Introduction to SLAM in ROS

Matteo Scucchia

ALMA MATER STUDIORUM — Bologna University, Cesena

21 May 2021

Outline

ROS introduction

ROS (Robot Operating System) provides libraries and tools to help software developers create robot applications. It provides hardware abstraction, device drivers, libraries, visualizers, message-passing, package management, and more [9]. It is available for Linux and Windows 10. Versioning is strongly coupled with Linux:

- ROS Kinetic for Linux 16
- ROS Melodic for Linux 18
- ROS Noetic for Linux 20

ROS as distributed framework

ROS is a distributed framework of processes called Nodes that enables executables to be individually designed and loosely coupled at runtime. Nodes can run on different machines and the communication is based on message exchange.

Pro:

- code reusability
- modularity
- interoperability

Cons:

- message passing is still slower than shared memory
- delay introduced by message hashing
- necessary define conversions between objects and messages

ROS Computation Graph

The Computation Graph is the peer-to-peer network of ROS processes that are processing data together. The main Computation Graph concepts of ROS are:

- Master
- Nodes
- Messages
- Topics
- Services
- Bags

ROS Master

The ROS Master provides name registration and lookup to the rest of the Computation Graph. Without the Master, nodes would not be able to find each other, exchange messages, or invoke services.

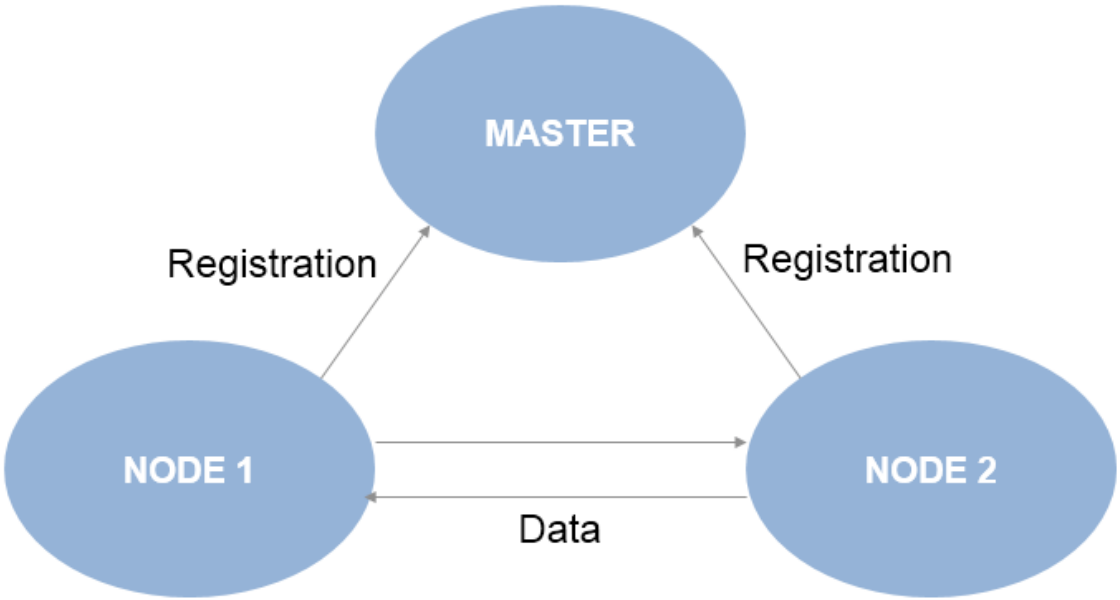

Figure: ROS master-nodes communication

ROS Nodes

Nodes are processes that perform computation. They should be fine-grained and specifically to solve a single task. Nodes are typically written in C++ or Python, but other languages are supported too.

An example:
`https://github.com/xaxxontech/autocrawler_ros/tree/master/src`[11]

Figure: Xaxxon autocrawler

ROS Messages

Nodes communicate with each other by passing messages. A message is simply a data structure, comprising typed fields, with .msg extension.

