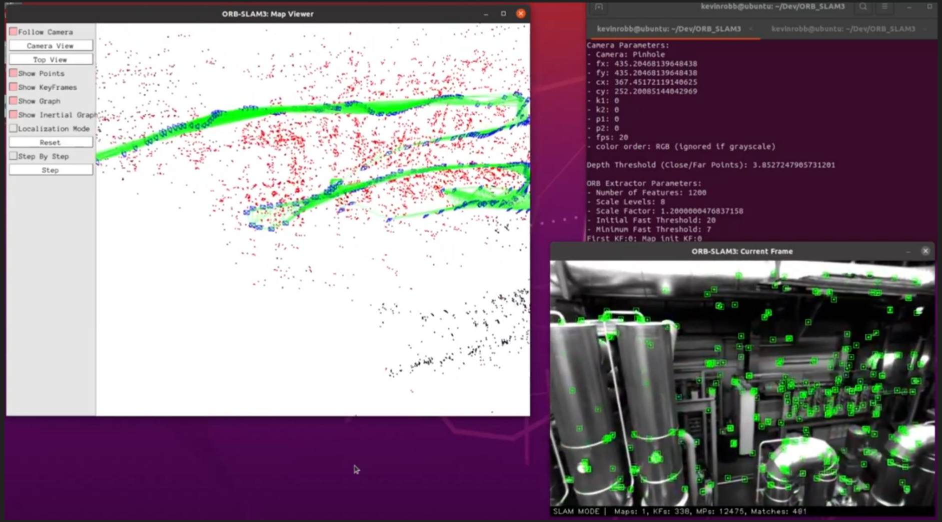
Stereo ORB-SLAM3 Implementation

Arun Anbu, Sasank Potluri, Kevin Robb, Tony Smoragiewicz

Presentation Summary

- Algorithm Overview
- Computational Costs
- Installation
- Data Collection
- Results/Demo
- Failure Modes
- Conclusions



```
Camera Parameters:
- Camera: Pinhole
- fx: 435.20468139648438
- fy: 435.20468139648438
- cx: 367.45172119140625
- cy: 252.20085144042969
- k1: 0
- k2: 0
- p1: 0
- p2: 0
- fps: 20
- color order: RGB (ignored if grayscale)

Depth Threshold (Close/Far Points): 3.8527247905731201

ORB Extractor Parameters:
- Number of Features: 1200
- Scale Levels: 8
- Scale Factor: 1.2000000476837158
- Initial Fast Threshold: 20
- Minimum Fast Threshold: 7
First KF:0; Map init KF:0
```

```
SLAM MODE |  Maps: 1, KFs: 338, MPs: 12475, Matches: 481
```

Screenshot of ORB_SLAM3 running on EuRoC example.

ORB-SLAM3 Overview

State-of-the-Art approach to the SLAM problem

“The first system able to perform visual-inertial and multi-map SLAM with monocular, stereo, and RGB-D cameras, using pinhole and fisheye lens models.”

Novelties:

- Maximum-a-Posteriori (MAP) estimation
- Multimap system with new map recognition
- 9 to 35 mm accuracies on public data sets

Universidad de Zaragoza - I3A Robotics, Perception, and Real-Time Group (RoPeRT)

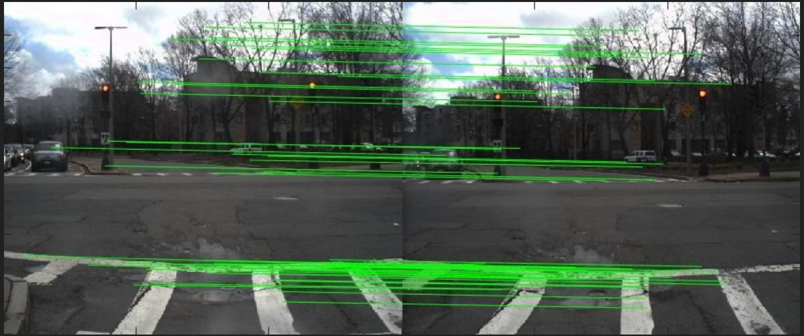
Carlos Campos & Richard Elvira along with Juan J. Gómez Rodríguez, José M.M. Motiel, and Juan D. Tardós

Visual Feature Detection and Description

Image pairs are recorded at 20Hz and 4MP resolution

Finds 1,200 ORB features per image

- Oriented FAST, rotated BRIEF
- Tracks scale and orientation
- Descriptor for data association


Matched features between stereo images in our dataset.

