

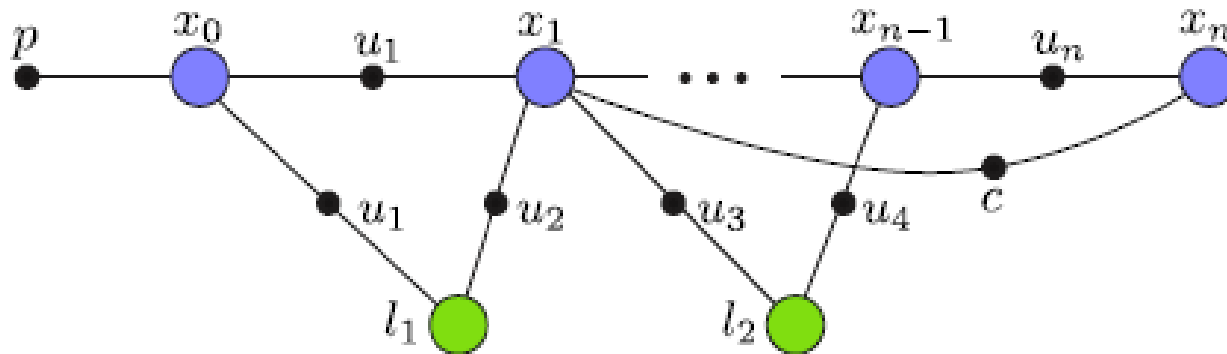
Localization using GTSAM and ROS

Bonus project for **IN315002 Autonomous Driving**

Technical University of Munich

Robotics and Embedded Systems

SS 2017



Source: MIT isam

Motivation: why localization?

- Autonomous cars need to **know** their position at **any** time
- GPS alone **error-prone** and **not always available**
- Car odometry **inaccurate** which leads to **accumulating error**
- Sensors such as Lidar or cameras can give us **information** about our **surroundings and the own position**
- In localization, a **map is available** which **represents the environment** (landmarks, images, scans, pointclouds...)

Motivation: why factor graphs?

- Localization can be solved using **Bayes/Kalman Filters, Monte Carlo Localization...**
- Factor graphs are the **most flexible solution**, as they
 - give us free control over the **type and amount of factors** we incorporate
 - allow the usage of **non-linear** motion and measurement models
 - can deal with **multi-modal** beliefs
 - break the task down into a graph problem that **scales well** with large environments and state representations
- More on the theory and GTSAM [here](#) and [here](#)

Task

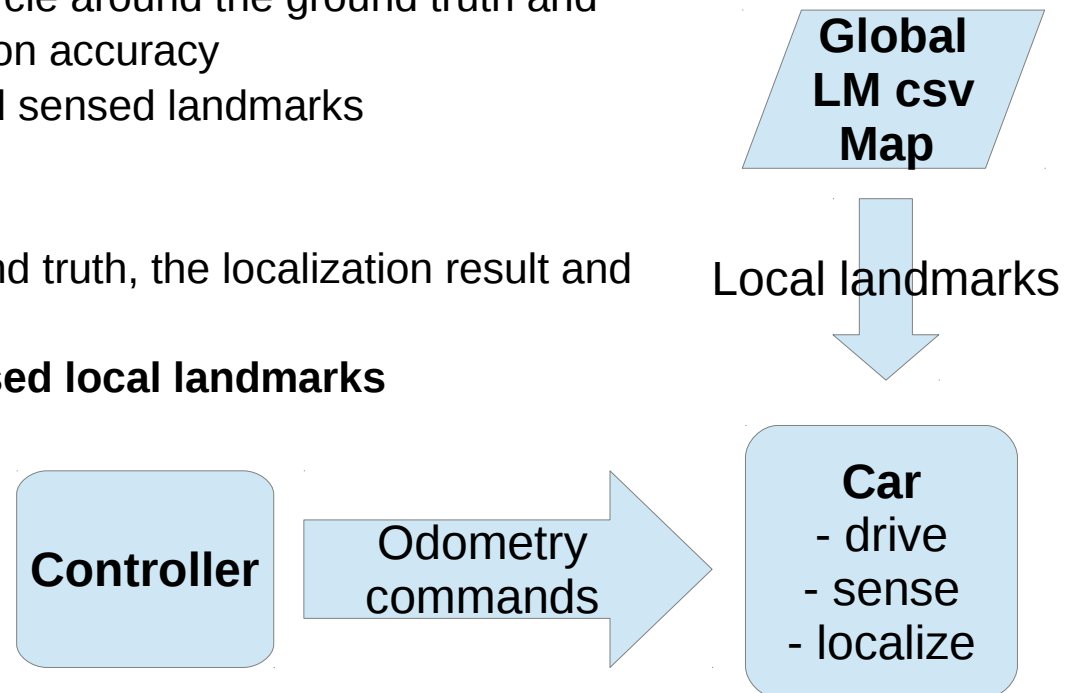
1. Define and visualize a rectangular vehicle
2. Parametrize sensor position and field of view as well as accuracy
3. Define a few landmarks
4. Let the rectangular car drive a curve between the landmarks
5. Localize the vehicle with GTSAM based on the parametrized positions and sensors

