pySLAM: An Open-Source, Modular, and Extensible Framework for Visual SLAM

Luigi Freda

January 21, 2025

Contents

Introduction

pySLAM is a python implementation of a *Visual SLAM* pipeline that supports **monocular**, **stereo** and **RGBD** cameras. It provides the following **features**:

- A wide range of classical and modern **local features** with a convenient interface for their integration.
- Various loop closing methods, including **descriptor aggregators** such as visual Bag of Words (BoW, iBow), Vector of Locally Aggregated Descriptors (VLAD), and modern **global descriptors** (image-wise descriptors).
- A **volumetric reconstruction pipeline** that processes available depth and color images with volumetric integration and provides an output dense reconstruction. This can use **TSDF** with voxel hashing or incremental **Gaussian Splatting**.
- Integration of **depth prediction models** within the SLAM pipeline. These include DepthPro, DepthAnythingV2, RAFT-Stereo, CREStereo, etc.
- A collection of other useful tools for VO and SLAM.

Main Scripts

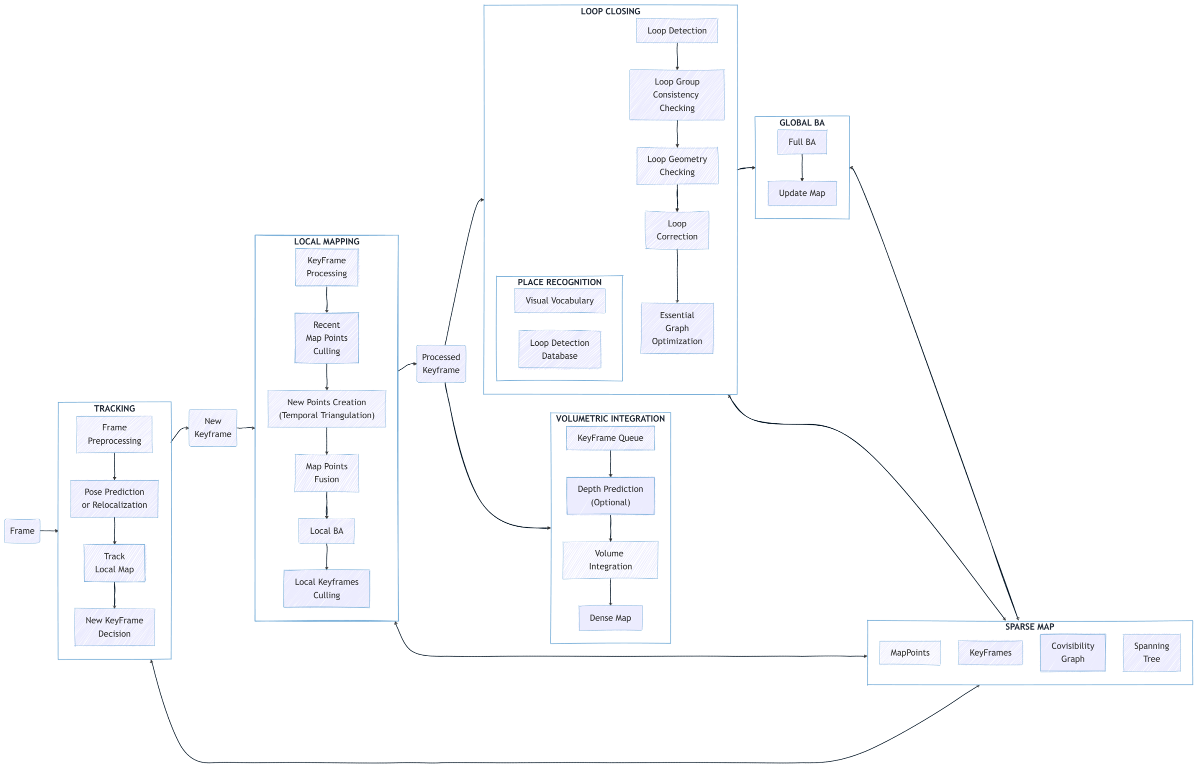
- `main_vo.py` combines the simplest VO ingredients without performing any image point triangulation or windowed bundle adjustment. At each step k, `main_vo.py` estimates the current camera pose C_k with respect to the previous one C_{k-1}. The inter-frame pose estimation returns $[R_{k-1,k}, t_{k-1,k}]$ with $\|t_{k-1,k}\| = 1$. With this very basic approach, you need to use a ground truth in order to recover a correct inter-frame scale s and estimate a valid trajectory by composing $C_k = C_{k-1}[R_{k-1,k}, st_{k-1,k}]$. This script is a first start to understand the basics of inter-frame feature tracking and camera pose estimation.
- `main_slam.py` adds feature tracking along multiple frames, point triangulation, keyframe management, bundle adjustment, loop closing, dense mapping and depth inference in order to estimate the camera trajectory and build both a sparse and dense map. It's a full SLAM pipeline and includes all the basic and advanced blocks which are necessary to develop a real visual SLAM pipeline.
- `main_feature_matching.py` shows how to use the basic feature tracker capabilities (*feature detector* + *feature descriptor* + *feature matcher*) and allows to test the different available local features.
- `main_depth_prediction.py` shows how to use the available depth inference models to get depth estimations from input color images.
- `main_map_viewer.py` reloads a saved map and visualizes it. Further details on how to save a map here.
- `main_map_dense_reconstruction.py` reloads a saved map and uses a configured volumetric integrator to obtain a dense reconstruction (see here).