Keyframe-based approach

- Only the most representative frames are used
- Feature-sparse intermediate frames are discarded

Data Association

First SLAM algorithm to cover all time frames

- Short-term - features from last few seconds
- Mid-term - features with small accumulated drift (key to high accuracy)
- Long-term - allows for drift removal, map merging, and relocalization

Bag-of-Words image representation from feature descriptors

- DBoW2 library for use with ORB and BRIEF features
- "Implements a hierarchical tree for approximating nearest neighbours in the image feature space and creating a visual vocabulary"

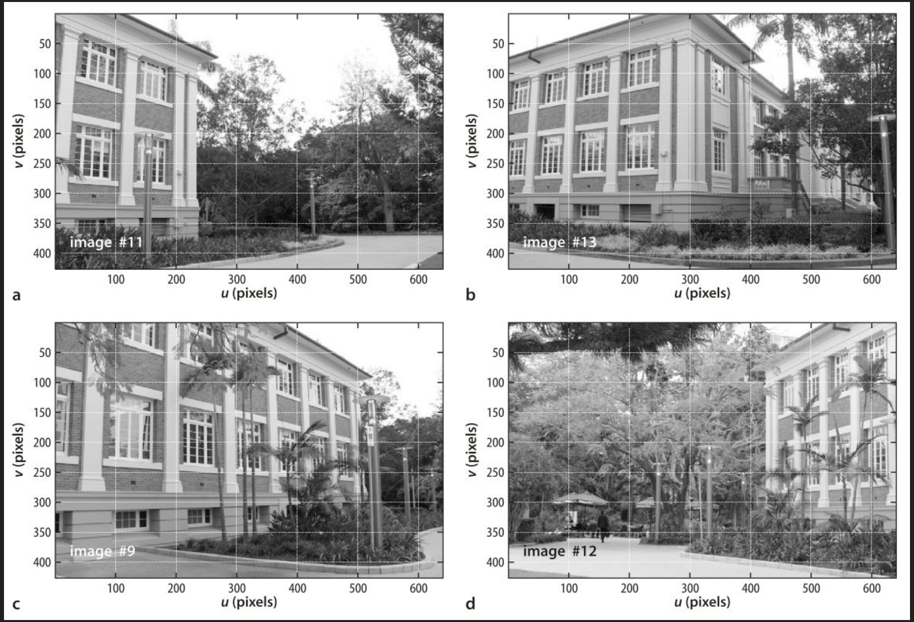
Bag-of-Words Example

Place to feature abstraction

Reduces search space

Feature similarity index compared to image #11 (a):

- 37% match for Image #13 (b)
- 34% match for Image #9 (c)
- 26% match for Image #12 (d)



Bundle Adjustment

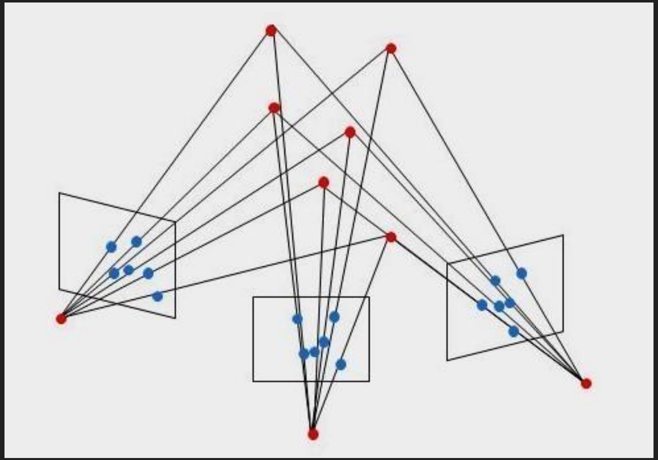
Minimize landmark and vehicle pose given uncertainty
Bayesian approach to model and parameter estimation

Local BA occurs with:

- Seven most recent keyframes

Full BA occurs after:

- Loop closure
- Map merging



Landmark estimation through multi-scene geometry.

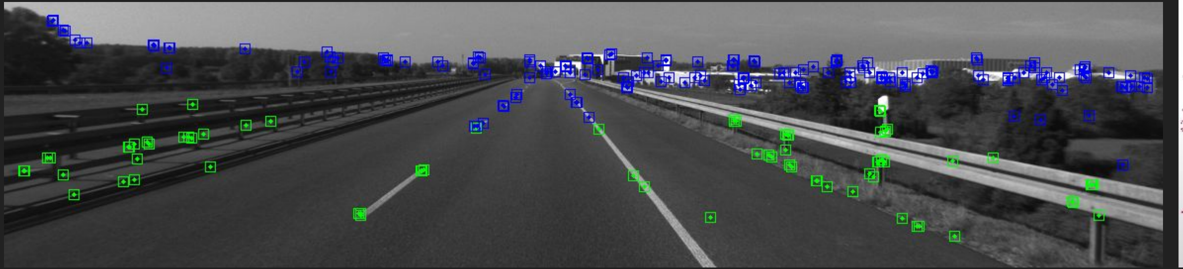
Tracking and Local Mapping

Tracking relies only on features and discards the rest of the image

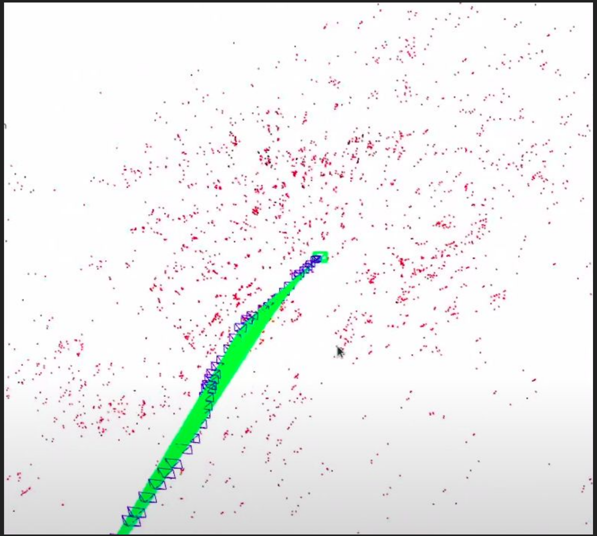
Calculate pose and new keyframe decision

Local mapping is done using Local BA

Keyframes are updated in DBoW2 library



Detected features corresponding to image in KITTI Dataset.
Only the green features are near enough to be used for tracking.