Complex messages are composed hierarchically from fixed types, such as primitive types and arrays (`http://wiki.ros.org/msg`).

There are several types of message implemented and provided by ROS (`http://wiki.ros.org/common_msgs?distro=melodic`).

ROS Topics

Messages are routed via a transport system with publish/subscribe semantics. A node sends out a message by publishing it to a given topic. The topic is a name that is used to identify the content of the message. A node that is interested in a certain kind of data will subscribe to the appropriate topic.

```
matteo@matteo-MSI:~$ rostopic list
/camera/aligned_depth_to_color/camera_info
/camera/aligned_depth_to_color/image_raw
/camera/color/camera_info
/camera/color/image_raw
/clock
/odom
/rosout
/rosout_agg
/scan
/tf
/tf_static
```

Figure: Used topics

ROS Services

Services are structures appropriate for request/response interactions.
Services are defined by a pair of message structures: one for the request and one for the reply, with extension .srv.

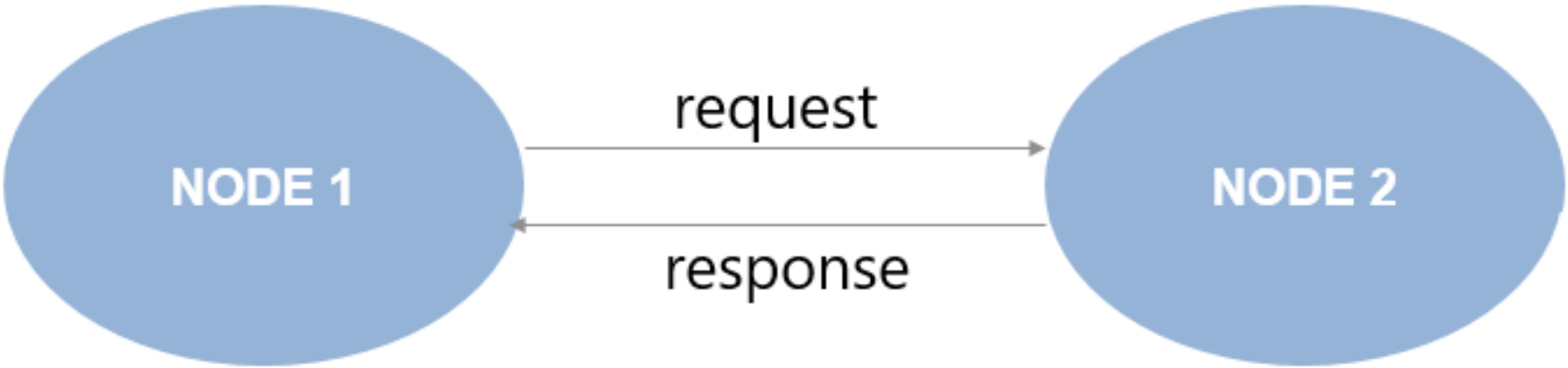

Figure: Service communication

ROS Bags

Bags are a format for saving and playing back ROS message data. Bags are an important mechanism for storing data, such as sensor data, that can be difficult to collect but is necessary for developing and testing algorithms.

ROS communication example

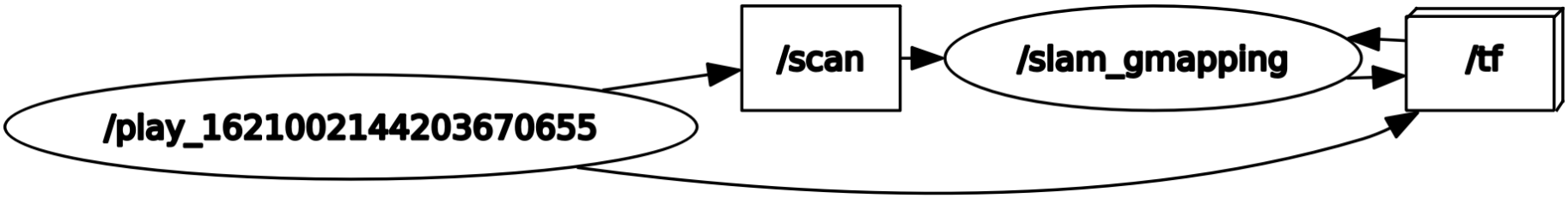

Figure: Simple ROS computation graph

Nodelets

Nodelets are designed to provide a way to run multiple algorithms on a single machine, in a single process, without incurring copy costs when passing messages intraprocess.