You can use the pySLAM framework as a baseline to experiment with VO techniques, *local features*, *descriptor aggregators*, *global descriptors*, *volumetric integration*, *depth prediction*, and create your own (proof of concept) VO/SLAM pipeline in python. When working with it, please keep in mind this is a research framework written in Python and a work in progress. It is not designed for real-time performances.

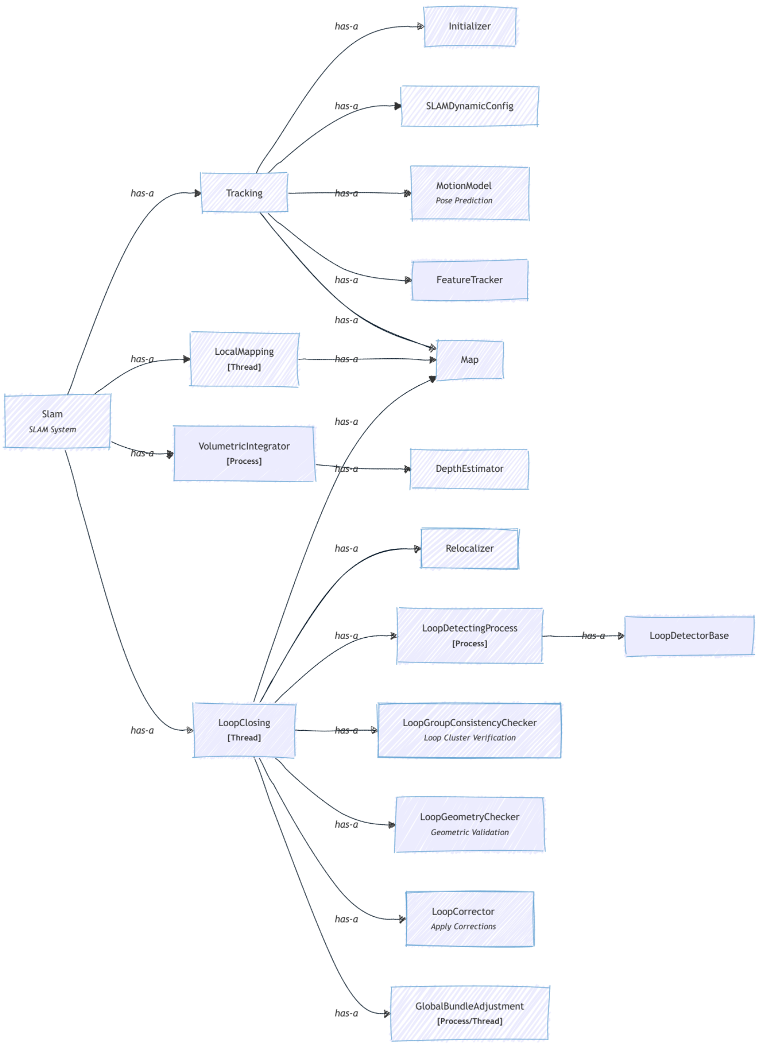
Enjoy it!

