CS 3630!

Lecture 18:
SLAM with LIDARS

Topics

1. **LIDAR**
2. **Localization with LIDAR**
3. **PoseSLAM: SLAM with ICP**
4. **The PoseSLAM Factor Graph**
5. **MAP = MPE = Nonlinear Optimization**
6. **Optimization with GTSAM**

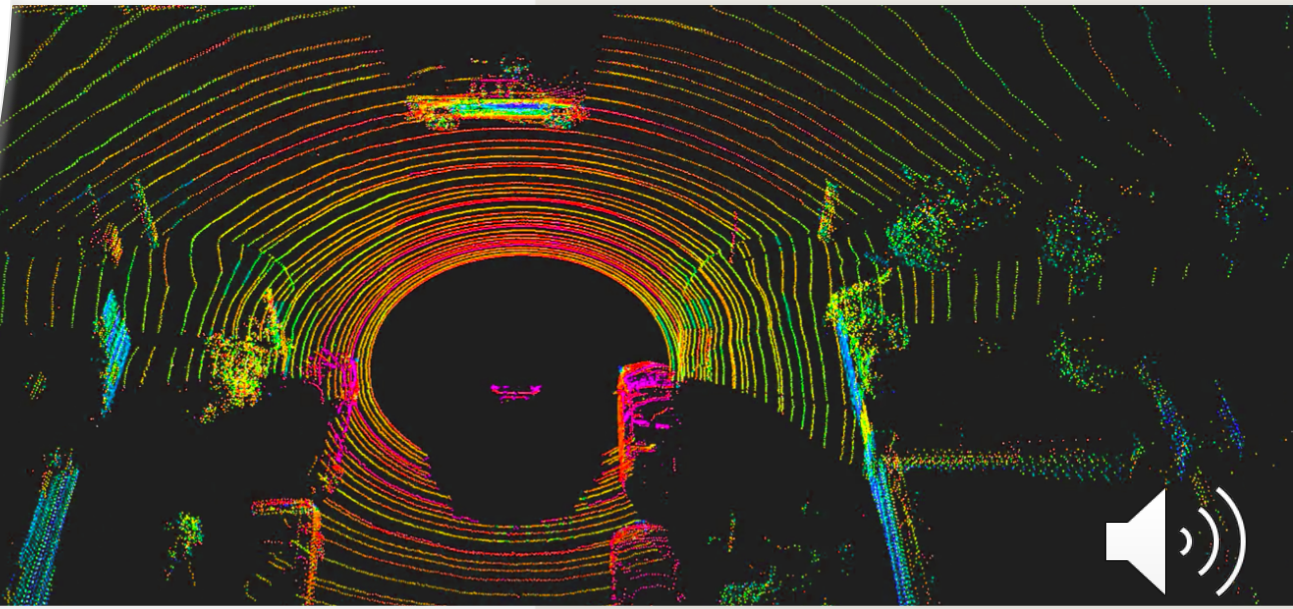
Motivation

- LIDAR = light detection and ranging
- Key sensor in Autonomous Driving
  - Used for localization
  - First, need a map to localize in!
- SLAM =
  Simultaneous Localization and Mapping
- Use Iterated Closest Points to relate scans
- Use optimization over SE(2) to do SLAM

Image by Voyage

1. LIDAR

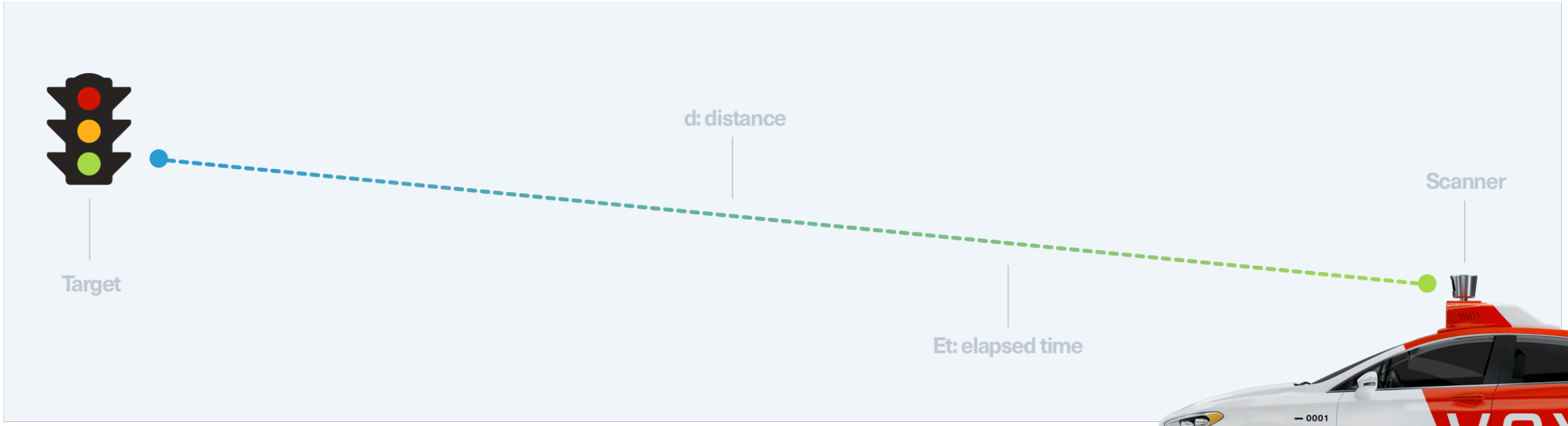
- Superpowers:
  - 360 Visibility
  - Accurate depth!
- Almost all AV prototypes have them (not all 360)



https://news.voyage.auto/an-introduction-to-lidar-the-key-self-driving-car-sensor-a7e405590cff