Snap from EuRoC example.

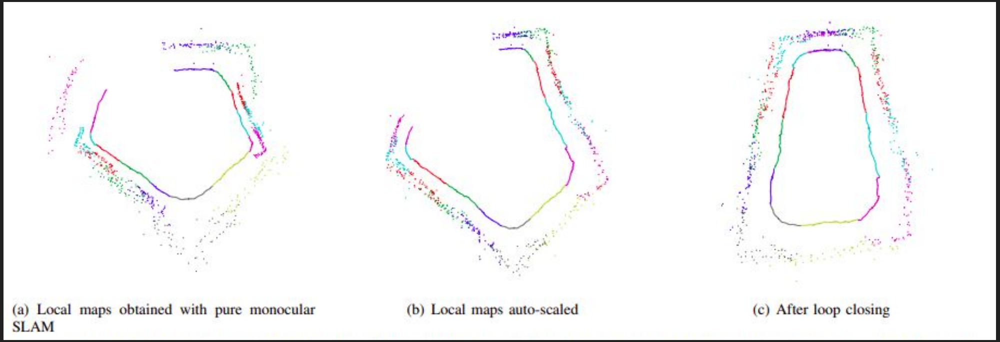
Loop Closing

Long-term data association

Same process as map merging

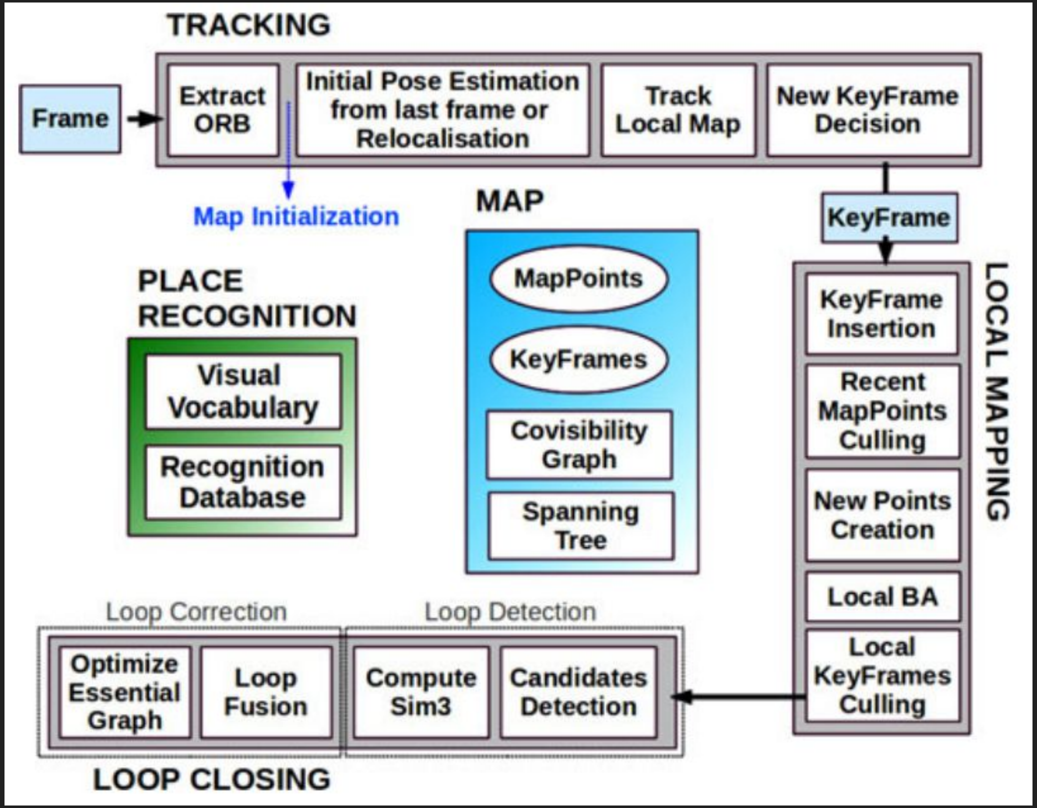
Global bundle adjustment helps to reduce long-term drift

Allows relocalization after location uncertainty grows too large



Example of loop closure detection from image to map closing for Monocular SLAM.

Overview



TRACKING
Frame
Extract ORB
Initial Pose Estimation from last frame or Relocalisation
Track Local Map
New KeyFrame Decision
Map Initialization
MAP
KeyFrame
PLACE RECOGNITION
Visual Vocabulary
Recognition Database
MapPoints
KeyFrames
Covisibility Graph
Spanning Tree
LOCAL MAPPING
KeyFrame Insertion
Recent MapPoints Culling
New Points Creation
Local BA
Local KeyFrames Culling
Loop Correction
Loop Detection
Optimize Essential Graph
Loop Fusion
Compute Sim3
Candidates Detection
LOOP CLOSING

Computational Costs

Original authors used 32GB of RAM to run their program.

