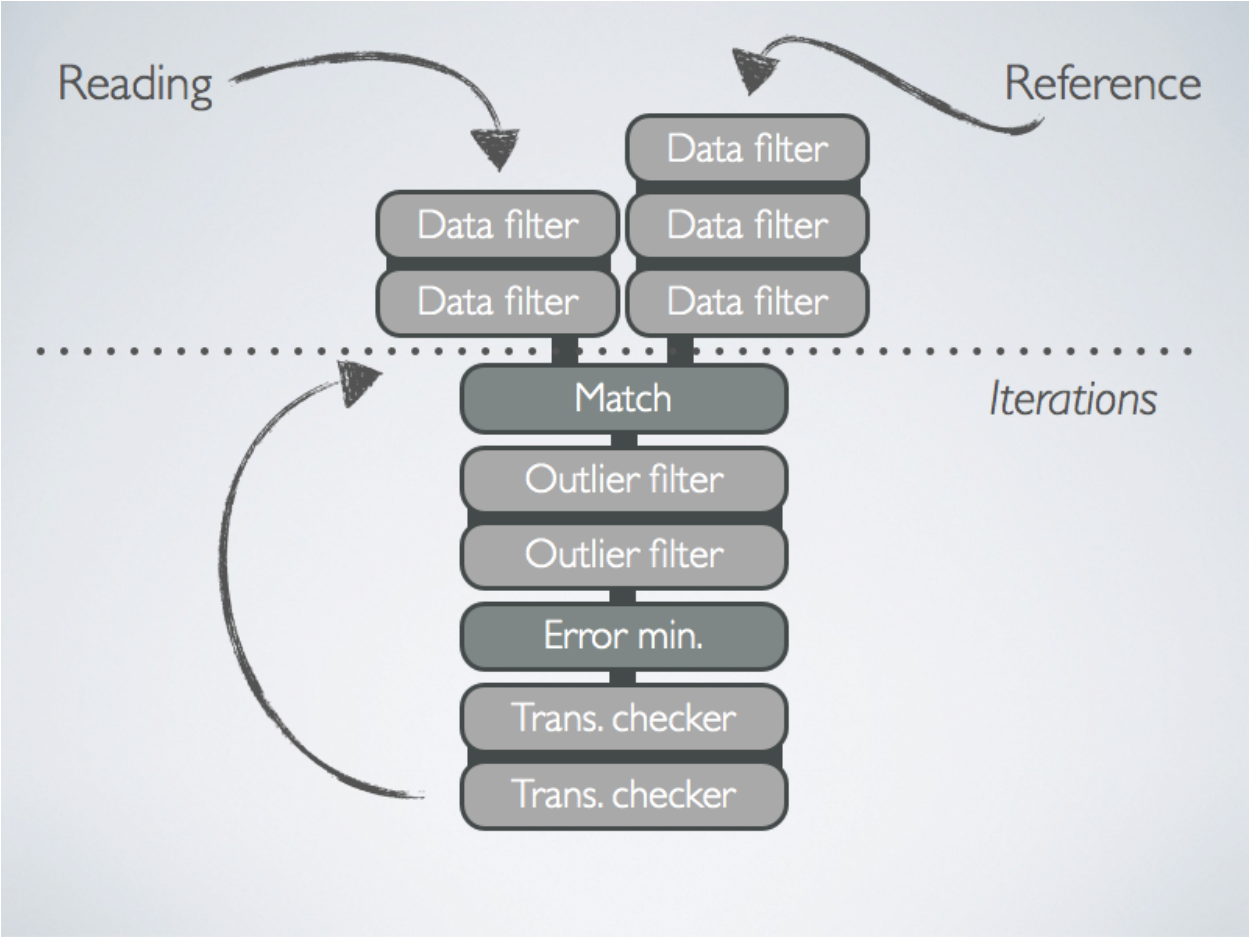
[1] F. Pomerleau, M. Liu, F. Colas, and R. Siegwart, *Challenging data sets for point cloud registration algorithms*, **International Journal of Robotic Research**, vol. 31, no. 14, pp. 1705–1711, Dec. 2012.
[2] http://projects.asl.ethz.ch/datasets/doku.php?id=hardware:tiltinglaser

c. Is the point cloud noisy, with dynamic elements?
d. Is the platform/robot present in the point cloud?
e. Is the density of the points uniform?
f. What about the structure of the environment?
g. Etc.

3. Open the second file pointcloud1.vtk and compare its location to pointcloud0.vtk.
   a. Is the misalignment reasonable?

Step 2: solution design

Based on your observations, you can start to put in place critical components of the registration pipeline. In the repository you downloaded, open the file ex1_icp.yaml with any text editor. A basic pipeline doing practically nothing is already in place to explicit the possible modules. The next figure shows in which order the pipeline is executed.