Nodelet everything paradigm [8]: all things are nodelets.
Instead of use different processes, that are Nodes, Nodelets are different threads.
Very useful when the entire computation takes place on a single machine.

ROS thread models

ROS allows the use of several different thread models, based on single or multi threading.

The basic thread model of a node is an event loop. A callback is called if the previous one is ended, otherwise the message is queued.

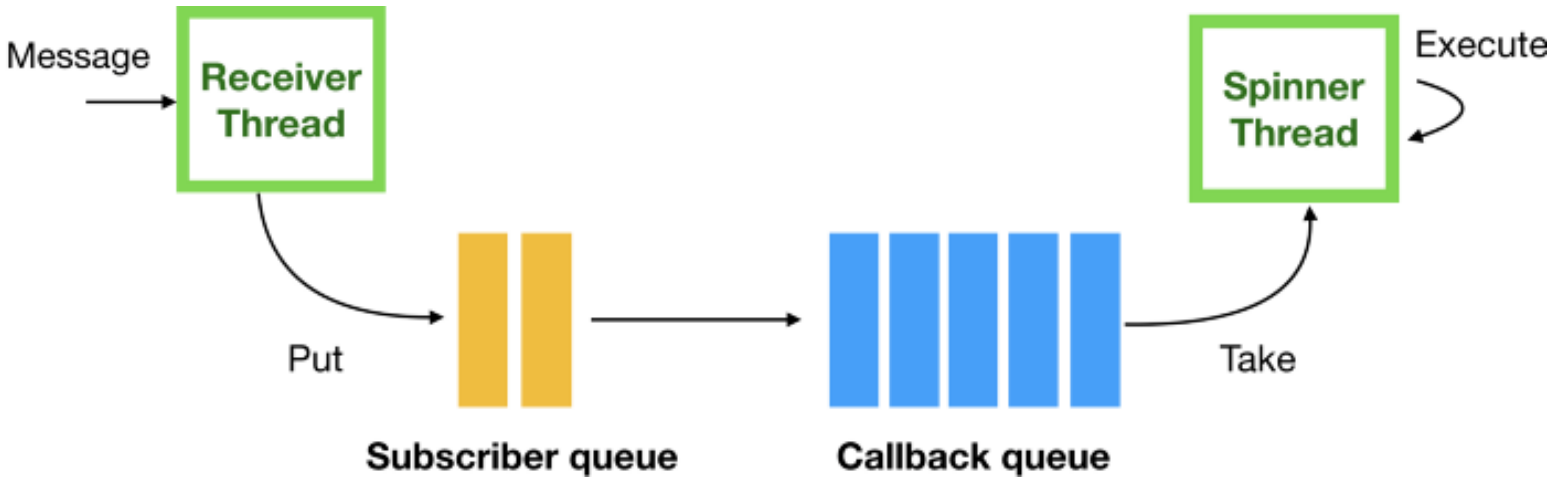

Figure: Single thread model [7]

Rviz

Rviz is a 3D visualizer for the Robot Operating System (ROS) framework.