Images and exposition take from excellent Voyage Blog post

LIDAR Basic Principle

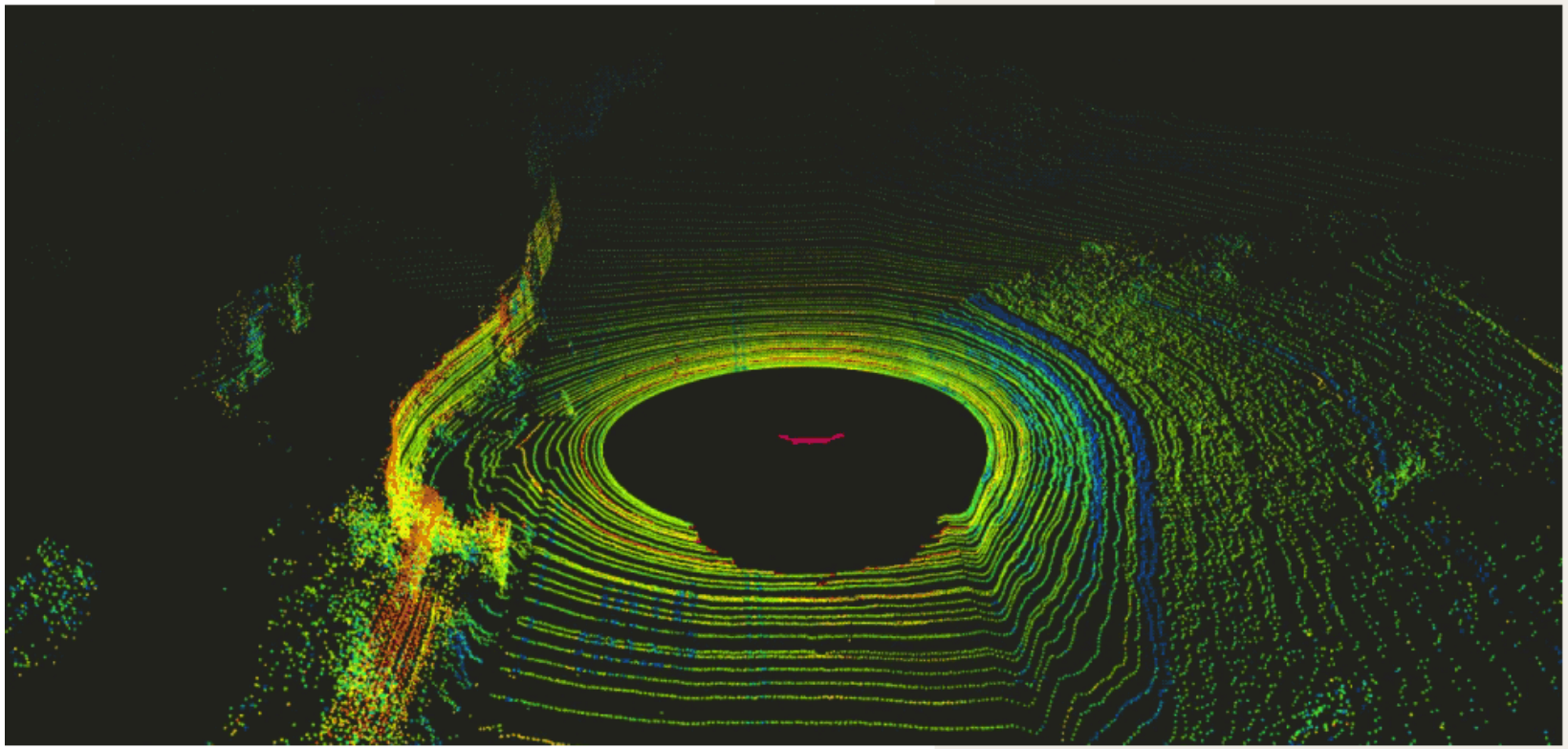


$$\frac{d = (Et \times c)}{2}$$ Where c = speed of light

- Send a light pulse
- Measure elapsed time Et
- Infer distance d

Images and exposition taken from excellent Voyage Blog post

Example



[Images and exposition take from excellent Voyage Blog post]

2. Localization with LIDAR

- ICP = Iterated Closest Points:
- Call current scan S, map M
- Predict pose from motion model: use other sensors if available
- Iterate:
  - For every point s: find closest m
  - Re-estimate pose
- In practice:
  - outlier rejection to account for moving objects, unmodeled structures, parked cars etc…