Implementation

- Usage of **GTSAM 4.0**
 - one of the most complete **c++ libraries** for factor graphs and SLAM
 - comes with a **Matlab interface** for fast evaluation/plotting
- Usage of **ROS Kinetic LTS**
 - **Open Source** Framework
 - includes many **efficient libraries and tools** for robotic applications
 - solves **communication** between separated packages in **real time**

System architecture

- Every **100 ms** the controller sends **odometry commands** (+ error estimates) that cause the car to drive in a circle
- After receiving the commands, the car
 - **simulates the driving** (to get the ground truth position)
 - **senses** the landmarks in a local circle around the ground truth and according to a prespecified detection accuracy
 - **localizes** itself given odometry and sensed landmarks
- Real time visualization (in **rviz**) of
 - the **path and position** of the ground truth, the localization result and the accumulated odometry
 - landmark **map** and **currently sensed local landmarks**



Localization

- Factors used are:
 - **Initial** factor for initial pose estimation
 - **Between** factor for relative (odometry) commands
 - **Unary** factors from the measurements

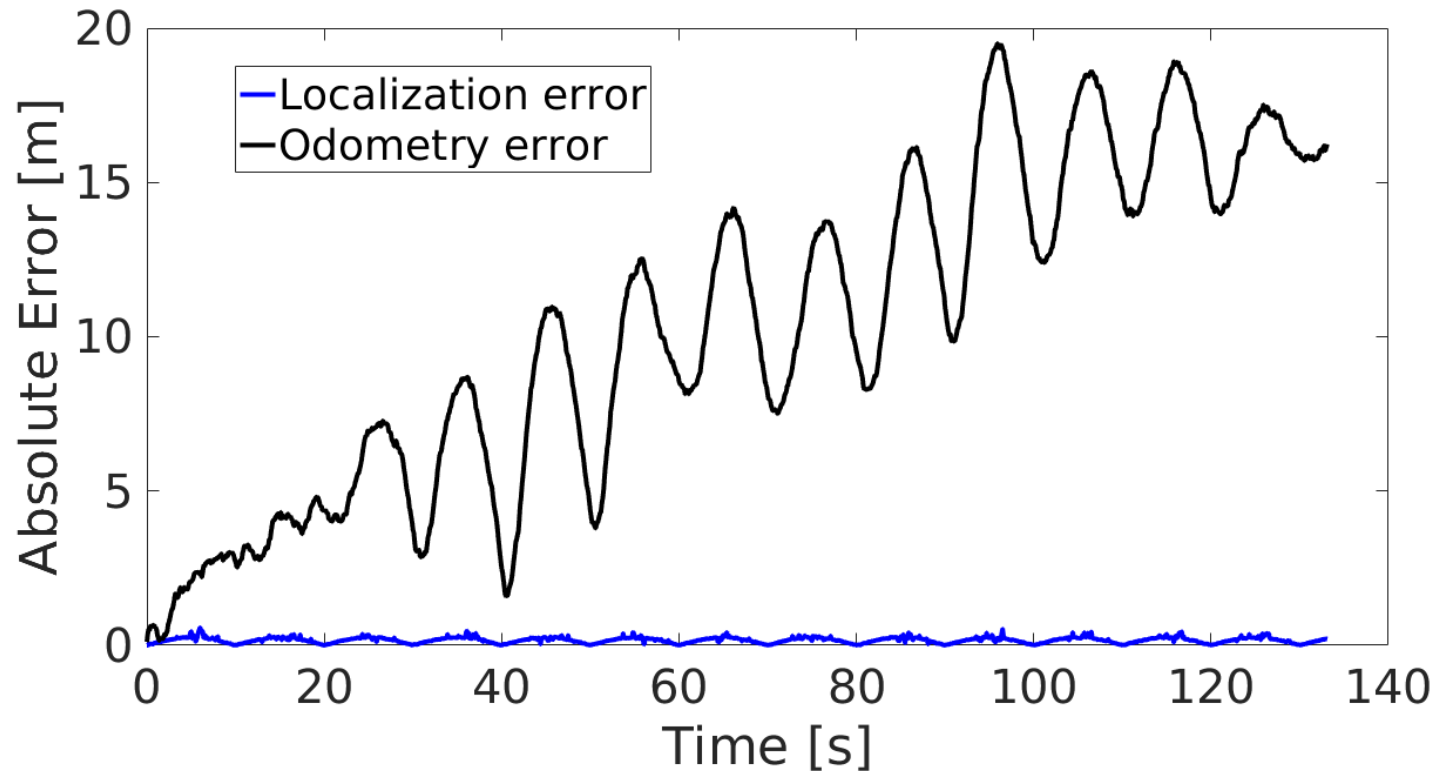
- Motion model:

$$\begin{pmatrix} x_{i+1} \\ y_{i+1} \\ \Theta_{i+1} \end{pmatrix} = \begin{pmatrix} x_i \\ y_i \\ \Theta_i \end{pmatrix} + \begin{pmatrix} v_x * \cos(\Theta_i) - v_y * \sin(\Theta_i) \\ v_x * \sin(\Theta_i) + v_y * \cos(\Theta_i) \\ v_{\Theta_i} * \Delta t \end{pmatrix}$$

- Measurement model: $h(z) = \begin{pmatrix} x \\ y \\ \Theta \end{pmatrix}$, altern. $h(z) = \begin{pmatrix} \sqrt{\Delta x^2 + \Delta y^2} \\ \text{atan}(\frac{\Delta y}{\Delta x}) - \Theta \end{pmatrix}$

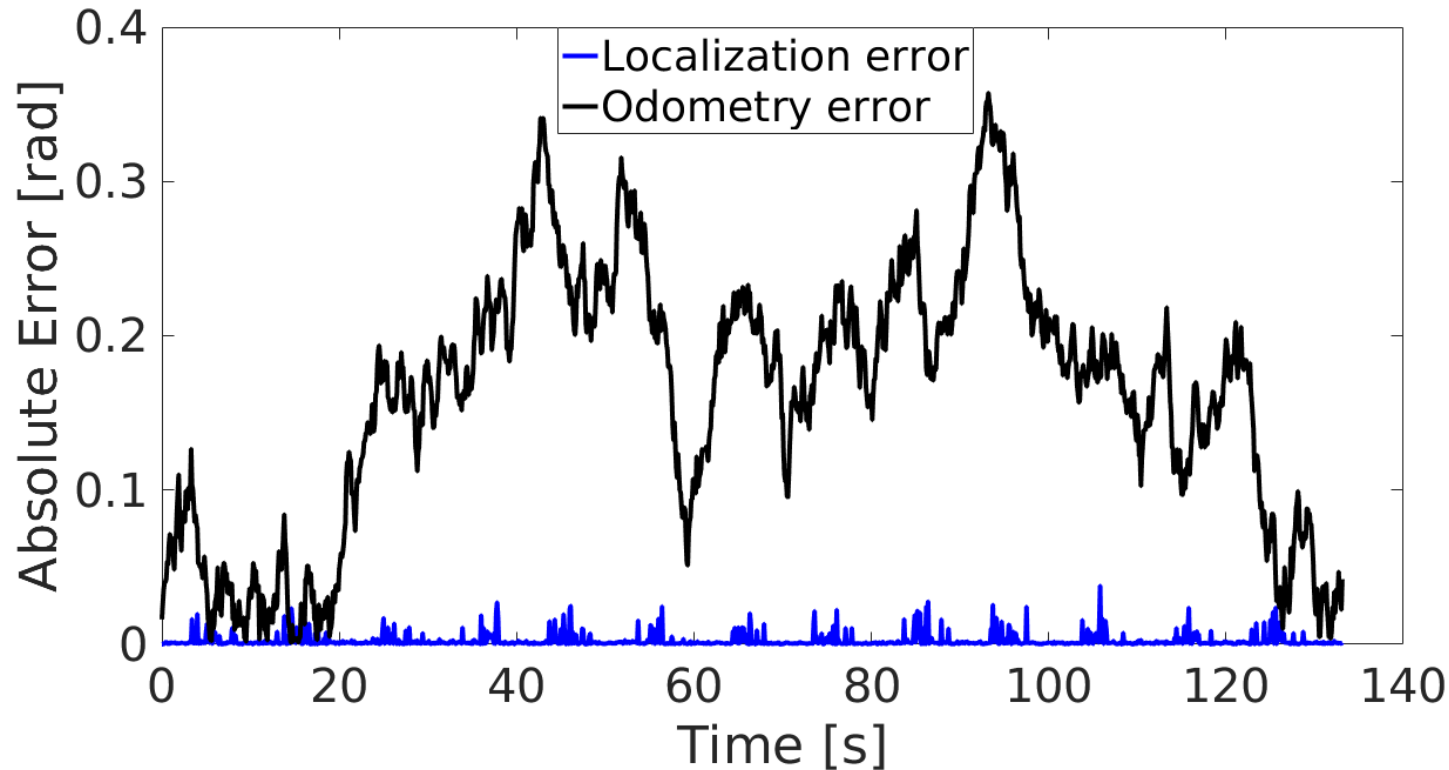
Error analysis

- **car_node.cpp** writes the localization result, the ground truth and the accumulated odometry into a **csv file**
- This file is read by a **Matlab script** that **plots the error** over time



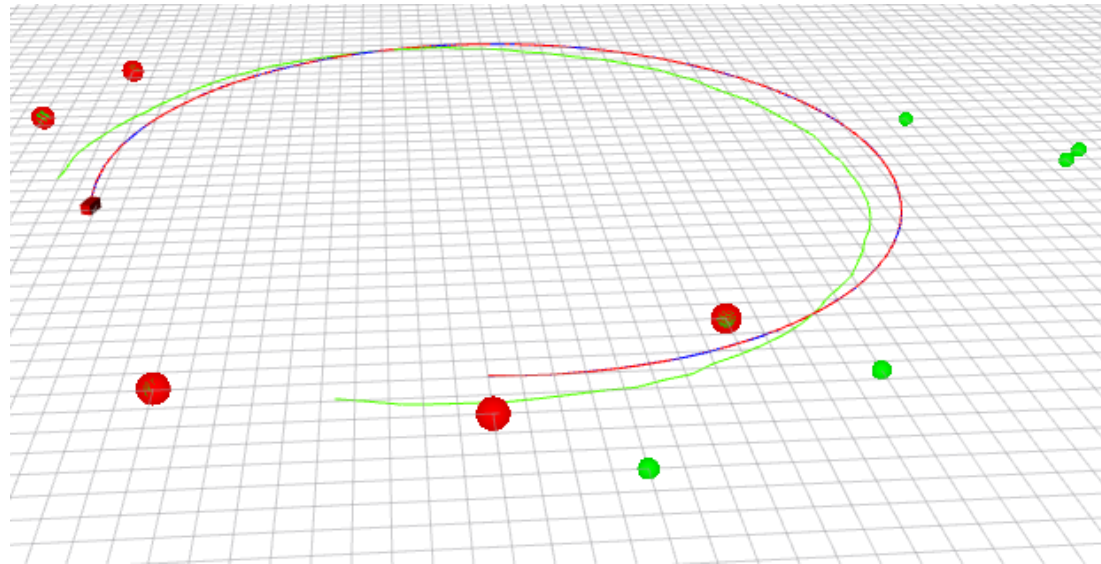
Error analysis

- **car_node.cpp** writes the localization result, the ground truth and the accumulated odometry into a **csv file**
- This file is read by a **Matlab script** that **plots the error** over time



Try it yourself!

- 1. Requirements
 - ROS
 - GTSAM
- 2. Download from [this repository](#)
- 3. Create a catkin workspace and copy the files into it
- 4. change filenames in car_lib.h
- 5. Invoke the following commands
 - catkin_make
 - source devel/setup.bash
 - roscore
 - rosrun car car_node
 - rosrun controller controller_node



Try it yourself!

- 6. For visualization, invoke `roslaunch rviz rviz` and make sure that rviz subscribes to the topics specified in `car/src/car_node.cpp`
- 7. Experiment with:
 - Odometry (commands and noise)
 - Landmarks (change map and measurement noise)
 - Maximum size of graph
- 8. Build your own projects!

