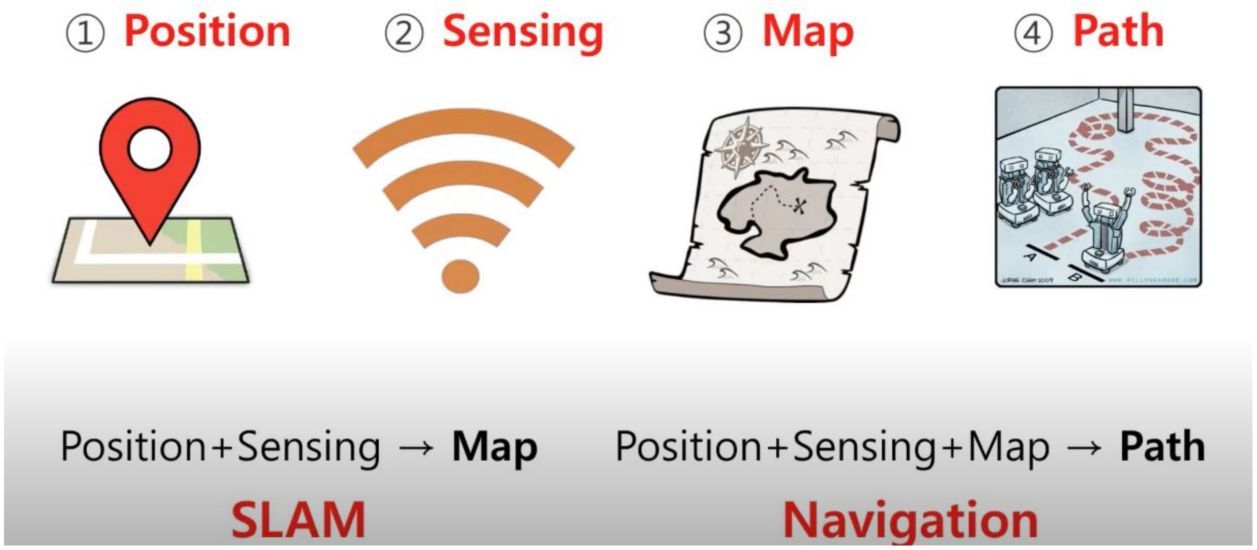
Great navigation library in python: https://github.com/rpng/open_vins
Great Map standalone application: http://introlab.github.io/rtabmap/
Great professors:
https://www.csail.mit.edu/person/john-leonard
http://karaman.mit.edu/
http://robots.stanford.edu/

Resources:
Visual Feature by Cyrill Stachniss
http://ais.informatik.uni-freiburg.de/teaching/ws13/mapping/
Probabilistic Robotics by Thrun is a great book
Data-Driven-Science-Engineering-Learning-Dynamical by Brunton (http://databookuw.com/)

SLAM and Navigation



Computer Vision Fundamentals for SLAM

Features:

- **Keypoints**: Useful information in an image represented with a position (visualized with circle/ellipse)
- **Descriptors**: Are used to describe keypoints usually using a vector format (of constant length).
  - Are independent of keypoint position
  - Robust to image transformation
  - Scale independent
  - Descriptors of 2 or more images can be compared with something as simple as Euclidean distance

Algorithms used to extract keypoints and descriptors:

- SURF (Speeded-up robust features)
  - Faster than SIFT
- SIFT
- ORB
- FAST algorithm for corner detection

Feature Matching:

- **Brute-Force matcher**

Brute-Force matcher is simple. It takes the descriptor of one feature in first set and is matched with all other features in second set using some distance calculation. And the closest one is returned.
In case of SLAM it checks 2 consequent frames with each other.
bf = cv2.BFMatcher(normType, crossCheck)
matches = bf.knnMatch(f1.des, f2.des, k=2)

First parameter is **normType**. It specifies the distance measurement to be used. By default, it is cv2.NORM_L2. It is good for SIFT, SURF etc (cv2.NORM_L1 is also there). For binary string based descriptors like ORB, BRIEF, BRISK etc, cv2.NORM_HAMMING should be used, which uses Hamming distance as measurement.
We noticed that with Local deep features the best distance measurement type is: ??
Second parameter is **crossCheck**. If it is true, Matcher returns only those matches with value (i,j) such that i-th descriptor in set A has j-th descriptor in set B as the best match and vice-versa. That is, the two features in both sets should match each other. It provides consistent result, and is a good alternative to ratio test proposed by D.Lowe in SIFT paper.