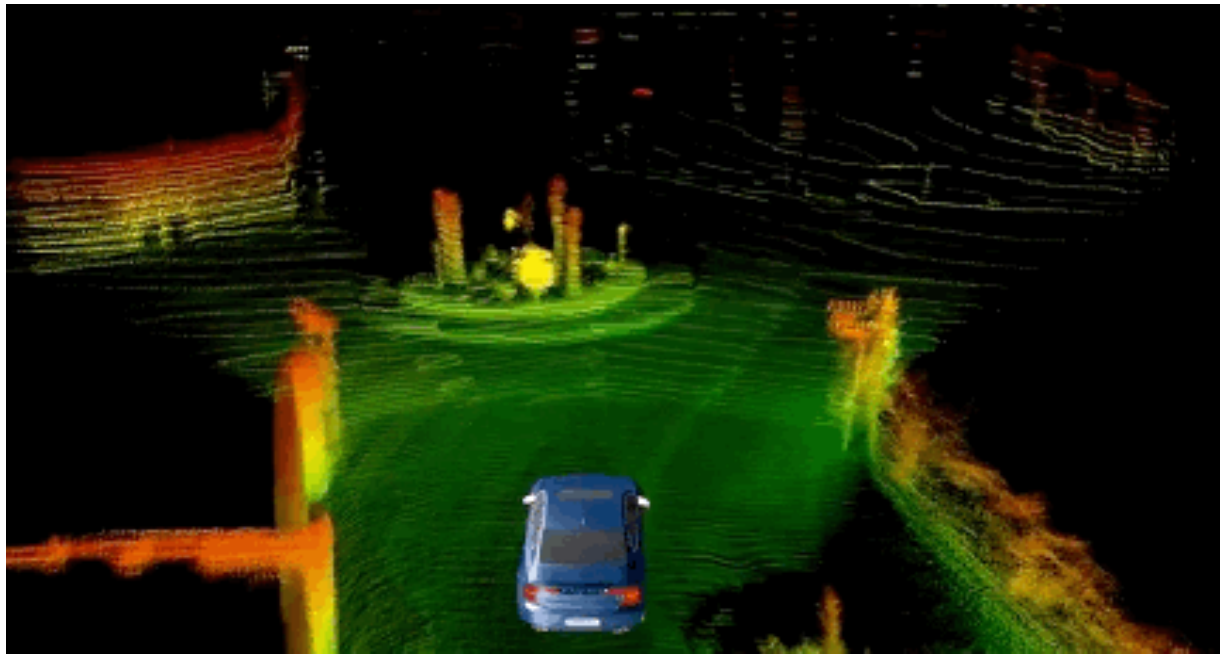
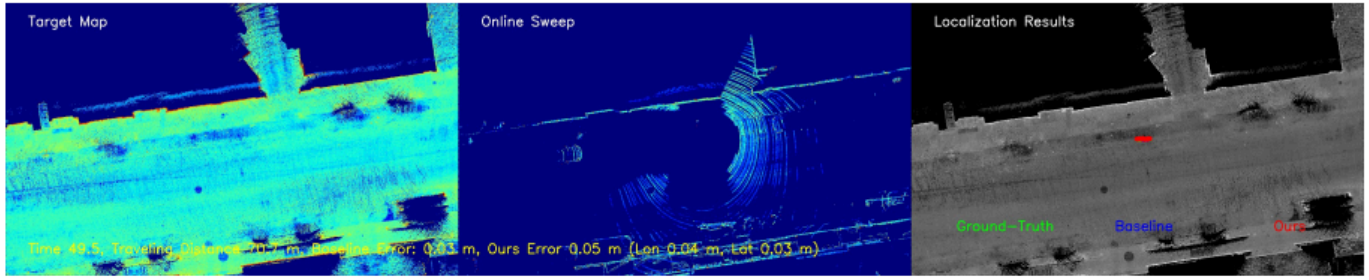


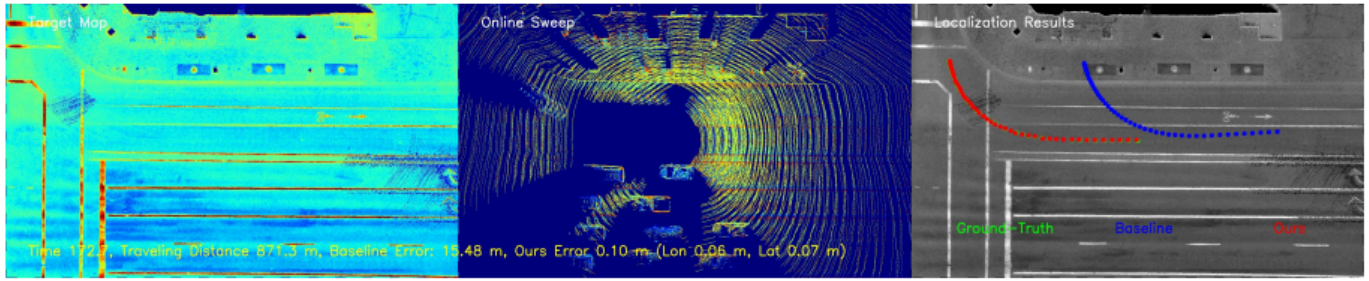
Image Credits: Innoviz

Still an active area of research

- E.g., recent paper from Uber ATG
- “reliable and accurate localization remains an open problem,”
- “[ICP] can lead to high-precision localization, but remain vulnerable in the presence of geometrically non-distinctive or repetitive environments, such as tunnels, highways, or bridges”



(c) Reverse parallel parking.



(d) A sharp turn into an intersection.

Learning to Localize Using a LiDAR Intensity Map

Ioan Andrei Bârsan[*,1,2] **Shenlong Wang**[*,1,2] **Andrei Pokrovsky**[1] **Raquel Urtasun**[1,2]
[1]Uber ATG, [2]University of Toronto
{andreib, slwang, andrei, urtasun}@uber.com

Abstract: In this paper we propose a real-time, calibration-agnostic and effective localization system for self-driving cars. Our method learns to embed the online LiDAR sweeps and intensity map into a joint deep embedding space. Localization is then conducted through an efficient convolutional matching between the embeddings. Our full system can operate in real-time at 15Hz while achieving centimeter level accuracy across different LiDAR sensors and environments. Our experiments illustrate the performance of the proposed approach over a large-scale dataset consisting of over 4000km of driving.

Keywords: Deep Learning, Localization, Map-based Localization

1 Introduction

One of the fundamental problems in autonomous driving is to be able to accurately localize the vehicle in real time. Different precision requirements exist depending on the intended use of the localization system. For routing the self-driving vehicle from point A to point B, precision of a few meters is sufficient. However, centimeter-level localization becomes necessary in order to exploit high definition (HD) maps as priors for robust perception, prediction, and safe motion planning.

Despite many decades of research, reliable and accurate localization remains an open problem, especially when very low latency is required. Geometric methods, such as those based on the iterative closest-point algorithm (ICP) [1, 2] can lead to high-precision localization, but remain vulnerable in the presence of geometrically non-distinctive or repetitive environments, such as tunnels, highways, or bridges. Image-based methods [3, 4, 5, 6] are also capable of robust localization, but are still behind geometric ones in terms of outdoor localization precision. Furthermore, they often require capturing the environment in different seasons and times of the day as the appearance might change dramatically.