System overview

SLAM Workflow



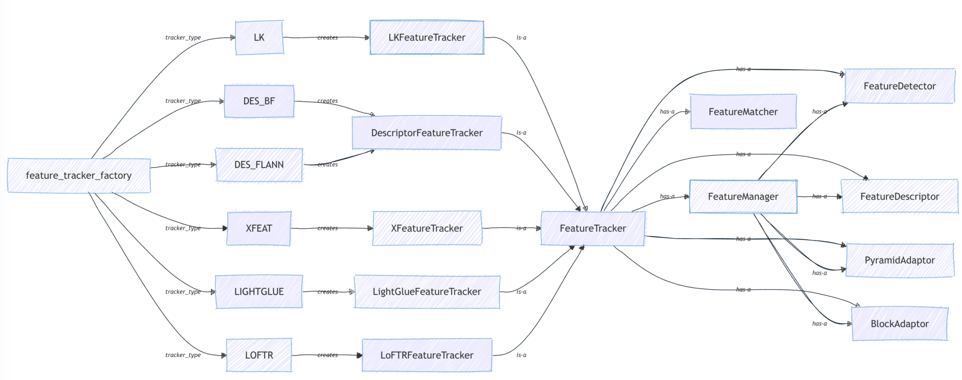
SLAM Components



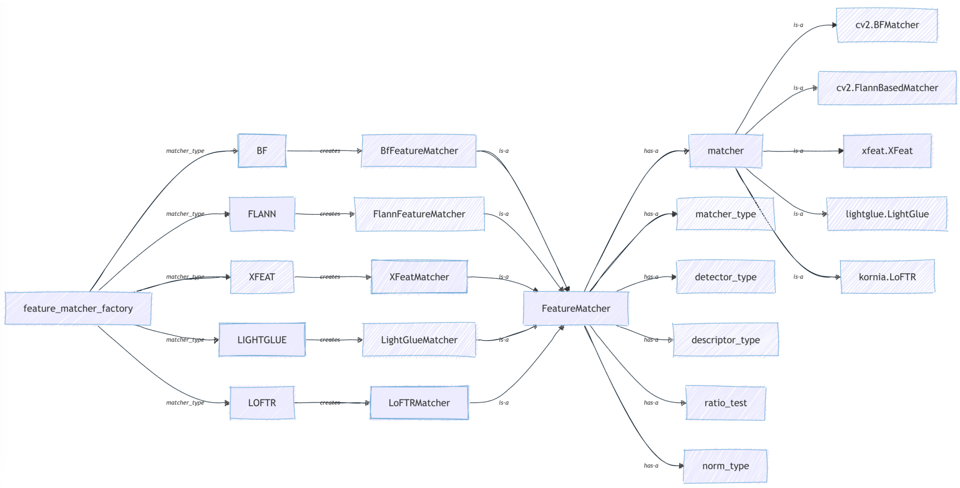
Note: You might be wondering why I used **Processes** instead of **Threads** in some cases. The reason is that, at least in Python 3.8 (the version supporting pySLAM), only one thread can execute at a time within a single process due to the Global Interpreter Lock (GIL). On the other hand, using multiprocessing (separate processes that do not share the GIL) enables better parallelism. See this nice post.