- FLANN Matcher is another one

Visual SLAM related terms:

- visual-inertial odometry,
- LIDAR point cloud processing (advanced)
- sensor/camera calibration
- Pose estimation
- Depth map from Stereo images (stereo version of ORB-SLAM2)
- Epipolar Geometry

What is Vision:
David Marr says that vision is the process of discovering from images what is present in the world and where it is. This is fundamental to SLAM based on Vision or Visual SLAM.

Kalman Filter:
https://www.youtube.com/watch?v=Lgq4R-F8SX8
https://www.intechopen.com/books/introduction-and-implementations-of-the-kalman-filter/introduction-to-kalman-filter-and-its-applications
Probability density function
Gaussian distributions
Random variables
Bayes Filter (is the base)

The Kalman filter algorithm consists of two stages: prediction and update. Note that the terms “prediction” and “update” are often called “propagation” and “correction,” respectively, in different literature. The Kalman filter algorithm is summarized as follows.

SLAM types:

- Feature Based
    - ORB-SLAM
    - ORB-SLAM2

ORB-SLAM is a very well known and popular SLAM algorithm that uses ORB feature detector. ORB-features consist of two algorithms FAST and BRIEF.
It is a monocular SLAM that can be used with only 1 camera as input. We are going to use the same database to compare both SLAM applications. KITTI is a dataset gathered by driving a car around an urban environment and ground truth is provided by a Velodyne lidar and RTK-GPS. It is meant for use in research in the computer vision and autonomous driving field and it was used to benchmark both ORB-SLAM and ORB-SLAM2 against other SLAM systems. The current state of the art uses expensive sensors like lidar to gather data for the SLAM system

No doubt that errors resulted by drift in pose estimation and map evaluation keep accumulating

ORB-SLAM2 is an improvement over ORB-SLAM to work with RGB-D camera or stereo camera instead of a monocular camera.The structure contains three threads that work in parallel, the tracking, local mapping, and loop closure which is visualized in Figure 1.1

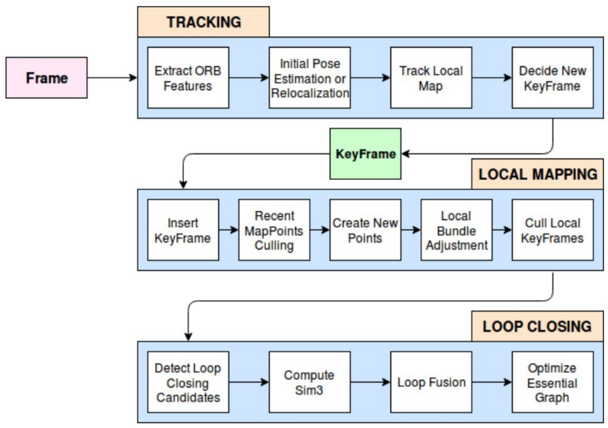


Figure 1.1 Algorithm structure for ORB-SLAM

For stereo cameras ORB features are extracted from both the left and right images, the left image is used as a reference image for feature matching and its ORB-features are then matched against the ORB-features of the right image. This results in ORB-features with depth-information enabling direct 3D-mapping without triangulation

Tracking
This thread does deep local feature detection and matching to the local map.

- **Extract Local Deep Features (TFNet) or ORB features (ORB-SLAM2)**

Getting the best features from images that can be used further.

    - **ORB explained:**

ORB was built at OpenCV labs. It performs as well as SIFT on the task of feature detection (and is better than SURF) while being almost two orders of magnitude faster. ORB builds on the well-known FAST keypoint detector and the BRIEF descriptor. Both of these techniques are attractive because of their good performance and low cost.