The documentation of all the modules can be found here:
http://wiki.ros.org/ethzasl_icp_configuration

Add the modules you need based on your prior observations. Keep the following lines unchanged, they will be useful in the further steps.

```
[...]
inspector:
  VTKFileInspector

logger:
  FileLogger
```

Step 3: Test your solution

We will use the executable called pmicp (short for pointmatcher ICP) to test your specific pipeline. You should be able to find it by doing:

```
$ roscd libpointmatcher/bin/
```

The generic way to run ICP is:

./pmicp [OPTION] referenceFileName.{csv or vtk} readingFileName.{csv or vtk}

Warning: some of the modules require write access to the disk to save log files. If you did the installation from the debian package, the package is most probably in /opt/ros/[...] and it is a folder reserved for admin rights. We suggest the following steps to deal with that.

The command rospack find will output the path to a specific package. The following command will show you where is the package libpointmatcher:

```
$ rospack find libpointmatcher
```

You can directly append this command to execute a specific file using `stuff`. For example, the following command will print the help information of pmicp from anywhere:

```
$ `rospack find libpointmatcher`/bin/pmicp
```

Let assume that the path to the repository you downloaded for the exercised is YOUR_PATH. Move to the exercise folder:

```
$ cd YOUR_PATH/exercise1/
```

Then, test your solution:

```
$ `rospack find libpointmatcher`/bin/pmicp  --config ex1_icp.yaml
```

```
../data/pointcloud0.vtk ..data/pointcloud1.vtk
```

Hint: you can access to the module documentation directly in the console using

```
$ `rospack find libpointmatcher`/bin/pmicp  -l
```

Step 4: Debug and validate

The transformation matrix should be close to:

```
0.993175, -0.116489, 0.005872, 0.614863
0.116470, 0.993188, 0.003496, -0.014263
-0.006240, -0.002789, 0.999977, 0.008586
0.000000, 0.000000, 0.000000, 1.000000
```

A visual way to validate your solution is to use the output generated by the VTKFileInspection, which drop a file per iteration on disk. You can open the sequence of files directly in Paraview and press *Play* to see the evolution of the registration solution. Those files should be called point-matcher-output*.vtk.

To have a look at the evolution of different values used to exit the iterative process, you can open the file point-matcher-output-iterationInfo.csv in Matlab, Excel, LibreOffice or even Paraview and plot the evolution.

Once you have confirmed that your solution outputs the right transformation, you can also deactivate console and VTK logging by changing your yaml configuration file with:

```
[...]
inspector:
  NullInspector

logger:
  NullLogger
```

and check your timing performance by adding the command *time* before your command line:

```
$ time ./pmicp  --config YOUR_PATH/exercise1/ex1_icp.yaml
YOUR_PATH/data/pointcloud0.vtk YOUR_PATH/data/pointcloud1.vtk
```

This will include the time of loading the two point clouds as compare to the time reported when using the FileLogger which give the time for the pre-processing and the iteration separately.

Exercise 2: matching a stream of point clouds

This exercise highlights the steps of tuning a solution for a robot moving through an environment and gathering 3D point clouds in order to build a 3D map and localize within this map.

Step 1: Observation

In a terminal, launch a roscore.

```
$ roscore &
```

then, ensure that ROS uses the clock provided by the rosbag by doing:

```
$ rosparam set use_sim_time true
```

Play the rosbag by allowing it to publish its clock:

```
$ rosbag play --clock YOUR_PATH/data/multifloors.bag
```

Hint: the bag is 17 minutes long. As a first overview of the situation, it might be more convenient to use the option **-r 10** (i.e., run the bag 10 times faster).

You can use spacebar to pause the bag. To view the point clouds in rviz:

```
$ rosrun rviz rviz
```

In Global Options, you will have to put the Fixed Frame to /odom and add a point cloud 2 topic to listen to the recorded point clouds. We suggest to add also the /tf topic, which will show where the robot is.

Hints: the topics of the rosbag are:

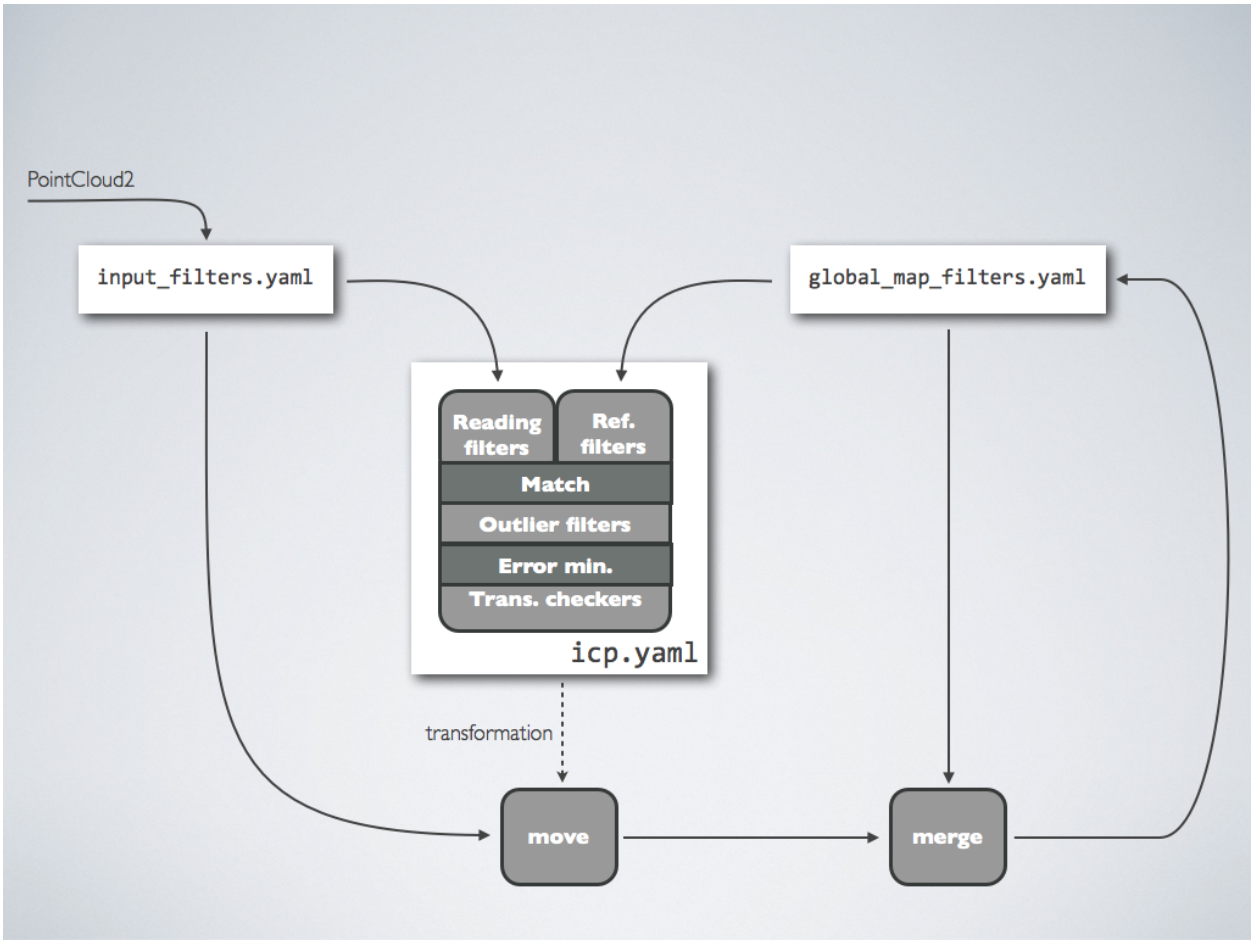
```
/dynamic_point_cloud   103 msgs    : sensor_msgs/PointCloud2
/tf                    154173 msgs : tf/tfMessage
```

Use your expertise from exercise 1 to draw your own conclusions about the quality of the data and identify what kind of ICP solutions would make sense.

Step 2: solution design

The ROS node used for the registration / mapping solution is called *mapper* and it is located in the package named *ethzasl_icp_mapper*. This node is based on libpointmatcher and uses the

same configuration format that in Exercise 1. The main difference is that, instead of having single configuration file for the ICP solution, there are two more for the mapping process. The following graph explains how the data flow through the node.



You will find templates for the needed configuration files in the folder *exercise2*. The files *input_filters.yaml* and *global_map_filters.yaml* are already in a reasonable configuration for the exercise but nothing is stopping you from changing them to cope with your needs.

Open the file ex2_*icp.yaml* and complete the pipeline as in Exercise 1.

Step 3: Test your solution

Restart the bag. Don't forget to put the speed of the playback close to real time with the -r flag..

In folder exercise2, there is also a launch file prepared for the the rosbag used. You can open it to view what parameters are used. To avoid putting absolute path, we had to force relative path to where to launch command is used. This limit you to launch the node from the exercise2 folder:

```
$ cd YOUR_PATH/exercise2/
```

Then, launch directly the file without specifying the package in which it is:

```
$ roslaunch SSRR_mapper.launch
```

In rviz, you will need to add another field of PointCloud2 message type to listen on /point_map topic to view the resulting 3D map being build.