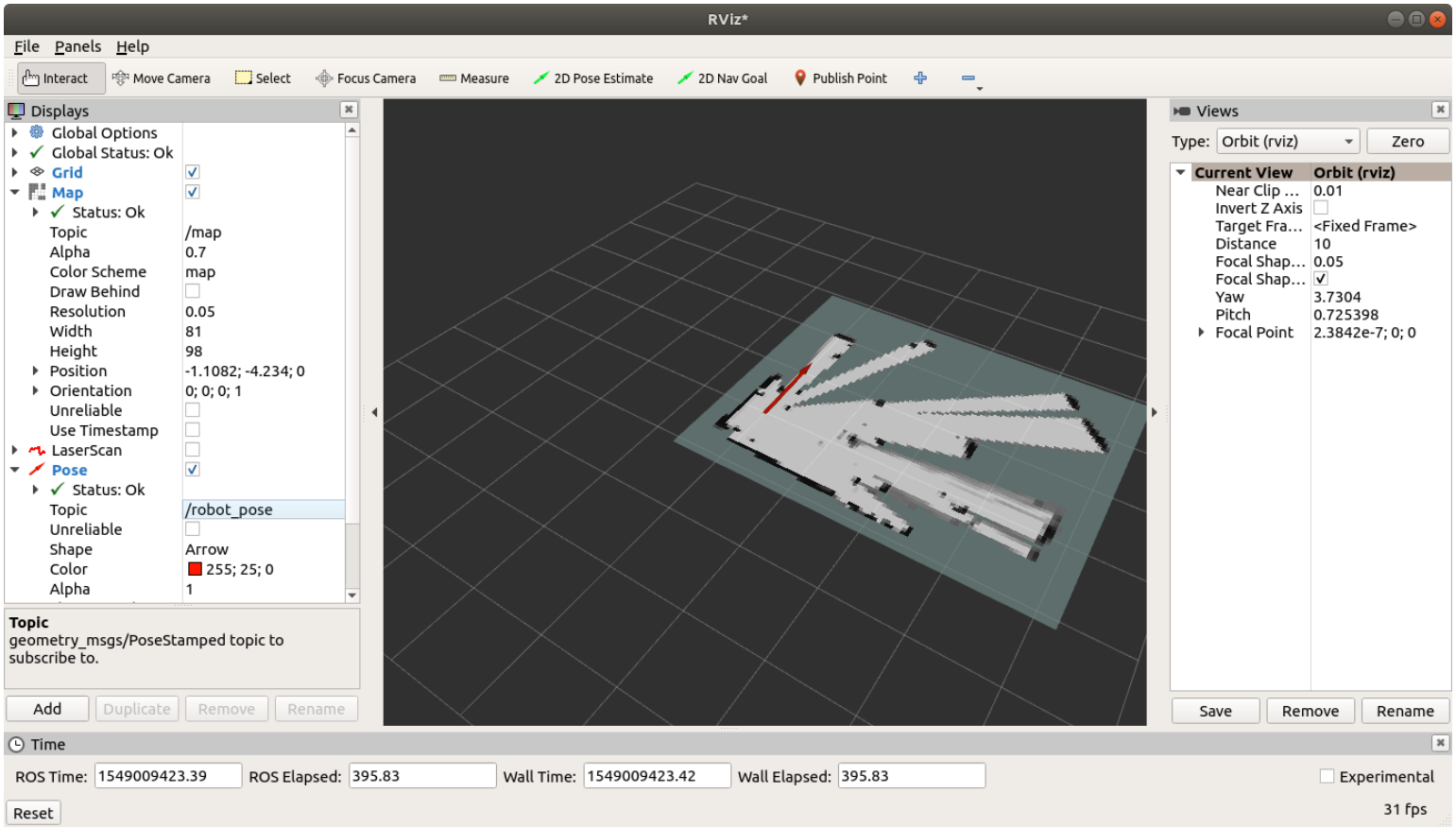

Figure: Rviz window

SLAM

SLAM (Simultaneous Localization And Mapping) is a process in which a mobile robot has to build a map of the environment and simultaneously localize itself on the map [4].