ORB's main contributions are as follows:

- The addition of a fast and accurate orientation component to FAST
- The efficient computation of oriented BRIEF features
- Analysis of variance and correlation of oriented BRIEF features
- A learning method for decorrelating BRIEF features under rotational invariance, leading to better performance in nearest-neighbor applications

- **TFeat explained (based on paper)**
    - **Shallow neural network**

Our motivation for such a shallow network is to develop a de- scriptor for practical applications including those requiring real time processing. This is a challenging goal given that all previously introduced descriptors are computationally very intensive, thus impractical for most applications. We demonstrate that excellent descriptor performance can be obtained with a shallow network thus avoiding computationally complex architectures and expensive mini-batch hard negative mining. Using a shallow network helps us to match the performance of ORB which in itself is really quick and vastly used for real time applications.

- **Triplets of examples improvements**

- **Pose estimation** here

The problem of determining the position and orientation of the camera relative to the object (or vice-versa). Usually before estimating the position of the camera we do a **camera calibration** here.

In camera calibration we estimate values like the intrinsic and extrinsic parameters of the camera. Intrinsic parameters are specific to a camera. They include information like focal length and optical centers.The focal length and optical centers can be used to create a camera matrix, which can be used to remove distortion due to the lenses of a specific camera.

Extrinsic parameters correspond to rotation and translation vectors which translates coordinates of a 3D point to a coordinate system.

Given a pattern image, we can utilize the above information to calculate its pose, or how the object is situated in space, like how it is rotated, how it is displaced. During pose estimation we use camera matrix and distortion coefficients that we get from calibration phase.

Local mapping

Manages local map and performs local bundle adjustments. Usually in bundle adjustment we do refinements to the trajectory in order to have better camera pose.

Loop closing

Detects loop closing to avoid map duplication and corrects and accumulated drifts. For this purpose we use A visual bag of words. **Bag of words** are used in order to recognize already visited places a hierarchical bag of words structure is created based on an image database. To achieve a fast and manageable feature matching in the big image datasets, the bag of words method uses a visual vocabulary to represent image features.

- **Sim(3)**

To detect **loop closures** and **scale-drift**, a similarity transform $\xi \in$ sim(3) is estimated using scale-aware, direct sim(3)-image alignment. sim(3) is a Lie group used for DoF (degrees of freedom).

Visual odometry

Getting the odometry of the camera is really important. We are going to use visual odometry which uses a camera instead of other techniques that use IMU.
Visual odometry is very different depending on if you have a stereo or monocular camera.
Pose: estimating not only the distance traveled, but the entire trajectory of a moving robot. The job is to determine each object's position and orientation relative to some coordinate system.The objective is to construct a 6-DOF trajectory using the video stream coming from this camera
Usually monocular visual odometry works best with really small robots and thats a key focus in the area, but with lets say with autonomous vehicles, ships and so on visual odometry gives better results since more data is available.

The advantage of stereo visual odometry is that you can estimate the exact trajectory, while in monocular you can only estimate the trajectory

What is Visual Odometry?

- Estimating the motion of a camera in real time using sequential images (i.e., egomotion)
- The idea was first introduced for planetary rovers operating on Mars based on Moravec 1980 phd paper.

VO Pipeline

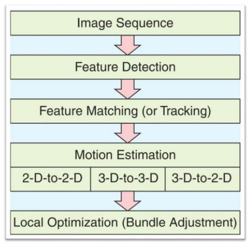


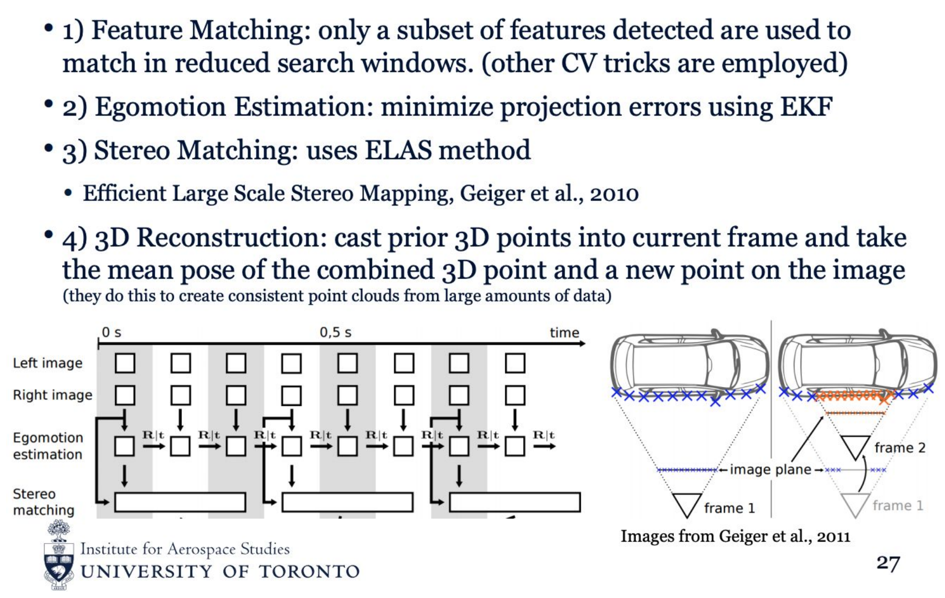
Image from Scaramuzza and Fraundorfer, 2011