Tracking or Localization ~33 ms

- ORB feature extraction ~15 ms
- Landmark tracking ~12 ms

Mapping ~195 ms

- Local bundle adjustment ~152 ms

Loop Closure ~1530 ms

- Full bundle adjustment ~1365 ms

Installation and Setup

Requirements:

- VM w/ Ubuntu 20.04, 16GB RAM, 4 cores
- OpenCV 3.2.0
- Python 2.7 w/ NumPy
- Pangolin
- Eigen3
- RealSense2
- ORB_SLAM3 repo

Full process is documented on the README in our GitHub repo:

https://github.com/kevin-robb/eece5554-vslam

We use old commit versions of ORB_SLAM3 and all dependencies to ensure compatibility.

Current version has many issues.

Requirements for Running ORB-SLAM3 in Stereo

Data format options:

- Folder of images + .txt file with timestamps, OR
- Rosbag with images (Modify subscribers' topics in ros_stereo.cc before building)

We wrote a ROS node to turn a rosbag into the first format.

Camera configuration .yaml, including:

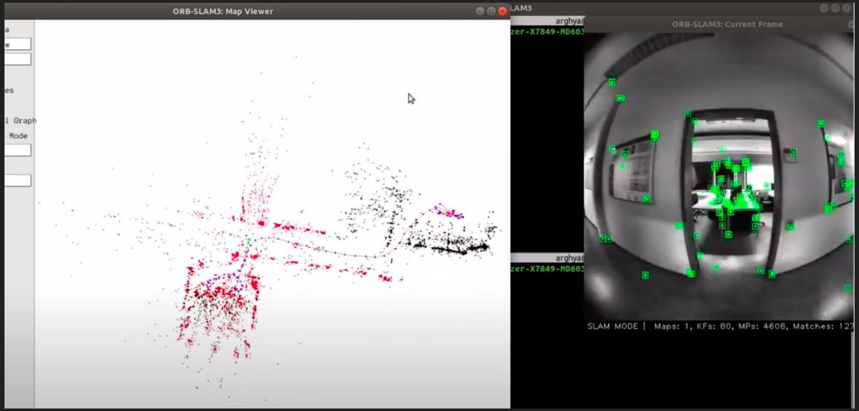
- Transformation between camera 1 and 2 coordinate frames
- Extrinsic parameters such as essential/fundamental matrices
- Intrinsic parameters of each camera
- ORB extractor & Viewer parameters

Expected Output from Running ORB-SLAM3

- Live view of the map as it is created.
- Live view of features detected on each image as it is processed.
- Frame and keyframe trajectory files written when execution ends, including:
    - Translation & quaternion of each frame relative to origin,

  where:

    - All frames are corrected for drift with loop closures.
    - First frame is re-centered on origin.