Main System Components

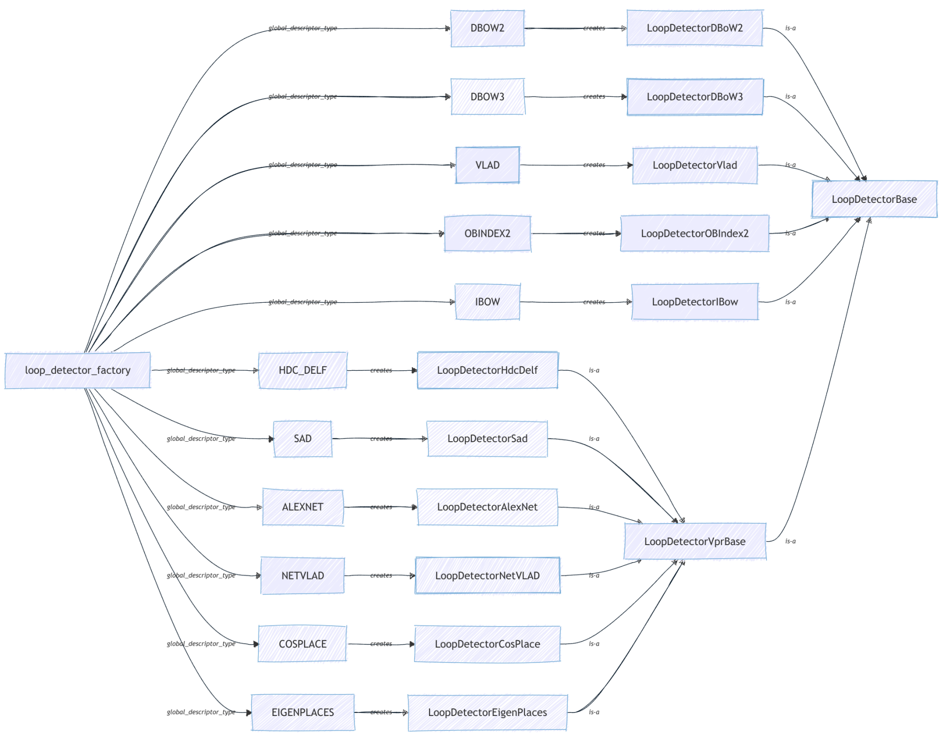
Feature Tracker



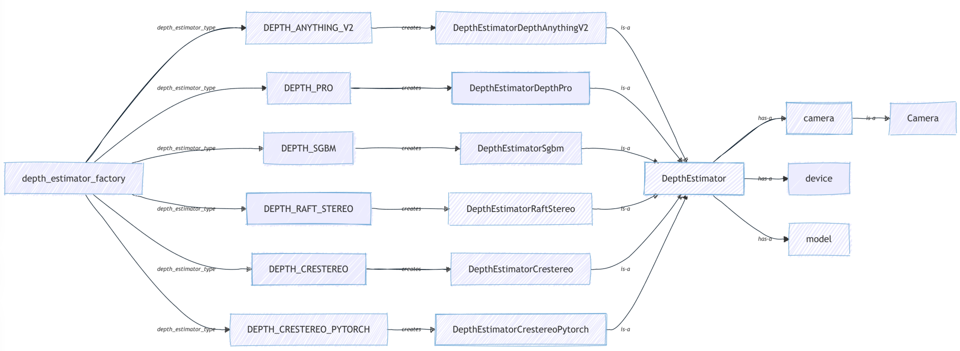
Feature Matcher



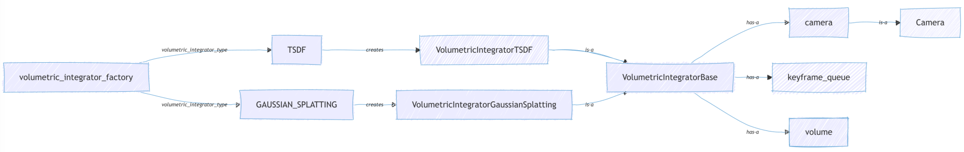
Loop Detector



Depth Estimator



Volumetric Integrator



Usage

Once you have run the script `install_all_venv.sh` (follow the instructions above according to your OS), you can open a new terminal and start testing the basic **Visual Odometry** (VO):

```
$ . pyenv-activate.sh   #  Activate pyslam python virtual environment. This is only needed once in a new terminal.
$ ./main_vo.py
```

This will process a default KITTI video (available in the folder `data/videos`) by using its corresponding camera calibration file (available in the folder `settings`), and its groundtruth (available in the same `data/videos` folder). If matplotlib windows are used, you can stop `main_vo.py` by focusing/clicking on one of them and pressing the key 'Q'. As explained above, this very *basic* script `main_vo.py` **strictly requires a ground truth**.

Similarly, you can test the **full SLAM** by running `main_slam.py`:

```
$ . pyenv-activate.sh   #  Activate pyslam python virtual environment. This is only needed once in a new terminal.
$ ./main_slam.py
```

This will process the same default KITTI video (available in the folder `data/videos`) by using its corresponding camera calibration file (available in the folder `settings`). You can stop it by focusing/clicking on one of the opened windows and pressing the key 'Q' or closing the 3D pangolin GUI.