Without loop closure the estimate still drifts, that's why we need loop closure in SLAM.
Visual odometry is really important for SLAM. Odometry affects localization which in turn affects SLAM. Sometimes Visual odometry and Visual SLAM are used interchangeably but SLAM produces a map of features and Visual Odometry focuses on camera trajectory.

NOTE: !opencv/demo example with visual odometry (http://avisingh599.github.io/vision/monocular-vo/)

https://www.youtube.com/watch?v=CwWYwqMHCD4 (Visual odometry and SLAM)

Dense 3D Reconstruction in Real-Time



- 1) Feature Matching: only a subset of features detected are used to match in reduced search windows. (other CV tricks are employed)
- 2) Egomotion Estimation: minimize projection errors using EKF
- 3) Stereo Matching: uses ELAS method
  - Efficient Large Scale Stereo Mapping, Geiger et al., 2010
- 4) 3D Reconstruction: cast prior 3D points into current frame and take the mean pose of the combined 3D point and a new point on the image
  (they do this to create consistent point clouds from large amounts of data)

Images from Geiger et al., 2011

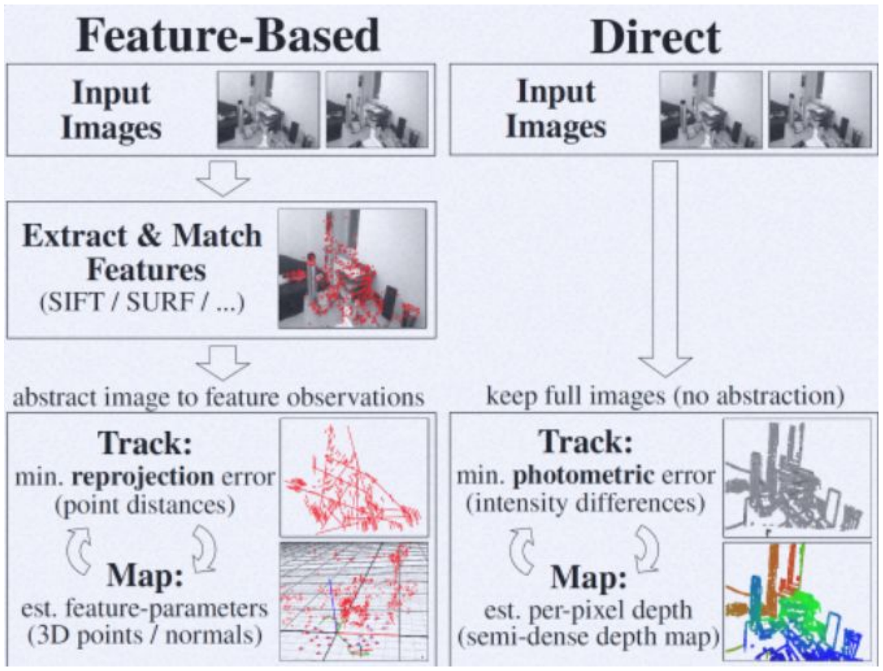
Institute for Aerospace Studies
UNIVERSITY OF TORONTO

27

- Graph based
- Visual
  - The map is built with only one information and those are images.
- Lidar Based SLAM