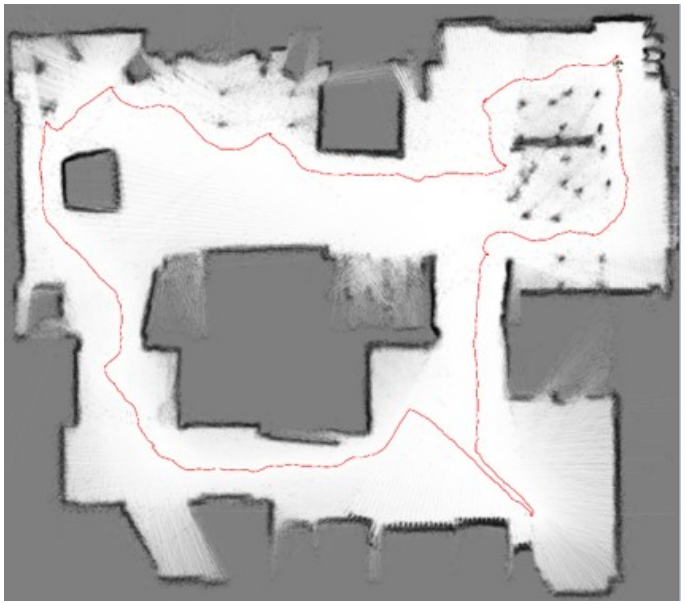

Figure: Map obtained through SLAM

Main concepts

Three fundamental concepts about SLAM:

- it is a probabilistic problem
- it must be a real-time process
- it assumes that the world is static

In theory, SLAM is a solved problem, several solutions have been proposed in literature, but it's not in practice.

Why SLAM?

Several reasons:

- for indoor applications where a map is needed but not provided, such as path planning
- the goal is to obtain the map of the environment
- provide to the robot a topological knowledge of the world

For the previous tasks, SLAM is the state-of-the-art technique.

Related concepts

SLAM is multi-disciplinary problem.

Concepts related to SLAM:

- landmarks
- data association
- odometry
- loop closure detection

Landmarks

Landmarks represent important points in the map. The formal definition of "landmark" depends on the sensors used by robot.

Figure: Visual landmarks

Data association

Data association is a recognition process, in which the robot must associate the same observations taken at different time.

Two types of data association:

- short-term data association
- long-term data association

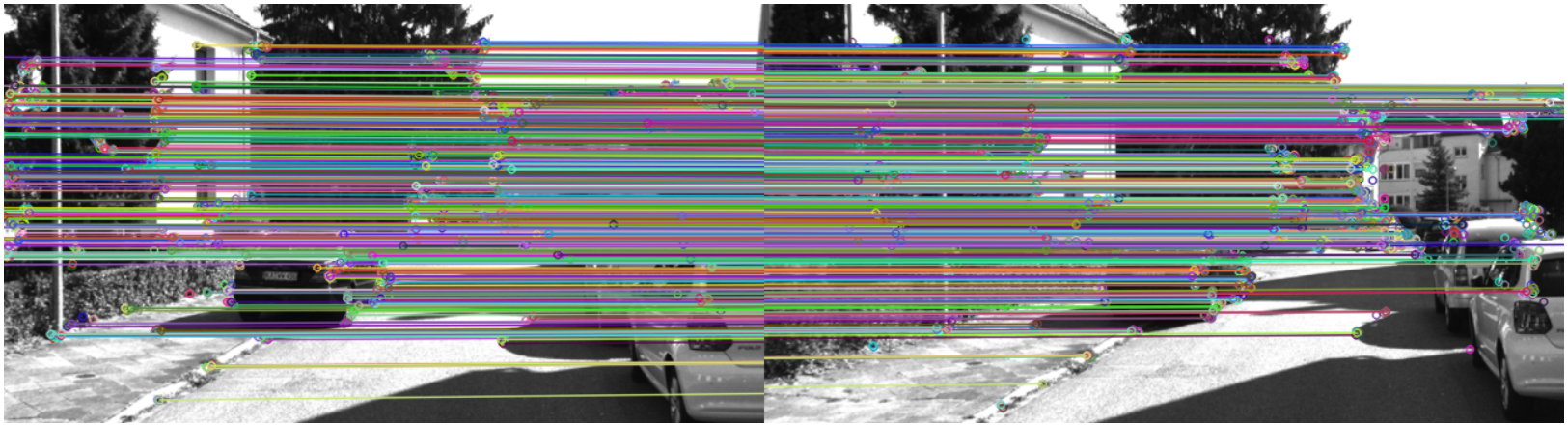

Figure: Short-term data association

Odometry

The word "odometry" is composed from the Greek words "odos", that means "route" and "metron", that means "measure" [10]. It's the use of data from motion sensors to estimate the change in position over time. Based on the sensor, there are several types of odometry:

- wheel odometry
- visual odometry
- LiDAR odometry
- inertial odometry
- a combination of the above

