



SAPIENZA
UNIVERSITÀ DI ROMA

FACOLTÀ DI INGEGNERIA DELL'INFORMAZIONE, INFORMATICA E STATISTICA

LOOP CLOSURE

**Slam project on loop-closures recognition based on a real
2D laser scanner dataset**

Professor

Giorgio Grisetti

Student

Edoardo Ghini

Supervisor

Dominik Shlegel

Academic Year 2018/2019

1 Loop Recognition: Big Picture

Loop closure detection is a technique that helps to reduce the escalating uncertainty that is generated from the iterative estimation of the evolution of the system state. In particular it can be applied in autonomous vehicle navigation tasks where the environmental informations gathered from the sensors can be used to build an artificial map. The main objective carried out from loop closure algorithms is to identify portions of the environment that have strong correspondence with others already seen and then decide if the robot is navigating in a location of the environment in which it has already been. In order to develop a loop closure algorithm, that could work in a simulation, it is necessary to have a dataset and other fundamental components:

- A feature extractor able to generate consistent features described by possibly unique descriptors.
- A data association heuristic that allows to find which one of the new features corresponds to the older.
- An estimator of the evolution of the state, that given the data association, iteratively minimise the error of the guess.
Since in this application it deals with laser scan measurements it would be called scan-matcher.
- An algorithm for loop closures recognition that processes the data memorised and tries to close a loop on the artificial map.

2 C++ Implementation Details

2.1 Dataset Parser

I was given a dataset of a real 2D laser scanner featuring an autonomous mobile robot navigating through a corridor.

The DatasetManager class gathers all the scanner ranges in data structures according to their sequence number.

2.2 Feature Extractor

The feature of choice is the line, precisely a 2D segment.

The class LineMatcher, through its heuristic, tries to align subsequent cartesian points that corresponds to the spots where the laser hits the walls.

When the distance or the orientation between two subsequent points exceeds a threshold a segment is built with the former and previously encountered points, then the latter will be the beginning of the following segment.

2.3 Data Associator

In order to recognise features among two consecutive sets of laser ranges that corresponds to the same environment entities, the class `DataAssociator` orders the possible correspondences assigning a score based on a similarity measure.

This measure has been chosen to be almost invariant to the rotation of the robot that happens between two scan matches.

In fact the length of the segments and also the orientation, assuming that the interval between scans is very small, would give assurance of only small changes.

Nevertheless the final decision for the best association is influenced, where it is possible to compute it, by an additional metric, which is the similarity between the sum of the angles that span from the examined segments and their respective neighboring segments. In fact this quantity would be invariant also with the respect of a significant rotation since it is an absolute quantity, not dependant on the robot orientation.

2.4 Scan Matcher

This class is responsible for the estimation of the rotation between scan matches. Once that some data associations have been identified, an iterative optimization (ICP) would be applied on points of the associated segments to recover the homogeneous transformation expressed in terms of $SE(2)$ with a matrix $T(\Delta x, \Delta y, \Delta \theta)$.

The state evolution is carried out on a manifold and stored on a graph.

2.5 Graph Manager

In the nodes of the graph is contained the state estimated at each iteration together with the features extracted and the laser scanner ranges. Moreover in the edges there are the transformations coming from scan-matching and also the data associations found.

With all this information the Graph class carries out a preprocessing operation in order to reduce the complexity of the loop closure phase: it tries to assemble the features of subsequent iterations in *trails*. It means that if a certain segment of scan k_n is associated with another that belongs to the scan k_{n+1} and finally the second is associated with a third belonging to scan k_{n+2} they will be assembled in a *trail*.

A *trail* can be treated as a single feature that could be identified with the first segment

of the trail, additionally the more would be its length, the more text weight it will have on the loop recognition procedure.

2.6 Loop Recognitor

The LoopCloser class uses a distance map approach to evaluate loop closure candidates. At first, whenever the loop recognition function is triggered, it divides all the available *trails* at the current iteration into two subsets: the **query set**, which contains the most recent ones, and the **tree set** in which there are all that belong to iterations remote in time.

Then it exploits the KD tree implementation available and feeds it with the **tree set** *trails*. Then it queries the KD tree with the other set to obtain some candidates.

After that those candidates will be validated and then a score will be assigned to every one of them.

Only the candidates that have a score greater than a threshold will be treated as legitimate loop closures.

2.7 GUI rendering and tests

Last but not least, all the graphical interface animation has been created using the opencv library.

Consequently all the computations to generate the points to be colored are being carried out from the Map and Drawer classes.

The overall proceedings of the project have been verified with unit tests to certify function behaviours, avoiding the introduction of regression bugs.