Useful library to visualize: https://github.com/simondlevy/PyRoboViz
Evaluation SLAM:
https://github.com/MichaelGrupp/evo

Feature based vs Direct SLAM


Feature-Based
Direct
Input Images
Input Images
Extract & Match Features (SIFT / SURF / ...)
abstract image to feature observations
keep full images (no abstraction)
Track: min. reprojection error (point distances)
Map: est. feature-parameters (3D points / normals)
Track: min. photometric error (intensity differences)
Map: est. per-pixel depth (semi-dense depth map)

ORB-SLAM2

Initializing the map

To initialize the map starting by computing the relative pose between two scenes, they compute two geometrical models in parallel, one for a planar scene, a homography and one for non-planar scenes, a fundamental matrix. They then choose one of both based on a relative score of both. Using the selected model they estimate multiple motion hypotheses and see if one is significantly better than the others, if so, a full bundle adjustment is done, otherwise the initialization starts over.

Tracking

The tracking part localizes the camera and decides when to insert a new keyframe. Features are matched with the previous frame and the pose is optimized using motion-only bundle adjustment. The features extracted are FAST corners. (for res. till 752x480, 1000 corners should be good, for higher (KITTI 1241x376) 2000 corners works). Multiple scale-levels (factor 1.2) are used and each level is divided into a grid in which 5 corners per cell are attempted to be extracted. These FAST corners are then described using ORB. The initial pose is estimated using a constant velocity motion model. If the tracking is lost, the place recognition module kicks in and tries to re-localize itself. When there is an estimation of the pose and feature matches, the co-visibility graph of keyframes, that is maintained by the system, is used to get a local visible map. This local map consists of keyframes that share map points with the current frame, the neighbors of these keyframes and a reference keyframe which share the most map points with the current frame. Through re-projection, matches of the local map are searched on the frame and the camera pose is optimized using these matches. Finally is decided if a new Keyframe needs to be created, new keyframes are inserted very frequently to make tracking more robust. A new keyframe is created when at least 20 frames has passed from the last keyframe, and last global re-localization, the frame tracks at least 50 points of which less then 90% are point from the reference keyframe.

Local mapping

First the new keyframe is inserted into the covisibility graph, the spanning tree linking a keyframe to the keyframe with the most points in common, and a 'bag of words' representation of the keyframe (used for data association for triangulating new points) is created.
New map points are created by triangulating ORB from connected keyframes in the covisibility graph. The unmatched ORB in a keyframe is compared with other unmatched ORB in other keyframes. The match must fulfill the epipolare constraint to be valid. To be a match, the ORB pairs are triangulated and checked if in both frames they have a positive depth, and the parallax, re projection error and scale consistency is checked. Then the match is projected to other connected keyframes to check if it is also in these.
The new map points first need to go through a test to increase the likelihood of these map points being valid. They need to be found in more than 25 % of the frames in which it is predicted to be visible and it must be observed by at least three keyframes.
Then through local bundle adjustment, the current keyframe, all keyframes connected to it through the co-visibility graph and all the map points seen by these keyframes are optimized using the keyframes that do see the map points but are not connected to the current keyframe.
Finally keyframes that are abundant are discarded to remain a certain simplicity. Keyframes from which more than 90 % of the map points can be seen by three other keyframes in the same scale-level are discarded.

Loop closing

To detect possible loops, they check the bag of words vectors of the current keyframe and its neighbors in the covisibility graph. The min. similarity of these bag of words vectors is taken as a benchmark and from all the keyframes with a bag of words similarly to the current key frame that is greater that this benchmark, all the keyframes that are already connected to the current keyframe are removed. If three loop candidates that are consistent are detected consecutively, this loop is regarded as a serious candidate.
For these loops, the similarity transformation is calculated (7DOF, 3 trans, 3 rot, 1 scale) RANSAC iterations are performed to find them and these are then optimized after which more correspondences are searched and then again an optimization is performed. If the similarity is supported by having enough inlier's, the loop is accepted.
The current keyframe pose is then adjusted and this is propagated to its neighbors and the corresponding map-points are fused. Finally a pose graph optimization is performed over the essential graph to take out the loop closure created errors along the graph. This also corrects for scale drift.