With both scripts, in order to process a different **dataset**, you need to update the file `config.yaml`: * Select your dataset `type` in the section `DATASET` (further details in the section *Datasets* below for further details). This identifies a corresponding dataset section (e.g. `KITTI_DATASET`, `TUM_DATASET`, etc). * Select the `sensor_type` (`mono`, `stereo`, `rgbd`) in the chosen dataset section.
* Select the camera `settings` file in the dataset section (further details in the section *Camera Settings* below). * The `groudtruth_file` accordingly (further details in the section *Datasets* below and check the files `io/ground_truth.py` and `io/convert_groundtruth.py`).

Feature tracking

If you just want to test the basic feature tracking capabilities (*feature detector* + *feature descriptor* + *feature matcher*) and get a taste of the different available local features, run

```
$ . pyenv-activate.sh   #  Activate pyslam python virtual environment. This is only needed once in a new terminal.
$./main_feature_matching.py
```

In any of the above scripts, you can choose any detector/descriptor among *ORB*, *SIFT*, *SURF*, *BRISK*, *AKAZE*, *SuperPoint*, etc. (see the section *Supported Local Features* below for further information).

Some basic examples are available in the subfolder `test/loopclosing`. In particular, as for feature detection/description, you may want to take a look at test/cv/test_feature_manager.py too.

Loop closing

Different loop closing methods are available, combining aggregation methods and global descriptors.

While running full SLAM, loop closing is enabled by default and can be disabled by setting `kUseLoopClosing=False` in `config_parameters.py`. Configuration options can be found in loop_closing/loop_detector_configs.py.

Examples: Start with the examples in `test/loopclosing`, such as test/loopclosing/test_loop_detector.py.

Vocabulary management

`DBoW2`, `DBoW3`, and `VLAD` require pre-trained vocabularies. ORB-based vocabularies are automatically downloaded in the `data` folder (see loop_closing/loop_detector_configs.py).

To create a new vocabulary, follow these steps:

1. **Generate an array of descriptors**: Use the script `test/loopclosing/test_gen_des_array_from_imgs.py` to generate the array of descriptors that will be used to train the new vocabulary. Select your desired descriptor type via the tracker configuration.
2. **DBOW vocabulary generation**: Train your target DBOW vocabulary by using the script `test/loopclosing/test_gen_dbow_voc_from_des_array.py`.
3. **VLAD vocabulary generation**: Train your target VLAD "vocabulary" by using the script `test/loopclosing/test_gen_vlad_voc_from_des_array.py`.

Vocabulary-free loop closing

Most methods do not require pre-trained vocabularies. Specifically: - `iBoW` and `OBindex2`: These methods incrementally build bags of binary words and, if needed, convert (front-end) non-binary descriptors into binary ones. - Others: Methods like `HDC_DELF`, `SAD`, `AlexNet`, `NetVLAD`, `CosPlace`, and `EigenPlaces` directly extract global descriptors and process them using dedicated aggregators, independently from the used front-end descriptors.

As mentioned above, only `DBoW2`, `DBoW3`, and `VLAD` require pre-trained vocabularies.

Volumetric reconstruction

Dense reconstruction while running SLAM

The SLAM back-end hosts a volumetric reconstruction pipeline. This is disabled by default. You can enable it by setting `kUseVolumetricIntegration=True` and selecting your preferred method `kVolumetricIntegrationType` in `config_parameters.py`. At present, two methods are available: `TSDF` and `GAUSSIAN_SPLATTING` (see dense/volumetric_integrator_factory.py). Note that you need CUDA in order to run `GAUSSIAN_SPLATTING` method.

At present, the volumetric reconstruction pipeline works with: - RGBD datasets - When a depth estimator is used in the back-end or front-end and a depth prediction/estimation gets available for each processed keyframe.

If you want a mesh as output then set `kVolumetricIntegrationExtractMesh=True` in `config_parameters.py`.

Reload a saved sparse map and perform dense reconstruction

Use the script `main_map_dense_reconstruction.py` to reload a saved sparse map and to perform dense reconstruction by using its posed keyframes as input. You can select your preferred dense reconstruction method directly in the script.

- To check what the volumetric integrator is doing, run in another shell `tail -f logs/volumetric_integrator.log` (from repository root folder).
- To save the obtained dense and sparse maps, press the `Save` button on the GUI.

Reload and check your dense reconstruction

You can check the output pointcloud/mesh by using CloudCompare.