A promising alternative to these methods is to leverage LiDAR intensity maps [7, 8], which encode information about the appearance and semantics of the scene. However, the intensity of commercial LiDARs is inconsistent across different beams and manufacturers, and prone to changes due to environmental factors such as temperature. Therefore, intensity based localization methods rely heavily on having very accurate intensity calibration of each LiDAR beam. This requires careful fine-tuning of each vehicle to achieve good performance, sometimes even on a daily basis. Calibration can be a very laborious process, limiting the scalability of this approach. Online calibration is a promising solution, but current approaches fail to deliver the desirable accuracy. Furthermore, maps have to be re-captured each time we change the sensor, e.g., to exploit a new generation of LiDAR.

In this paper, we address the aforementioned problems by learning to perform intensity based localization. Towards this goal, we design a deep network that embeds both LiDAR intensity maps and online LiDAR sweeps in a common space where calibration is not required. Localization is then simply done by searching exhaustively over 3-DoF poses (2D position on the map manifold plus rotation), where the score of each pose can be computed by the cross-correlation between the embeddings. This allows us to perform localization in a few milliseconds on the GPU.

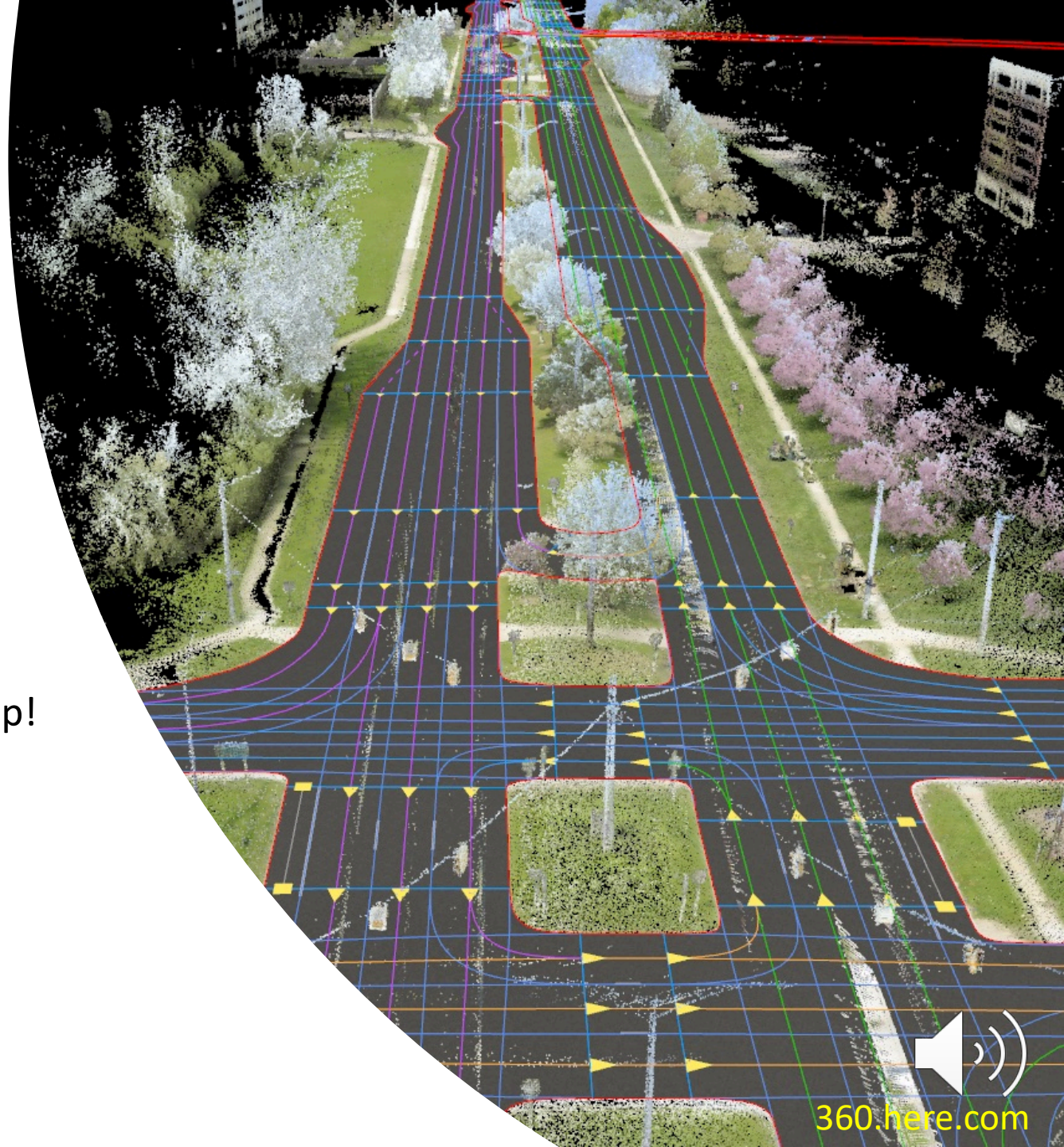
We demonstrate the effectiveness of our approach in both highway and urban environments over 4000km of roads. Our experiments showcase the advantages of our approach over traditional methods, such as the ability to work with uncalibrated data and the ability to generalize across different LiDAR sensors.

2nd Conference on Robot Learning (CoRL 2018), Zürich, Switzerland.

3. SLAM

- Mapping runs drive all accessible streets
- Record LIDAR, GPS, IMU (gyro + accel)
- SLAM: Simultaneous Localization and Mapping
  - Given a map, we can localize
  - Given accurate localization, we can build a map!
  - Do it simultaneously!