DF-SLAM

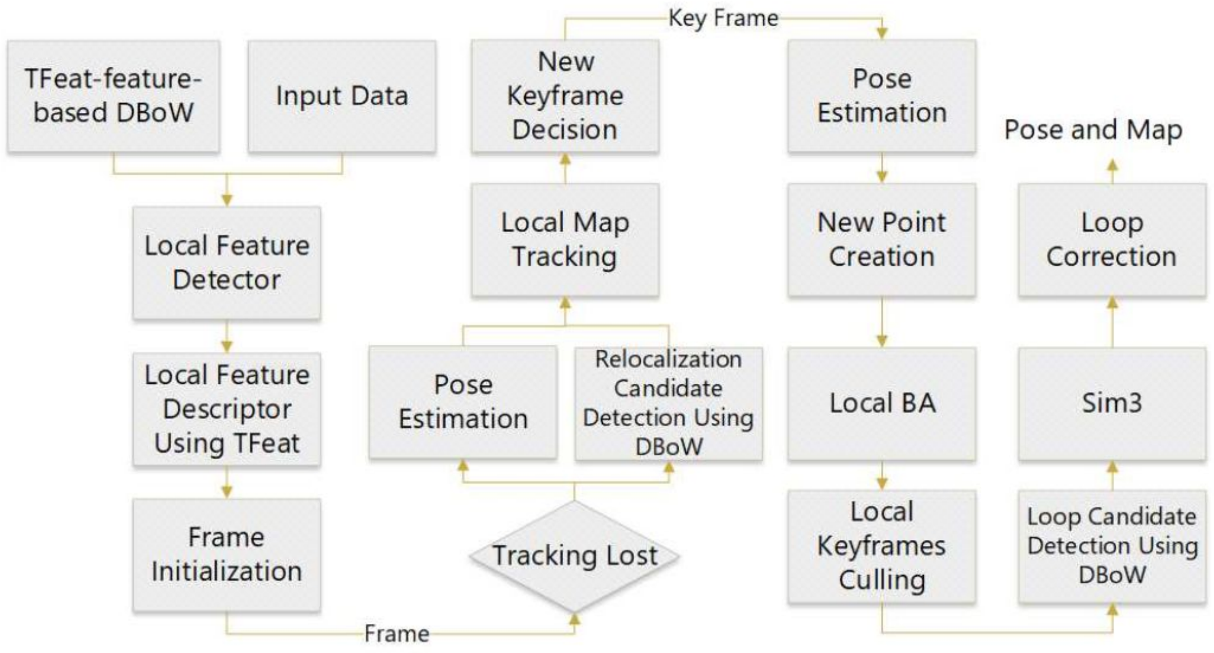
Introduction

Deep learning has great accuracy when it comes to classification and feature matching, thus it can be a great choice for data association. Researchers believe that the bottleneck for data association can be bypassed by the help of neural networks. A lot of SLAM systems in the past that used deep learning were built mainly to show the advantage of neural networks used in SLAM, but they usually sacrifice accuracy for efficiency. These systems are trained with particular datasets and in theory result in very high accuracy but can't adapt to other environments or are too slow to be used in production and real time systems.

DF SLAM is really simple and portable. We are able to replace traditional hand-crafted descriptors with local deep features. We are not only able to use it directly in all other SLAM-systems but we can also use it for other computer vision tasks such as camera calibration or SfM (structure from motion). From the results based on the paper they notice an increase in performance and accuracy. The shallow network gives the ability of the DF SLAM to operate in “near real time”. Siamese (such as matchnet and deepcompare) and triplet networks turn out to be the main architectures employed in local feature descriptor tasks.

How does it work ?

Tracking takes charge of constructing data associations between adjacent frames using visual feature matching. Afterward, it initializes frames with the help of data associ- ations and estimates the localization of the camera using the polar geometric constraint. It also decides whether new keyframes are needed. If lost, global relocalization is per- formed based on the same sort of features. Local Map- ping will be operated regularly to optimize camera poses and map points. It receives information constructed by the tracking thread and reconstructs a partial 3D map. If loops are detected, the Loop Closure thread will take turns to op- timize the whole graph and close the loop. The frame with a high matching score is selected as a candidate loop closing frame, which is used to complete loop closing and global optimization. None of these modules accept raw images as inputs to reduce space consumption.