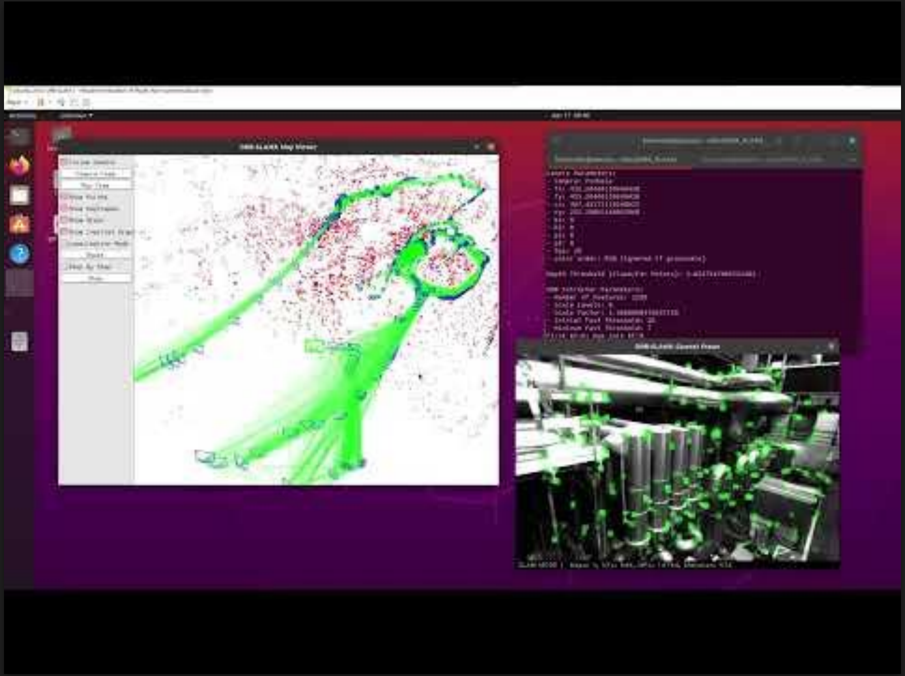


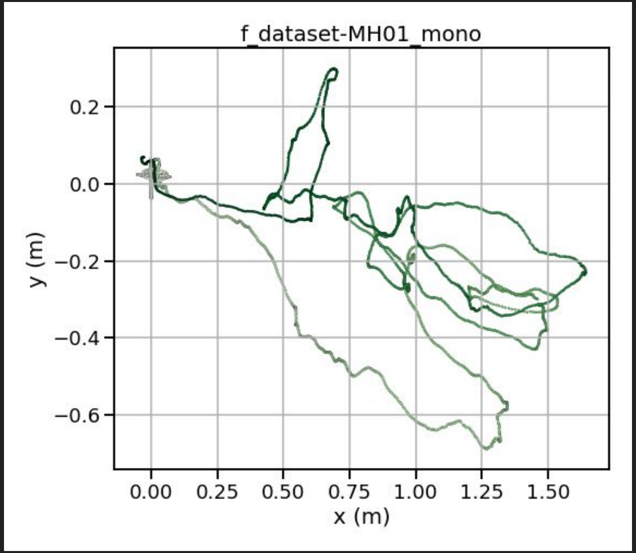
```
SLAM MODE |  Maps: 1, KFs: 80, MPs: 4608, Matches: 12
```

Example of ORB_SLAM3 running on TUM dataset by Arghya Chatterjee.

Working Demo with Sample Data

EuRoC examples are working on an Ubuntu 20.04 virtual machine running on a desktop PC with 16GB of RAM.





Final trajectory estimate. It's fairly accurate.

https://www.youtube.com/watch?v=mtbg8G4CSyw

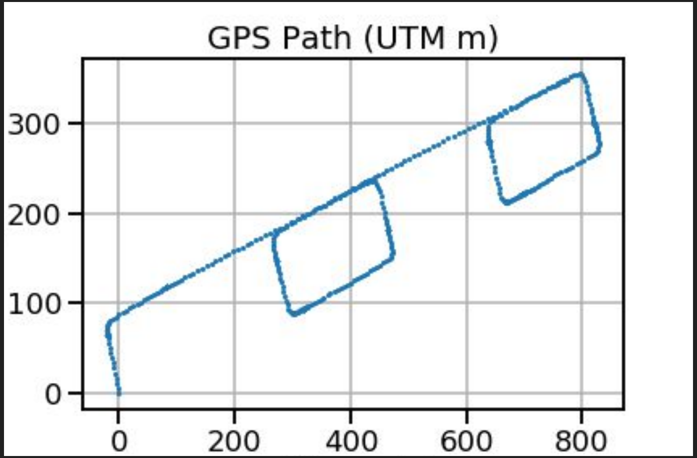
Data

We first tested with ORB-SLAM3's provided examples, such as EuRoC.

We're also using the *car_IR_RGB_lidar* dataset from ~10 minutes of the NUance car driving, and collected our own dataset with the car as well.

- Stereo vision with cam0 and cam1
- GPS for ground truth

For a static test, we also took data with a RealSense 435 camera in a small room.



Ground Truth trajectory from GPS measurements in the car_IR_RGB_lidar dataset.

Checklist for using NUance car

Before starting the car

- External hard drive with at least 100GB of storage
- One person to drive, and one person to sit in the backseat for data collection
- Startup the car and ensure all systems are turned on
- Start the systems publishing data
- Check physically if the required sensors are operational for data collection

While starting data collection

- Make sure the driver has a plan of the desired route
- Start Rosbag recording and echo the data to oversee their functionality
- Ensure you are recording only the things you need; LiDAR data is huge and unnecessary for the project

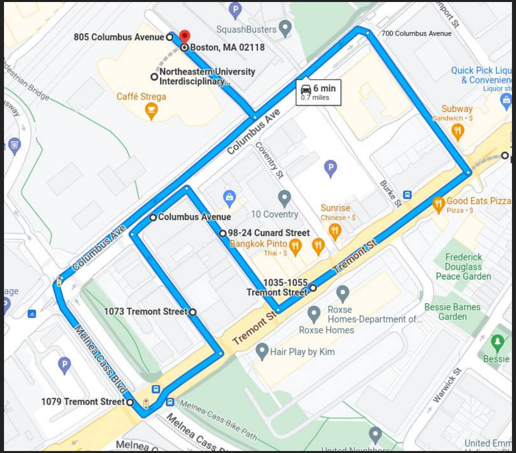
Data Collection

- We drove around on the roads near ISEC
- Completed two full loops
- Obtained distinguishable landmarks from the environment
- The camera lenses were dirty

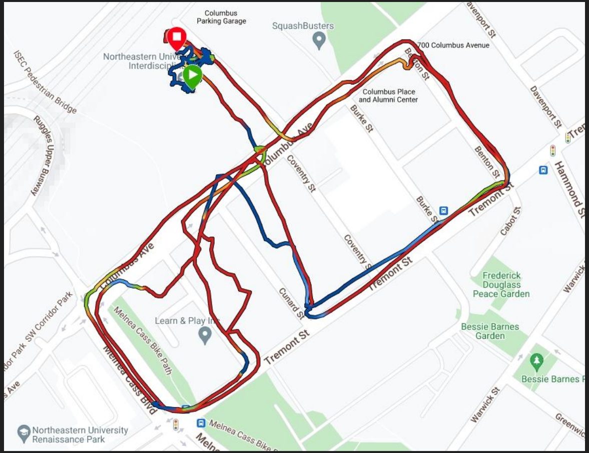
Our team about to take data with the NUance car.