- HD-Map: point clouds + annotations



PoseSLAM: SLAM with ICP

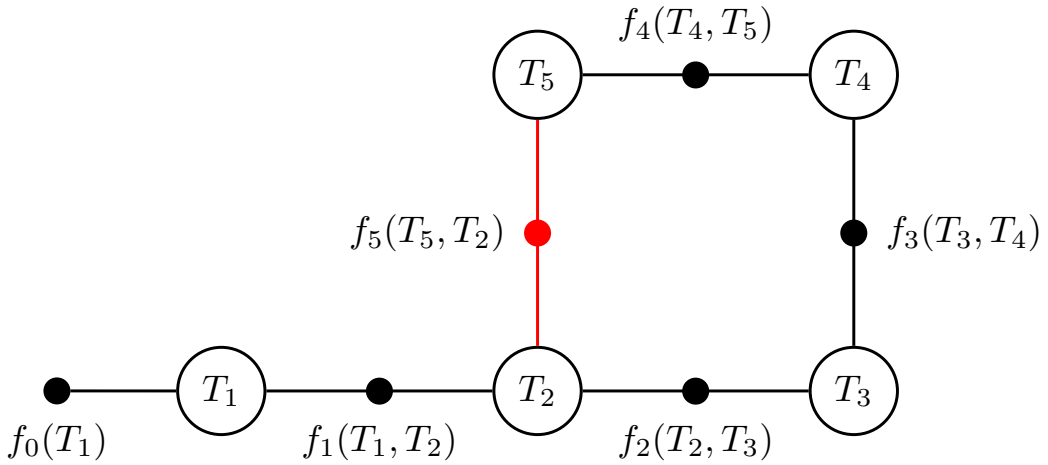
- One way: PoseSLAM:
  - Do ICP between overlapping scans
  - Can use GPS/IMU to decide which scans overlap
  - Optimize for 3D or 2D *poses* only
  - Re-construct HD map from laser-scans afterwards

4. The PoseSLAM Factor Graph

- Pose constraint = Factor
- MPE: maximize posterior

$$\phi(\mathcal{T}) = \prod_i \phi_i(\mathcal{T}_i)$$

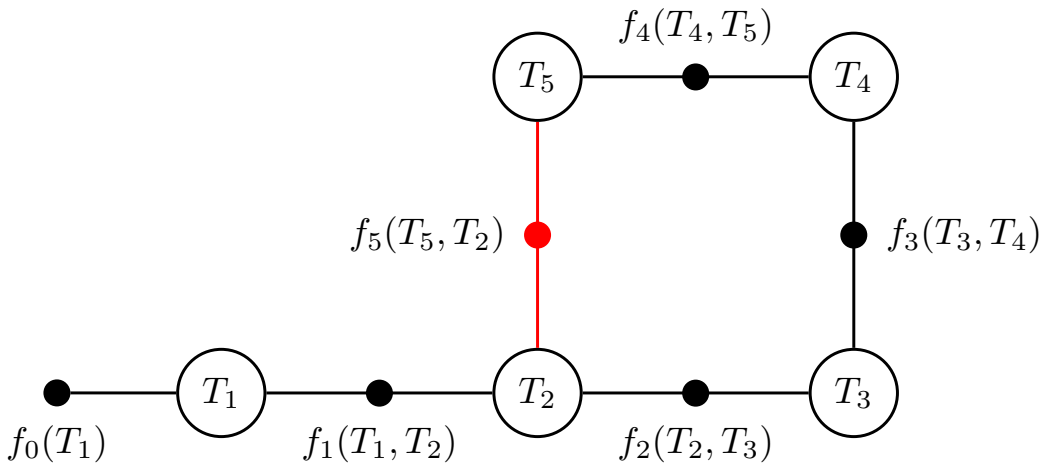
- In the example:
  - 4 constraints by matching successive scans
  - 1 “loop closure” constraint
  - 1 ”anchor” factor to give unique solution



Linear Least Squares

- If two assumptions hold:
  - measurement function is linear
  - Noise is zero-mean Gaussian

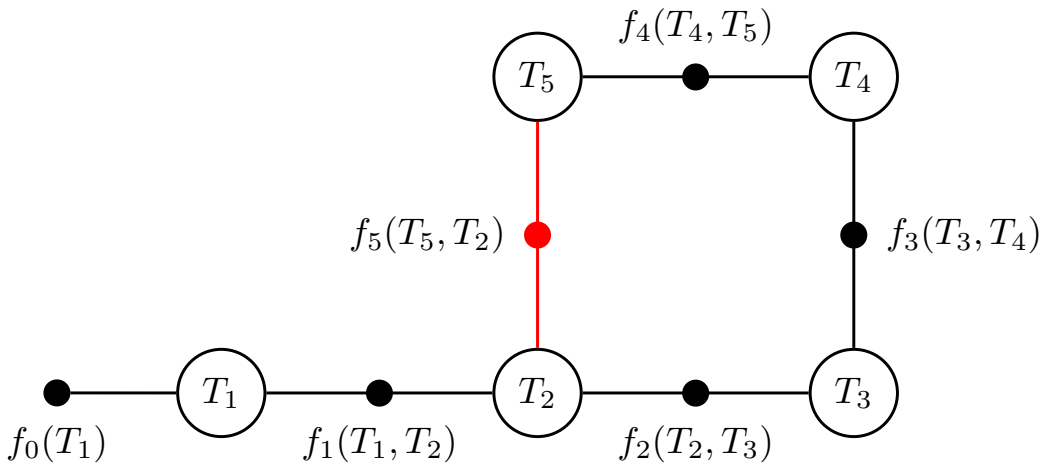
$$\mathcal{N}(x;\mu,\sigma^2) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left\{-\frac{1}{2}\left(\frac{x-\mu}{\sigma}\right)^2\right\}$$