Step 0: Preprocessing

0.1 Create datasets to train the model

We need to build patch datasets the same way as ORB-SLAM to make sure that the network is efficient and make proper comparisons. The patches have to be extracted from HPatches. In general the most popular way to extract local features in patch based datasets is DoG, but for SLAM systems the most popular one is using FAST detector.

0.2 Construct the visual vocabulary

We start the training procedure for visual vocabulary after having created the dataset and trained the model. We train BoW on the dataset of our choosing and choose **1e6 as the number of leaves in the vocabulary tree.** Since our descriptor is normalized float vector 128-D also our leaf nodes are normalized.

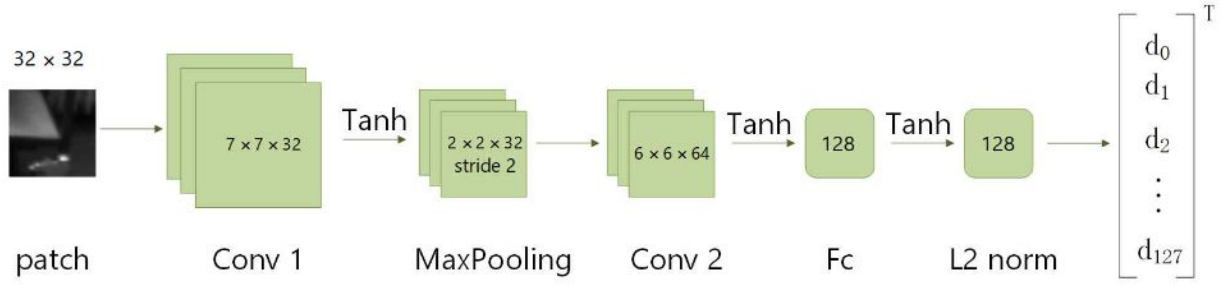
Step 1: Extract the most important points/features in the image using TFeat

We generate a normalized descriptor with float scale 128-D. We don't apply gaussian blur like in hand-made features, but we take raw patches of images instead.

We use an evenly distributed FAST detector to build the training dataset. All training is done using pytorch and stochastic gradient descent solver with the learning rate of 0.01, the momentum of 0.9

and weight decay of 0.0001. We also use typical data augmentation techniques, such as random rotation and crop, to improve the robustness of our network.

TFeat in details (based on paper):



Using triplet network

Recent work in **[12]** shows that learning representations with triplets of examples, gives much better results than learning with pairs using the same network. Inspired by this, we focus on learning feature descriptors based on triplets of patches.

Learning with triplets involves training from samples of the form {*a*, *p*, *n*}, where *a* is the *anchor*, *p positive*, which is a different sample of the same class as *a*, and *n negative* is a sample belonging to a different class. In our case, *a* and *p* are different viewpoints of the same physical point, and *n* comes from a different keypoint. Furthermore, optimising the parameters of the network brings *a* and *p* close in the feature space, and pushes *a* and *n* far apart.

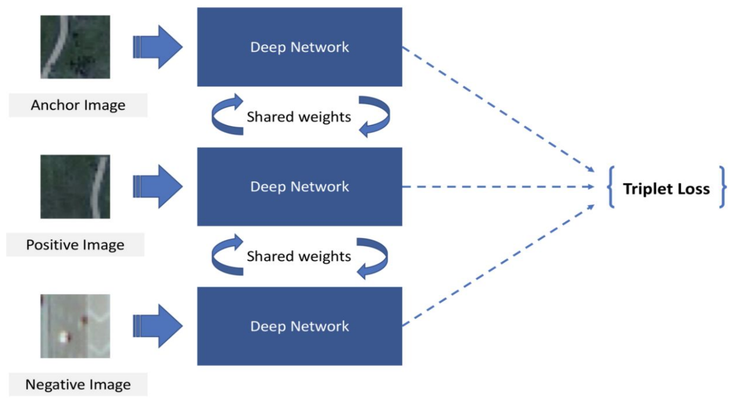
We can categorise the loss functions that have been proposed in the literature for learning convolutional embeddings with triplets into two groups, the *ranking-based losses* and the *ratio-based losses.*

It is worth noting that the CNN used in our experiments consists of only two convolu- tional layers, while all of the other state-of-the art deep feature descriptors consist of four or more layers

We build a simple hierarchical network that is based on 100 convolutional filters, followed by a linear transformation that projects the responses of the filters to the desired output dimensionality.