In the case of a saved Gaussian splatting model, you can visualize it by:

1. Using the superslat editor (drag and drop the saved Gaussian splatting `.ply` pointcloud in the editor interface).
2. Getting into the folder `test/gaussian_splatting` and running:
   `$ python test_gsm.py --load <gs_checkpoint_path>`

Controlling the spatial distribution of keyframe FOV centers

If you are targeting volumetric reconstruction while running SLAM, you can enable a **keyframe generation policy** designed to manage the spatial distribution of keyframe field-of-view (FOV) centers. The *FOV center of a camera* is defined as the backprojection of its image center, calculated using the median depth of the frame. With this policy, a new keyframe is generated only if its FOV center is farther than a predefined distance from the nearest existing keyframe's FOV center. You can enable this policy by setting the following parameters in the yaml setting:

```
KeyFrame.useFovCentersBasedGeneration: 1  # compute 3D fov centers of camera frames by using median depth and use their distances to control ke
KeyFrame.maxFovCentersDistance: 0.2       # max distance between fov centers in order to generate a keyframe
```

Depth prediction

The available depth prediction models can be utilized both in the SLAM back-end and front-end. - Back-end: Depth prediction can be enabled in the volumetric reconstruction pipeline by setting the parameter `kVolumetricIntegrationUseDepthEstimator=True` and selecting your preferred `kVolumetricIntegrationDepthEstimatorType` in `config_parameters.py`. - Front-end: Depth prediction can be enabled in the front-end by setting the parameter `kUseDepthEstimatorInFrontEnd` in `config_parameters.py`. This feature estimates depth images from input color images to emulate a RGBD camera. Please, note this functionality is still *experimental* at present time.

Refer to the file `depth_estimation/depth_estimator_factory.py` for further details. Both stereo and monocular prediction approaches are supported. You can test depth prediction/estimation by using the script `main_depth_prediction.py`.

Notes: * In the case of a monocular SLAM configuration, do NOT use depth prediction in the back-end volumetric integration: The SLAM (fake) scale will conflict with the absolute metric scale of depth predictions. With monocular datasets, enable depth prediction to run in the front-end. - The depth inference may be very slow (for instance, with DepthPro it takes ~1s per image on my machine). Therefore, the resulting volumetric reconstruction pipeline may be very slow.

Save the a map

When you run the script `main_slam.py` (`main_map_dense_reconstruction.py`): - You can save the current map state by pressing the button `Save` on the GUI. This saves the current map along with front-end, and backend configurations into the default folder `results/slam_state` (`results/slam_state_dense_reconstruction`). - To change the default saving path, open `config.yaml` and update target `folder_path` in the section: `bash SYSTEM_STATE: folder_path: results/slam_state # default folder path (relative to repository root) where the system state is saved or reloaded`

Reload a saved map and relocalize in it

- A saved map can be loaded and visualized in the GUI by running:

```
$ . pyenv-activate.sh   #  Activate pyslam python virtual environment. This is only needed once in a new terminal.
$ ./main_map_viewer.py  #  Use the --path options to change the input path
```

- To enable map reloading and relocalization when running `main_slam.py`, open `config.yaml` and set

```
SYSTEM_STATE:
  load_state: True                  # flag to enable SLAM state reloading (map state + loop closing state)
  folder_path: results/slam_state   # default folder path (relative to repository root) where the system state is saved or reloaded
```

Note that pressing the `Save` button saves the current map, front-end, and backend configurations. Reloading a saved map overwrites the current system configurations to ensure descriptor compatibility.

Trajectory saving

Estimated trajectories can be saved in three different formats: *TUM* (The Open Mapping format), *KITTI* (KITTI Odometry format), and *EuRoC* (EuRoC MAV format). To enable trajectory saving, open `config.yaml` and search for the `SAVE_TRAJECTORY`: set `save_trajectory: True`, select your `format_type` (`tum`, `kitti`, `euroc`), and the output filename. For instance for a `tum` format output:

```
SAVE_TRAJECTORY:
  save_trajectory: True
  format_type: tum
  filename: results/kitti_trajectory.txt
```

SLAM GUI

Some quick information about the non-trivial GUI buttons of `main_slam.py`:

- `Step`: Enter the *Step by step mode*. Press the button `Step` a first time to pause. Then, press it again to make the pipeline process a single new frame.
- `Save`: Save the map into the file `map.json`. You can visualize it back by using the script `/main_map_viewer.py` (as explained above).
- `Reset`: Reset SLAM system.