Secondary Verification

- Attached a Garmin Forerunner 245 watch to the top of the car, since NUance did not have a working GPS sensor.
- Obtained a GPS plot of the path followed
- Used the GPS path for visual verification
- Drove two laps to ensure data collection had loop closure.



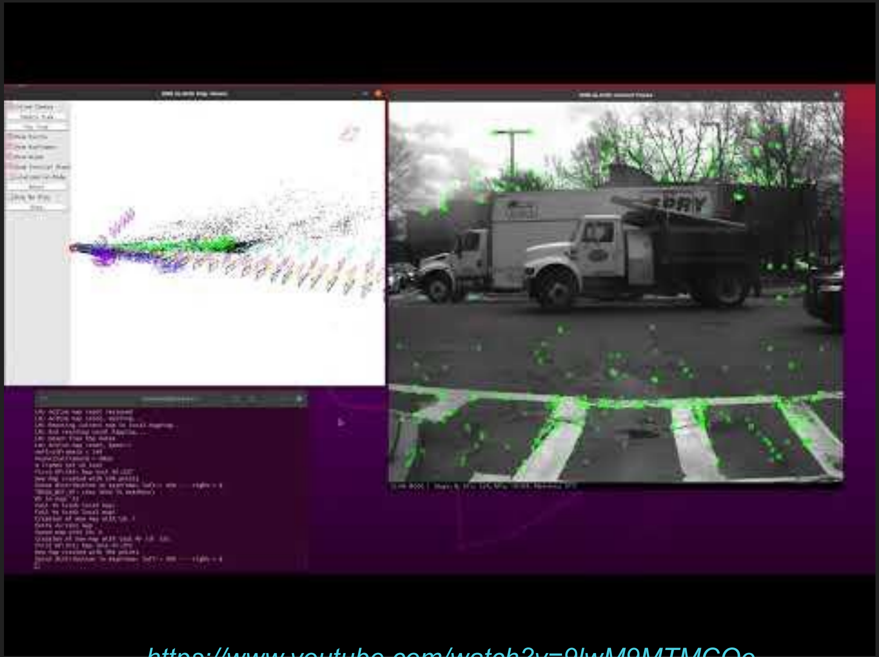
Planned path on Google Maps.



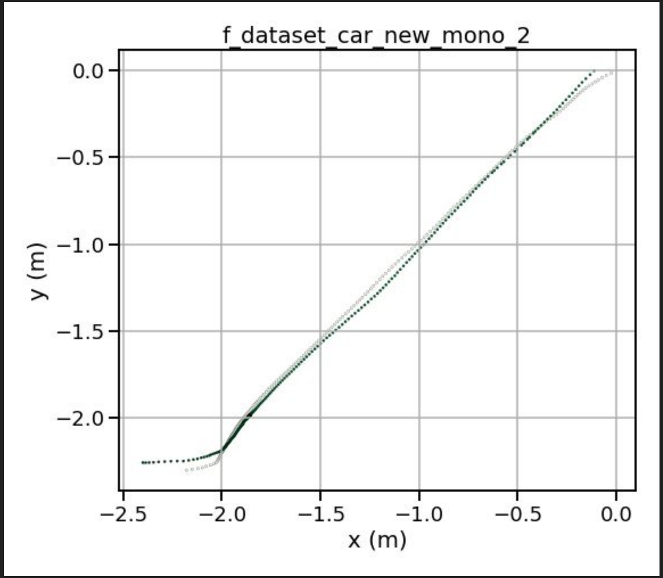
GPS data from the watch.

Demo of Poor Monocular Performance with Our Data

ORB SLAM runs for monocular NUance data, but perform poorly on turns. It results in many segmented maps that each end at a turn, resulting in a failed map.



https://www.youtube.com/watch?v=9lwM9MTMCQo



Final trajectory estimate. It's completely wrong, since the mapping failed during all turns.

Demo of Stereo Performance with Our Data

The camera calibration parameters provided in the /camera_info topics results in the following warped image, in which no features are detected by ORB SLAM.

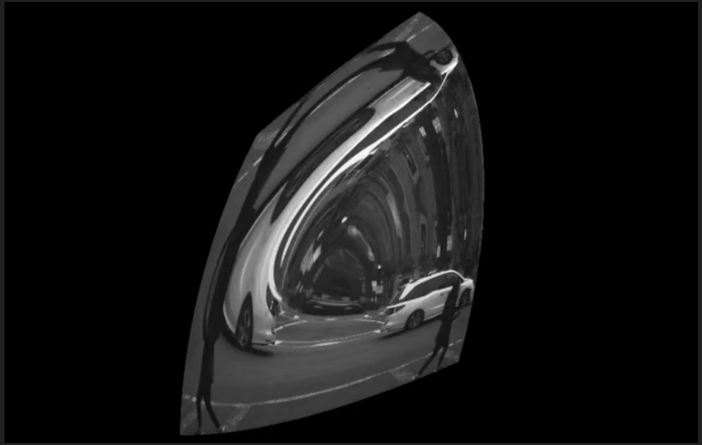


Image preview in ORB_SLAM3 with stereo from our dataset.