In fact, when running on GPU, we reach speeds of 10μ*s* per patch which is comparable with the CPU speeds of the fast binary descriptors[4]. This is a significant advantage over the previously proposed descriptors and makes CNN based descriptors applicable to practical problems with large datasets

We also demonstrate that ratio-loss based methods are more suitable for patch pair classification, and margin-loss based methods work better in nearest neighbour matching applications. This indicates that a good performance on patch classification does not necessarily generalise to a good performance in nearest neighbour based frameworks

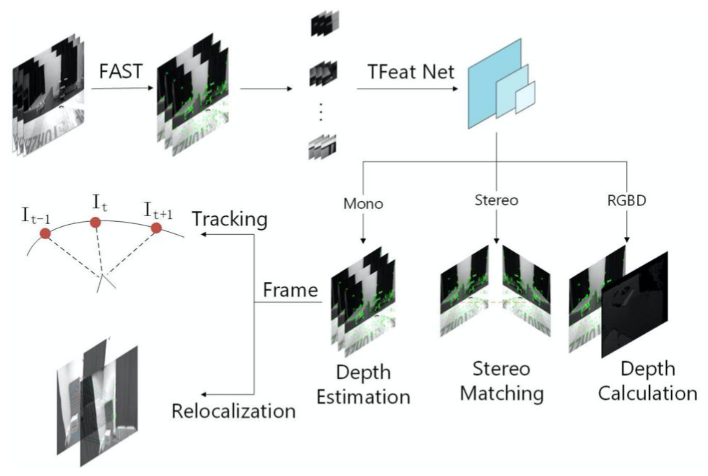


Step 2: Localization based on BoW(Bag of words)
After we extract the features for every frame we pass them to tracking,mapping and loop closing. The localization is performed using D BoW (Bag of words) same as ORB-SLAM. The similarity between two consecutive frames is done using Euclidean distance. *Relocalization and loop closing modules rely heavily on the local feature descriptors.*

Step 3: Improve localization using Visual Vocabulary

The speed of the algorithm and our system is really important if we want it to be used in real time applications. That's why Visual vocabulary is introduced. *TODO: explain visual vocabulary*

We trained the vocabulary using the feature descriptors extracted by TFeat. This way we could assign a word vector and feature vector for each frame, and calculate their similarity easier and faster.



Implementation

Requirements:

- Ubuntu 20.04
- OpenCV 3.4.2
- Pangolin (latest)
- ORB_SLAM2 (latest version from forked repo)

Installation script:

./Examples/RGB-D/rgbd_tum Vocabulary/ORBvoc.txt Examples/RGB-D/TUM1.yaml Examples/RGB-D/rgbd_dataset_freiburg1_desk/ Examples/RGB-D/associations.txt

Compilation problems:

Since the library is not maintained anymore there are 2 main issues that I occurred while trying to build ORB_SLAM2:

1. Usleep error which can be fixed by adding #include <unistd> in System.h

2. std::map must have the same value_type as its allocator which can be fixed by editing Modifying ORB_SLAM2/include/LoopClosing.h::50

```
typedef map<KeyFrame*,g2o::Sim3,std::less<KeyFrame*>,
Eigen::aligned_allocator<std::pair<KeyFrame* const, g2o::Sim3> > > KeyFrameAndPose;
```

For this reason I have create a fork https://github.com/erkidhoxholli/ORB_SLAM2 of the original repo and made the above modifications.

Running examples from ORB_SLAM2

References:

https://nicolovaligi.com/bag-of-words-loop-closure-visual-slam.html

https://towardsdatascience.com/image-similarity-using-triplet-loss-3744c0f67973

Y. LeCun, Y. Bengio, and G. Hinton. Deep learning. *Nature*, 521(7553):436–444, 2015.
[12]E. Hoffer and N. Ailon. Deep metric learning using a triplet network. *Arxiv*, 2014.