- `Draw Ground Truth`: If a ground truth dataset is loaded (e.g., from KITTI, TUM, EUROC, or REPLICA), you can visualize it by pressing this button. The ground truth trajectory will be displayed in 3D and progressively aligned (approximately every 30 frames) with the estimated trajectory. The alignment improves as more samples are added to the estimated trajectory. After ~20 frames, if the button is pressed, a window will appear showing the Cartesian alignment errors (ground truth vs. estimated trajectory) along the axes.

Monitor the logs for tracking, local mapping, and loop closing simultaneously

The logs generated by the modules `local_mapping.py`, `loop_closing.py`, `loop_detecting_process.py`, and `global_bundle_adjustments.py` are collected in the files `local_mapping.log`, `loop_closing.log`, `loop_detecting.log`, and `gba.log`, which are all stored in the folder `logs`. For debugging, you can monitor a parallel flow by running the following command in a separate shell:

```
$ tail -f logs/<log file name>
```

Otherwise, to check all parallel logs with tmux, run:

```
$ ./scripts/launch_tmux_logs.sh
```

To launch slam and check all logs in a single tmux, run:

```
$ ./scripts/launch_tmux_slam.sh
```

Press `CTRL+A` and then `CTRL+Q` to exit from `tmux` environment.

Supported components and models

Supported local features

At present time, the following feature **detectors** are supported:

- FAST [45]
- Good features to track [48]
- ORB [46]
- ORB2 (improvements of ORB-SLAM2 to ORB detector)
- SIFT [25]
- SURF [8]
- KAZE [1]
- AKAZE [2]
- BRISK [19]
- AGAST
- MSER [30]
- StarDector/CenSurE
- Harris-Laplace
- SuperPoint
- D2-Net [13]
- DELF [38]
- Contextdesc [28]
- LFNet [39]
- R2D2 [43]
- Key.Net [5]
- DISK [57]
- ALIKED [6]
- Xfeat [7]
- KeyNetAffNetHardNet (KeyNet detector + AffNet + HardNet descriptor)

The following feature **descriptors** are supported:

- ORB [46]
- SIFT [25]
- ROOT SIFT
- SURF [8]
- AKAZE [2]
- BRISK [19]
- FREAK
- SuperPoint
- Tfeat
- BOOST-DESC [56]
- DAISY [55]
- LATCH [20]
- LUCID
- VGG [49]
- Hardnet [32]
- GeoDesc [60]
- SOSNet
- L2Net
- Log-polar descriptor
- D2-Net [13]
- DELF [38]
- Contextdesc [28]
- LFNet [39]
- R2D2 [43]
- BEBLID
- DISK [57]
- ALIKED [6]
- Xfeat [7]
- KeyNetAffNetHardNet (KeyNet detector + AffNet + HardNet descriptor)

For more information, refer to local_features/feature_types.py file. Some of the local features consist of a *joint detector-descriptor*. You can start playing with the supported local features by taking a look at `test/cv/test_feature_manager.py` and `main_feature_matching.py`.

In both the scripts `main_vo.py` and `main_slam.py`, you can create your preferred detector-descritor configuration and feed it to the function `feature_tracker_factory()`. Some ready-to-use configurations are already available in the file local_features/feature_tracker.configs.py

The function `feature_tracker_factory()` can be found in the file `local_features/feature_tracker.py`. Take a look at the file `local_features/feature_manager.py` for further details.

N.B.: You just need a *single* python environment to be able to work with all the supported local features!

Supported matchers

- `BF`: Brute force matcher on descriptors (with KNN).
- FLANN [34]
- XFeat [7]
- LightGlue

- LoFTR

See the file `local_features/feature_matcher.py` for further details.

Supported global descriptors and local descriptor aggregation methods

Local descriptor aggregation methods

- Bag of Words (BoW): DBoW2 [16], DBoW3. [paper]
- Vector of Locally Aggregated Descriptors: VLAD [3]. [paper]
- Incremental Bags of Binary Words (iBoW) via Online Binary Image Index: iBoW, OBIndex2. [paper]
- Hyperdimensional Computing: HDC [36]. [paper]

NOTE: *iBoW* and *OBIndex2* incrementally build a binary image index and do not need a prebuilt vocabulary. In the implemented classes, when needed, the input non-binary local descriptors are transparently transformed into binary descriptors.