Linear Least Squares

- If two assumptions hold:
  - measurement function is linear
  - Noise is zero-mean Gaussian

$$\mathcal{N}(x;\mu,\sigma^2) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left\{-\frac{1}{2}\left(\frac{x-\mu}{\sigma}\right)^2\right\}$$



- Then we can solve via linear least squares.
- Example: x-coordinates only, minimize *prediction error*:

$$\tilde{x}_{ij} \approx h(x_i,x_j) = x_j - x_i \qquad \phi(x_i,x_j) = \frac{1}{\sqrt{2\pi}} \exp\left\{-\frac{1}{2}\left(x_j - x_i - \tilde{x}_{ij}\right)^2\right\}$$

$$\mathcal{X}^* = \arg\min_{\mathcal{X}} \sum_k \frac{1}{2}\left(h(x_i,x_j) - \tilde{x}_{ij}\right)^2 = \arg\min_{\mathcal{X}} \sum_k \frac{1}{2}\left(x_j - x_i - \tilde{x}_{ij}\right)^2$$

Pose constraints are nonlinear!

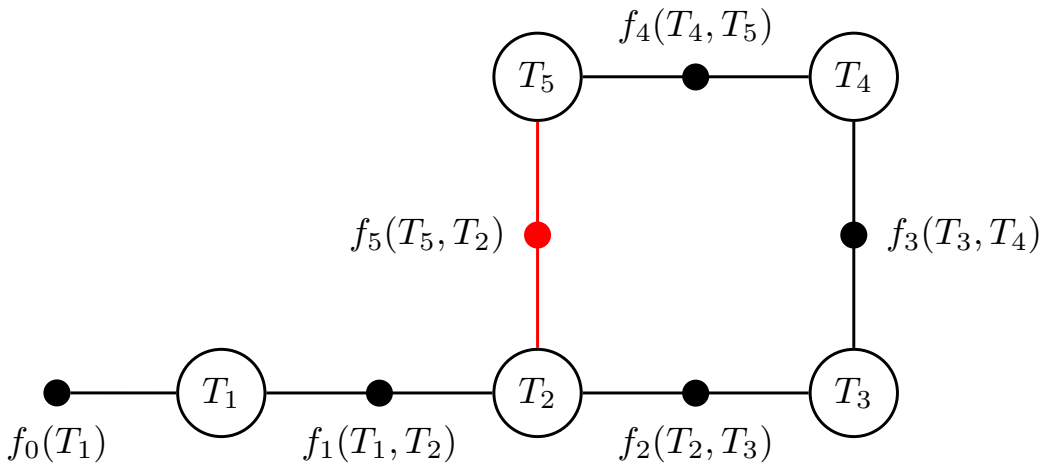
- Measurement prediction:

$$h(T_i, T_j) = T_i^{-1} T_j$$

- Measurement error:

$$\frac{1}{2} \left\| \log \left(\tilde{T}_{ij}^{-1} T_i^{-1} T_j \right) \right\|^2$$

- Here log[1] is a magic function converting a pose to three numbers $\xi \doteq (\delta x, \delta y, \delta\theta)$ that measure how far a pose is from the origin



[1] technically, matrix logarithm, the inverse of the matrix exponential exp.

5. MAP = MPE = Nonlinear Optimization

- Two different approaches:
  - Rotation averaging: first find rotations consistent with the measurements, then recover the translations linearly as discussed above. Sub-optimal, but a good initial estimate for…
  - Nonlinear minimization: locally linearize the problem and solve the corresponding linear problem using least-squares, and iterate this until convergence