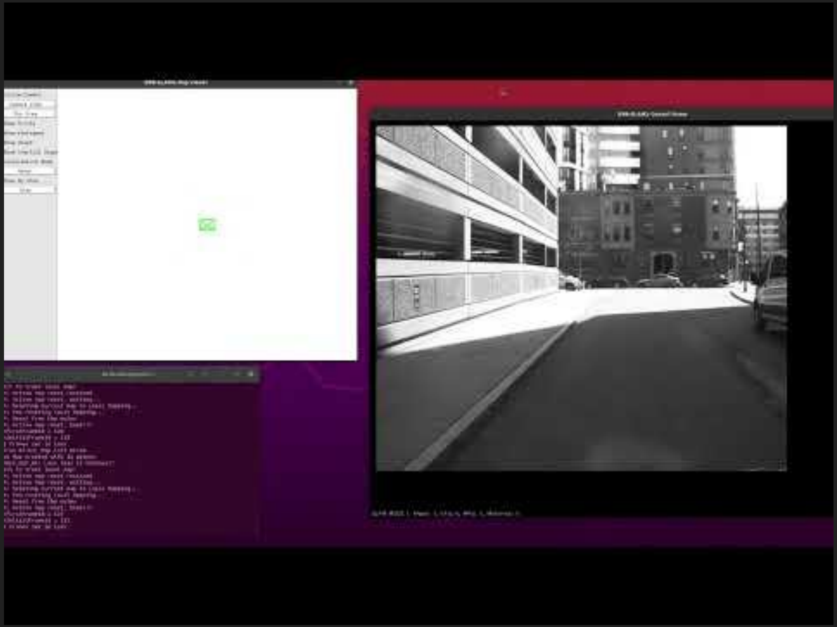
https://www.youtube.com/watch?v=-YRJ--opacs

When we use the .yaml provided as part of the EuRoC sample, it is able to actually run and create a map, although the images appear very zoomed in and distorted, since it is for a completely different camera setup.

By pulling aspects from the EuRoC .yaml into ours, we can find a balance that allows ORB SLAM to run with our data in stereo.

Demo of Stereo Performance with Our Data

Manually tweaking the .yaml to get ORB SLAM to run (since using the truly correct values according to the rosbag did not work at all) yields okay performance.

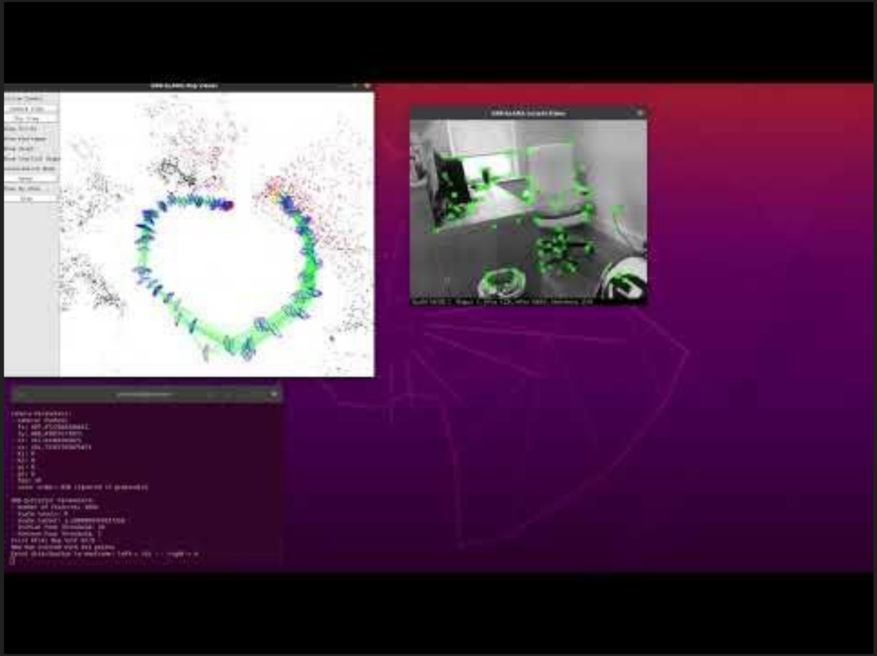


https://www.youtube.com/watch?v=vhw_BLc02v4

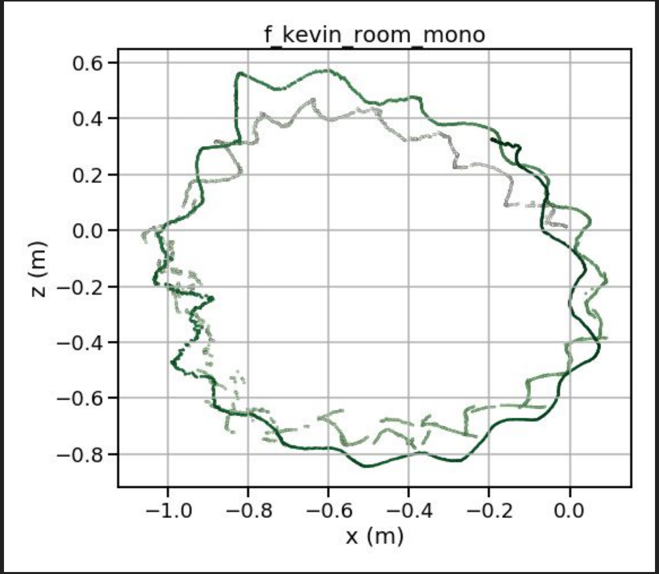
There is a “flickering” where detected features disappear and reappear often, rather than persisting. This makes it very difficult to get a map initialized.