SLAM problem

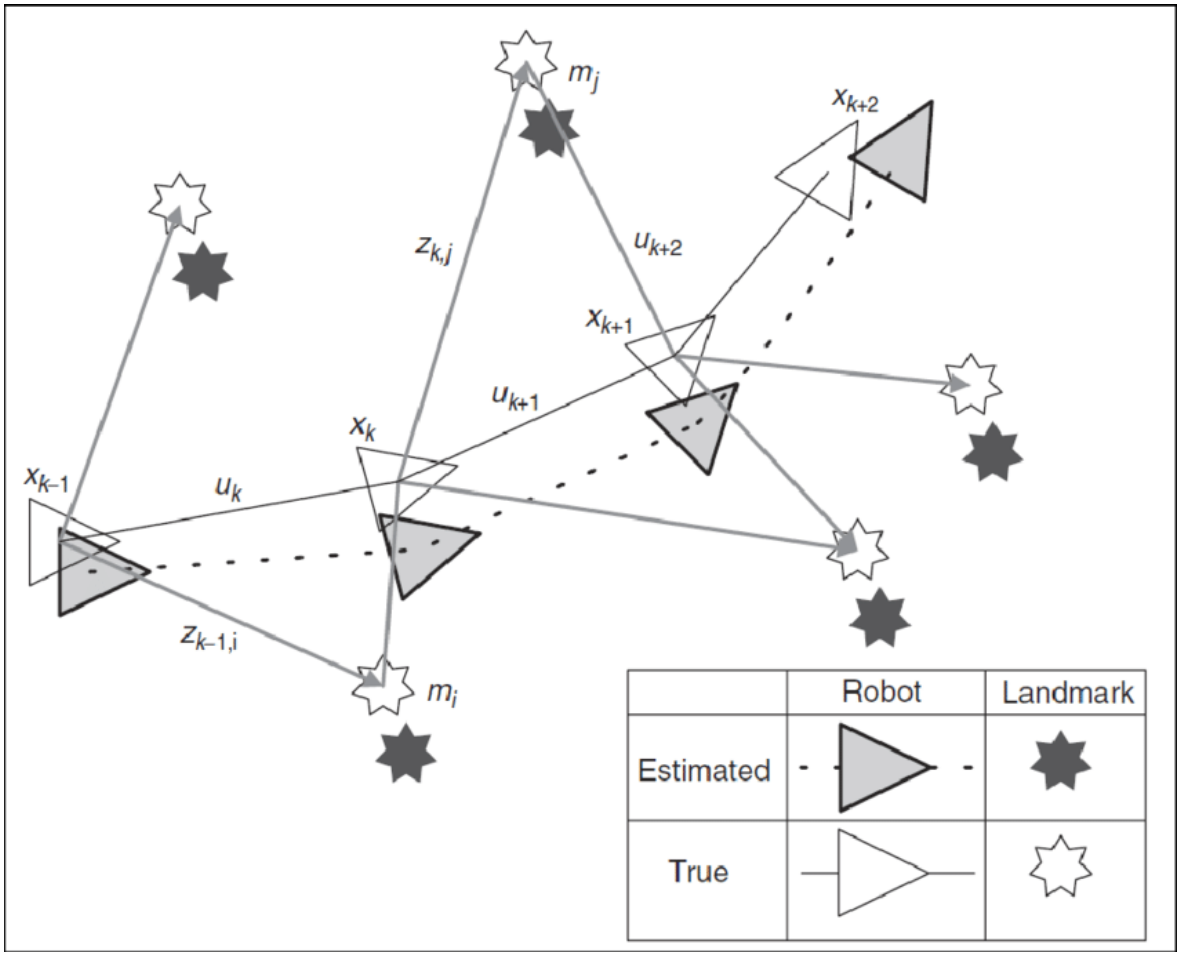

Figure: SLAM problem representation

Loop closure detection

Loop Closure Detection (LCD) is the process of determining whether an agent is returned to a previously visited location by analyzing data from its sensors [6].

When a loop closure detection is performed, the map can be adjusted.

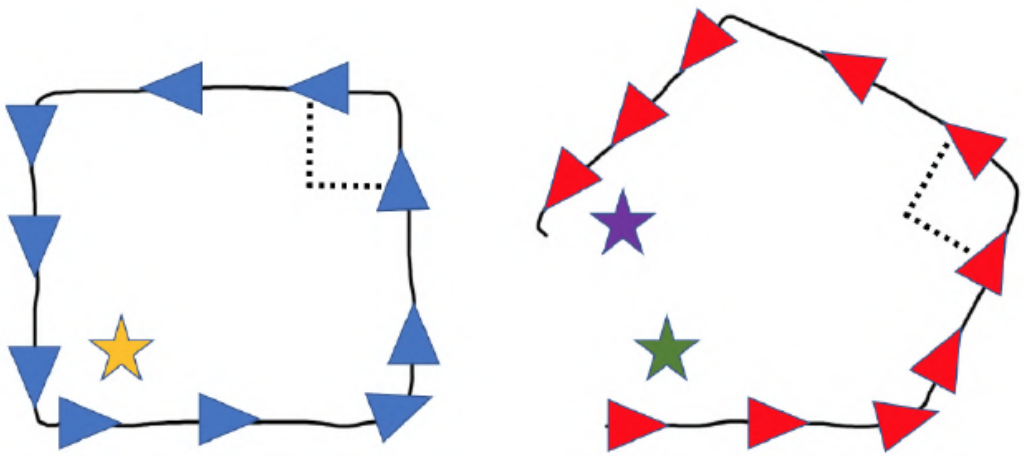

Figure: Loop closure detection process

An unsolved problem

Loop Closure Detection is the unsolved problem of SLAM. It can be considered solved only under certain conditions (static world, small environments...).

Why is it so hard?

- real-time constraint
- memory space
- the world is not static

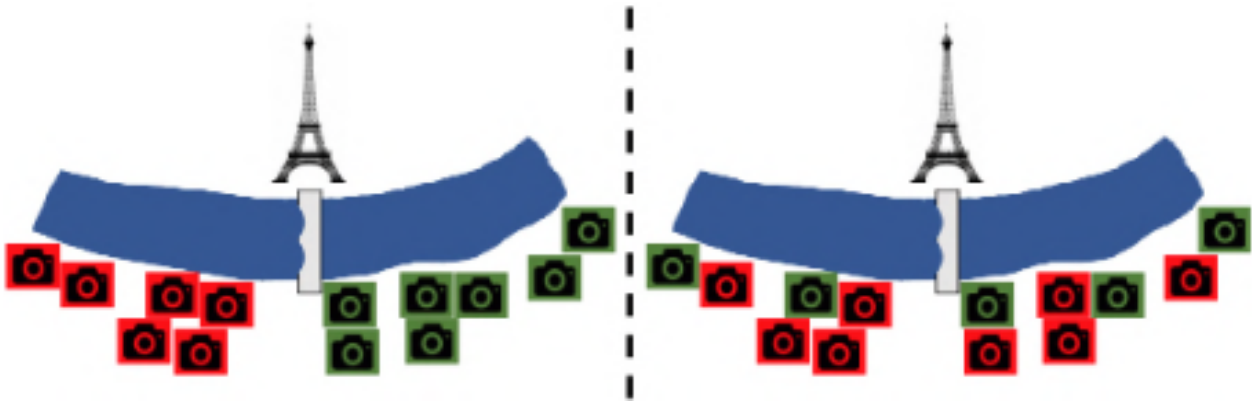

Figure: Loop closure detection ambiguity

Loop closure evaluation

The standard metric to evaluate the goodness of an algorithm in the detection of loop closures is Precision-Recall:

- false negative detection
- false positive detection

Typically LCD is not a reversible process, so false positives are much worse than false negatives.

Lifelong SLAM

Lifelong SLAM [3] is a term coined to indicate a long-term SLAM, capable of facing a huge map, with an environment that changes over time, in order to give a certain autonomy to a mobile robot.
Two main challenges of Lifelong SLAM:

- robustness
- scalability

What's next?

In order to improve SLAM, there is the necessity to provide to the robots a deeper understanding of the world:

- using different sensors for specific tasks
- using odometry information to avoid false positive detection
- using modern techniques of artificial intelligence fields

Deep learning techniques can bring important improvements to Loop Closure Detection and consequently to Lifelong SLAM [1]:

- continual learning
- object recognition + segmentation [2]
- GANs
- specific architectures for odometry [5][12]

Bibliography I

Saba Arshad and Gon-Woo Kim. "Role of Deep Learning in Loop Closure Detection for Visual and Lidar SLAM: A Survey". In: *Sensors* 21.4 (2021). ISSN: 1424-8220. DOI: 10.3390/s21041243. URL: https://www.mdpi.com/1424-8220/21/4/1243.

Berta Bescos et al. "DynaSLAM: Tracking, Mapping, and Inpainting in Dynamic Scenes". In: *IEEE Robotics and Automation Letters* 3.4 (Oct. 2018), pp. 4076–4083. ISSN: 2377-3774. DOI: 10.1109/lra.2018.2860039. URL: http://dx.doi.org/10.1109/LRA.2018.2860039.

Bibliography II

Cesar Cadena et al. "Simultaneous Localization And Mapping: Present, Future, and the Robust-Perception Age". In: *IEEE Transactions on Robotics* 32 (June 2016). DOI: 10.1109/TRO.2016.2624754.

H. Durrant-Whyte and T. Bailey. "Simultaneous localization and mapping: part I". In: *IEEE Robotics Automation Magazine* 13.2 (2006), pp. 99–110. DOI: 10.1109/MRA.2006.1638022.

Qiang Liu et al. "Unsupervised Deep Learning-Based RGB-D Visual Odometry". In: *Applied Sciences* 10.16 (2020). ISSN: 2076-3417. DOI: 10.3390/app10165426. URL: https://www.mdpi.com/2076-3417/10/16/5426.

Bibliography III


Samer B. Nashed. "A Brief Survey of Loop Closure Detection: A Case for Rethinking Evaluation of Intelligent Systems". In: *NeurIPS 2020 Workshop: ML Retrospectives, Surveys Meta-Analyses (ML-RSA)* (2020).


Kohei Otsuka. *ROS Spinning, Threading, Queuing*. URL: https://levelup.gitconnected.com/ros-spinning-threading-queuing-aac9c0a793f.


Clearpath Robotics. *Odometry*. URL: https://www.clearpathrobotics.com/assets/guides/kinetic/ros/Nodelet%20Everything.html.


Open Robotics. *Documentation*. URL: http://wiki.ros.org/.

Bibliography IV


Wikipedia. *Odometry*. URL: https://en.wikipedia.org/wiki/Odometry.


Xaxxon. *Xaxxon - Documentation*. URL: http://www.xaxxon.com/.


Nan Yang et al. *D3VO: Deep Depth, Deep Pose and Deep Uncertainty for Monocular Visual Odometry*. 2020. arXiv: 2003.01060 [cs.CV].