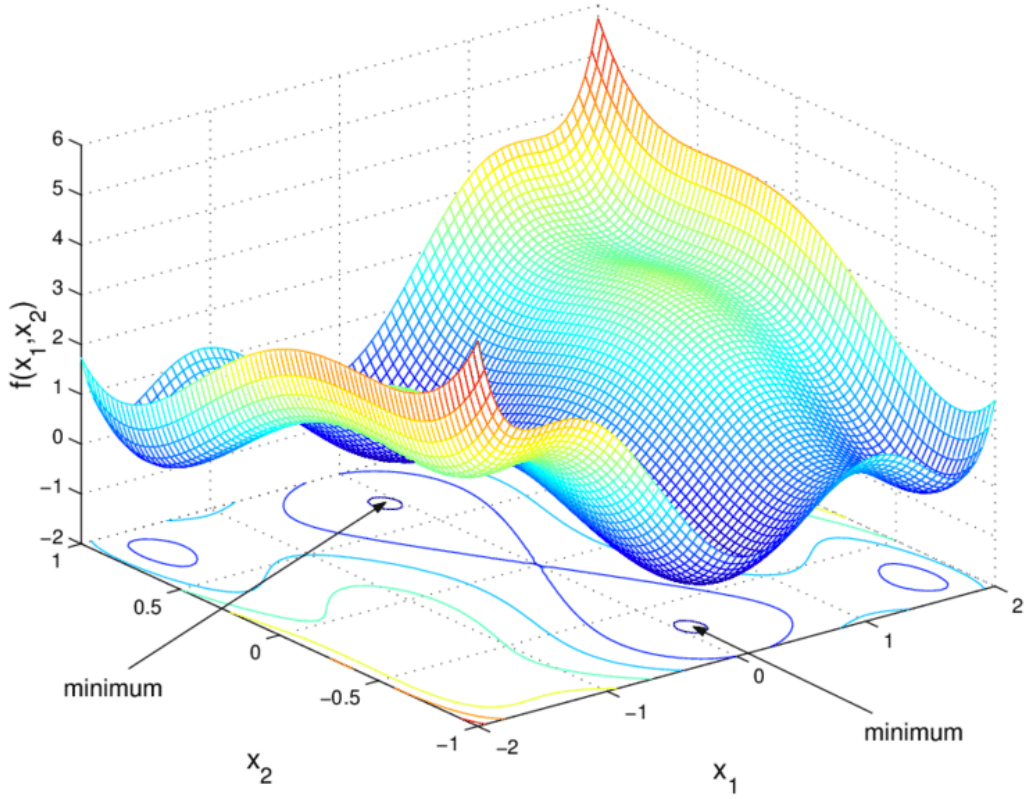


Image credit Tamás Terlaky

Incremental Pose Parameters

- Given an estimate for a pose, we can update it via

$$T \approx \bar{T}\Delta(\xi)$$

$$\xi \doteq (\delta x, \delta y, \delta\theta) \qquad \Delta(\xi) = \left[\begin{array}{cc|c} 1 & -\delta\theta & \delta x \\ \delta\theta & 1 & \delta y \\ \hline 0 & 0 & 1 \end{array}\right]$$

- With this we can approximate each factor linearly:

$$\frac{1}{2}\left\|\log\left(\tilde{T}_{ij}^{-1}T_i^{-1}T_j\right)\right\|^2 \approx \frac{1}{2}\left\|A_i\xi_i + A_j\xi_j - b\right\|^2$$

- Small print: For small increments, this works well, although we have to make sure to re-normalize the rotation afterwards. In practice, GTSAM uses something that holds even for large increments (an exponential map).

Solving a succession of linear problems

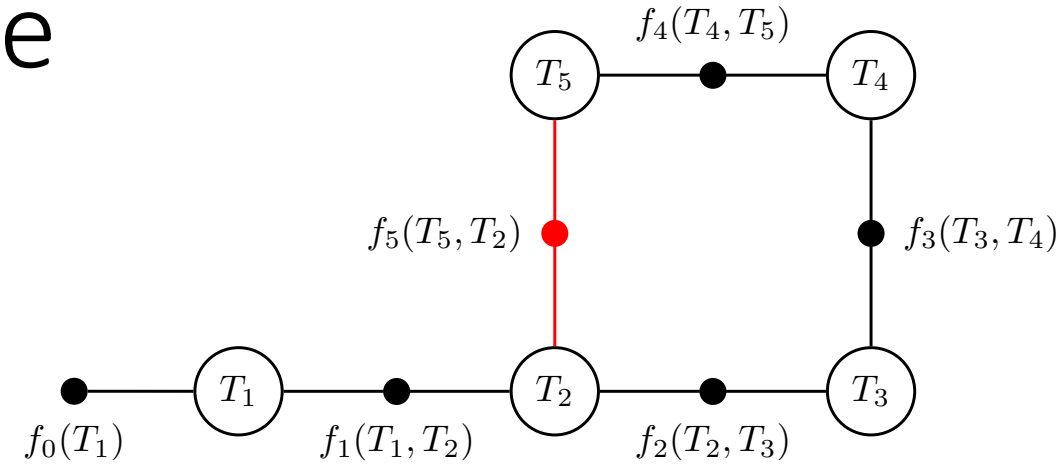
Summary:

- Start with an initial estimate $\mathcal{T}^0$
- Iterate:
  1. Linearize the factors $\frac{1}{2}\left\|\log\left(\tilde{T}_{ij}^{-1}T_i^{-1}T_j\right)\right\|^2 \approx \frac{1}{2}\left\|A_i\xi_i + A_j\xi_j - b\right\|^2$
  2. Solve the least squares problem $\Xi^* = \arg\min_\Xi \sum_k \frac{1}{2}\left\|A_{ki}\xi_i + A_{kj}\xi_j - b_k\right\|^2$
  3. Update $T_i^{t+1} \leftarrow T_j^t \Delta(\xi_i)$
- Until the nonlinear error $J(\mathcal{T}) \doteq \sum_k \frac{1}{2}\left\|\log\left(\tilde{T}_{ij}^{-1}T_i^{-1}T_j\right)\right\|^2$ converges.

6. Optimization with GTSAM

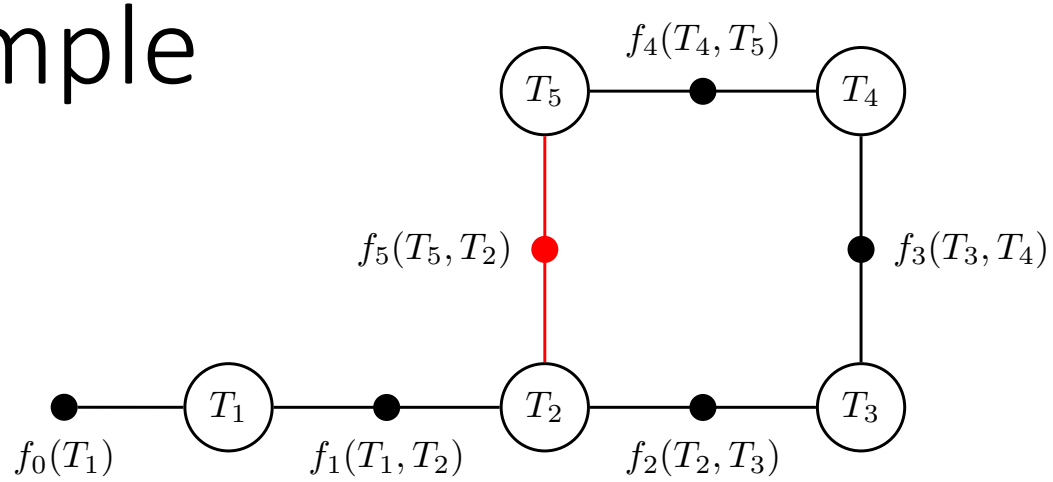
- The GTSAM toolbox (Georgia Tech Smoothing and Mapping) toolbox is a BSD-licensed C++ library based on factor graphs
- Website at http://gtsam.org.
- GTSAM exploits sparsity to be computationally efficient.