Demo of Good Monocular Performance in Static Environment

To verify that the dynamic environment and quick turns are indeed the cause of our issues with the NUance car datasets, we ran monocular ORB SLAM on an imageset gathered with a RealSense 435 camera walking around a bedroom.



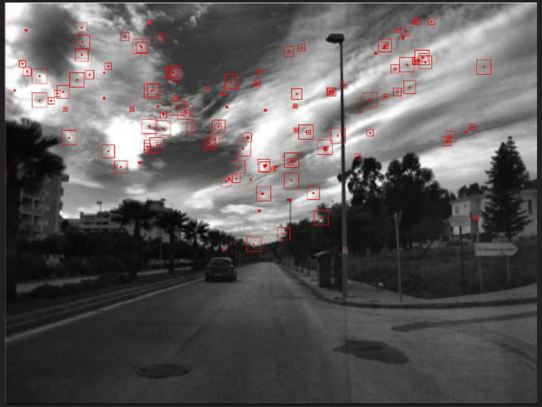
https://www.youtube.com/watch?v=3rCco0wBkBU



Final trajectory estimate. It appears accurate.

Failure Modes

- Indistinctive paths with few features
- Slow moving clouds introduce bias
- Dirty camera lenses worsen matching
- High computational requirement
- Extreme lighting conditions
- Maps can be very sparse
- Dynamic environments
- Loop Closure


Clouds detected as features.

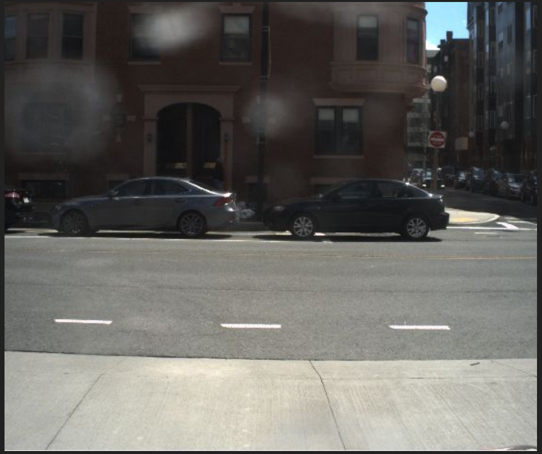
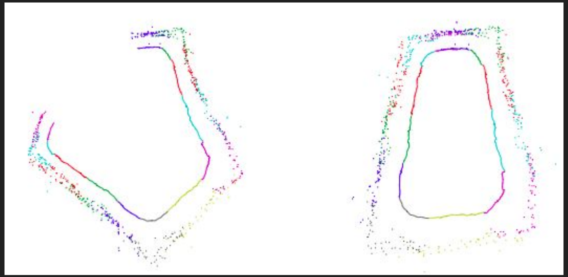


Image frame of cam0. Dirty spots are visible.



Ideas for improvement

Dirty sensors are a common problem. Some industry approaches are:

- Custom hardware to clean camera lenses when dirty
- Generative Adversarial Network to predict what's behind dirt and water spots.

Dynamic scenes are another problem. Some fixes include:

- Car and person detection to block dynamic objects
- Sky segmentation so clouds aren't tracked. Removing the top half of image is non-ideal when tall structures could be good features.
- Improved block factorization and outlier rejection

Front and rear views have very different features.

- People even suffer from this problem when hiking in the woods.
- Possible solution is two sets of stereo cameras to create maps.
- This would make "out and back" loops more robust to closure.

Camera washer from BMW.

Future Investigations

- Map relocalization from different scenes such as:
  - Night and day (lighting)
  - Summer and Winter (color and features)
- Loop closure detection for “out and back” paths
- Improved functionality with stereo camera pairs
  - Front and rear from loop closure
  - Left and right for improved turn mapping
- Dynamic object tracking to remove outliers

Conclusions

- ORB_SLAM3 is hard to use! (official README is out of date)
- Although it is algorithmically state of the art, in practice it is very difficult to use, even for an offline application; it’s slow and finicky.
- Very low resolution map. More useful for localization.
- ORB_SLAM3 works fine with their sample data, and with static indoor environments with slow camera movement.
- ORB_SLAM3 performs poorly with car data, due in large part to the highly dynamic environment.
- Turns in any of the car data completely breaks the map.

References

ORB SLAM publication: https://arxiv.org/abs/2007.11898

ORB SLAM 3 repo: https://github.com/UZ-SLAMLab/ORB_SLAM3

DBoW2: https://github.com/dorian3d/DBoW2

Bag-of-Words: Robot Vision and Control - Corke 2017

Installation guides:

- https://medium.com/@tristan.sch/setting-up-a-virtual-machine-with-orb-slam-3-1a12e7905cf5
- https://www.youtube.com/watch?v=DxqzwBQVCNw&t=0s