C++ Example




```
1  NonlinearFactorGraph graph;
2  auto priorNoise = noiseModel::Diagonal::Sigmas((Vector(3)<< 0.3, 0.3, 0.1))
3  graph.add(PriorFactor<Pose2>(1, Pose2(0,0,0), priorNoise));
4
5  // Add odometry factors
6  auto model = noiseModel::Diagonal::Sigmas((Vector(3)<< 0.2, 0.2, 0.1));
7  graph.add(BetweenFactor<Pose2>(1, 2, Pose2(2, 0, 0     ), model));
8  graph.add(BetweenFactor<Pose2>(2, 3, Pose2(2, 0, M_PI_2), model));
9  graph.add(BetweenFactor<Pose2>(3, 4, Pose2(2, 0, M_PI_2), model));
10 graph.add(BetweenFactor<Pose2>(4, 5, Pose2(2, 0, M_PI_2), model));
11
12 // Add pose constraint
13 graph.add(BetweenFactor<Pose2>(5, 2, Pose2(2, 0, M_PI_2), model));
```

Python Example

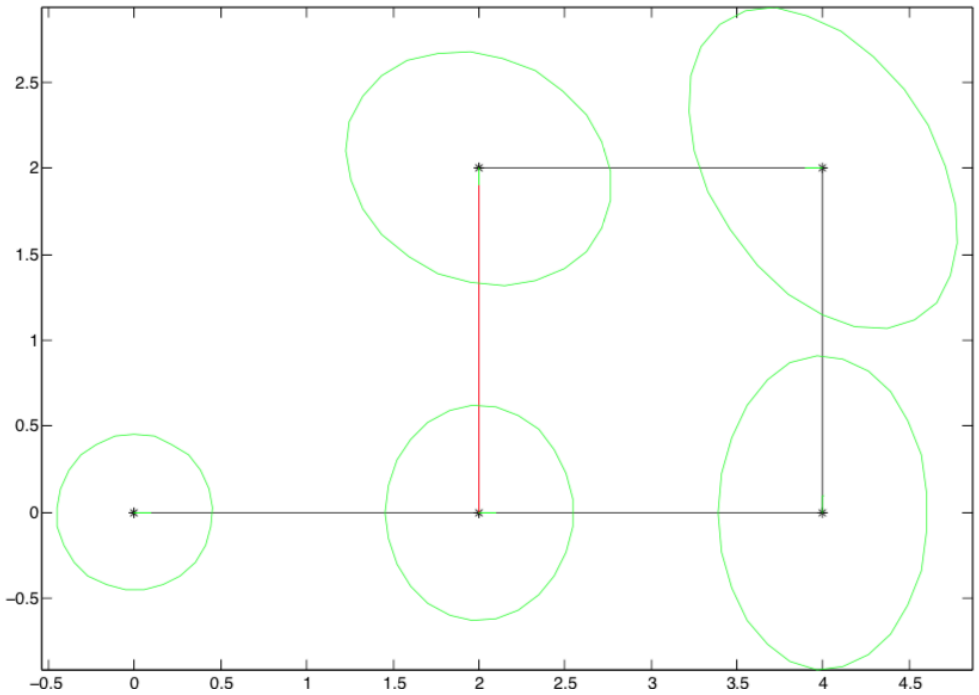



```
graph = gtsam.NonlinearFactorGraph()
priorNoise = gtsam.noiseModel_Diagonal.Sigmas(vector3(0.3, 0.3, 0.1))
graph.add(gtsam.PriorFactorPose2(1, gtsam.Pose2(0, 0, 0), priorNoise))

# Create odometry (Between) factors between consecutive poses
model = gtsam.noiseModel_Diagonal.Sigmas(vector3(0.2, 0.2, 0.1))
graph.add(gtsam.BetweenFactorPose2(1, 2, gtsam.Pose2(2, 0, 0), model))
graph.add(gtsam.BetweenFactorPose2(2, 3, gtsam.Pose2(2, 0, pi/2), model))
graph.add(gtsam.BetweenFactorPose2(3, 4, gtsam.Pose2(2, 0, pi/2), model))
graph.add(gtsam.BetweenFactorPose2(4, 5, gtsam.Pose2(2, 0, pi/2), model))

# Add the loop closure constraint
graph.add(gtsam.BetweenFactorPose2(5, 2, gtsam.Pose2(2, 0, pi/2), model))
```

Optimization in Python




```
# Create the initial estimate
initial_estimate = gtsam.Values()
initial_estimate.insert(1, gtsam.Pose2(0.5, 0.0, 0.2))
initial_estimate.insert(2, gtsam.Pose2(2.3, 0.1, -0.2))
initial_estimate.insert(3, gtsam.Pose2(4.1, 0.1, pi/2))
initial_estimate.insert(4, gtsam.Pose2(4.0, 2.0, pi))
initial_estimate.insert(5, gtsam.Pose2(2.1, 2.1, -pi/2))

# Optimize the initial values using a Gauss-Newton nonlinear optimizer
optimizer = gtsam.GaussNewtonOptimizer(graph, initial_estimate)
result = optimizer.optimize()
print("Final Result:\n{}".format(result))
```

Summary

1. LIDAR is a key sensor for autonomous driving
2. Localization can be done with LIDAR, or image-based
3. PoseSLAM: a SLAM variant using ICP pose constraints
4. The PoseSLAM factor graph graphically shows the constraints
5. MAP/MPE solution can be done via nonlinear optimization
6. GTSAM is an easy way to optimize over poses in C++/MATLAB/python