Global descriptors

Also referred to as *holistic descriptors*:

- SAD
- AlexNet
- NetVLAD [3]
- HDC-DELF
- CosPlace [9]
- EigenPlaces [10]

Different loop closing methods are available. These combines the above aggregation methods and global descriptors. See the file loop_closing/loop_detector_configs.py for further details.

Supported depth prediction models

Both monocular and stereo depth prediction models are available. SGBM algorithm has been included as a classic reference approach.

- SGBM: Depth SGBM from OpenCV (Stereo, classic approach) [17]
- Depth-Pro (Monocular) [11]
- DepthAnythingV2 (Monocular) [51]
- RAFT-Stereo (Stereo) [52]
- CREStereo (Stereo) [22]

Supported volumetric mapping methods

- TSDF with voxel block grid (parallel spatial hashing)
- Incremental 3D Gaussian Splatting. See here and MonoGS for a description of its backend [18].

Camera Settings

The folder `settings` contains the camera settings files which can be used for testing the code. These are the same used in the framework ORB-SLAM2. You can easily modify one of those files for creating your own new calibration file (for your new datasets).

In order to calibrate your camera, you can use the scripts in the folder `calibration`. In particular: 1. Use the script `grab_chessboard_images.py` to collect a sequence of images where the chessboard can be detected (set the chessboard size therein, you can use the calibration pattern `calib_pattern.pdf` in the same folder) 2. Use the script `calibrate.py` to process the collected images and compute the calibration parameters (set the chessboard size therein)

For more information on the calibration process, see this tutorial or this other link.

If you want to **use your camera**, you have to: * Calibrate it and configure WEBCAM.yaml accordingly * Record a video (for instance, by using `save_video.py` in the folder `calibration`) * Configure the

Supported components and models

Supported local features

At present time, the following feature **detectors** are supported:

- FAST [45]
- Good features to track [48]
- ORB [46]
- ORB2 (improvements of ORB-SLAM2 to ORB detector)
- SIFT [25]
- SURF [8]
- KAZE [1]
- AKAZE [2]
- BRISK [19]
- AGAST
- MSER [30]
- StarDector/CenSurE
- Harris-Laplace
- SuperPoint
- D2-Net [13]
- DELF [38]
- Contextdesc [28]
- LFNet [39]
- R2D2 [43]
- Key.Net [5]
- DISK [57]
- ALIKED [6]
- Xfeat [7]
- KeyNetAffNetHardNet (KeyNet detector + AffNet + HardNet descriptor)

The following feature **descriptors** are supported:

- ORB [46]
- SIFT [25]
- ROOT SIFT
- SURF [8]
- AKAZE [2]
- BRISK [19]
- FREAK
- SuperPoint
- Tfeat
- BOOST-DESC [56]
- DAISY [55]
- LATCH [20]
- LUCID

- VGG [49]
- Hardnet [32]
- GeoDesc [60]
- SOSNet
- L2Net
- Log-polar descriptor
- D2-Net [13]
- DELF [38]
- Contextdesc [28]
- LFNet [39]
- R2D2 [43]
- BEBLID
- DISK [57]
- ALIKED [6]
- Xfeat [7]
- KeyNetAffNetHardNet (KeyNet detector + AffNet + HardNet descriptor)

For more information, refer to local_features/feature_types.py file. Some of the local features consist of a *joint detector-descriptor*. You can start playing with the supported local features by taking a look at `test/cv/test_feature_manager.py` and `main_feature_matching.py`.

In both the scripts `main_vo.py` and `main_slam.py`, you can create your preferred detector-descritor configuration and feed it to the function `feature_tracker_factory()`. Some ready-to-use configurations are already available in the file local_features/feature_tracker.configs.py

The function `feature_tracker_factory()` can be found in the file `local_features/feature_tracker.py`. Take a look at the file `local_features/feature_manager.py` for further details.

N.B.: You just need a *single* python environment to be able to work with all the supported local features!

Supported matchers

- `BF`: Brute force matcher on descriptors (with KNN).
- FLANN [34]
- XFeat [7]
- LightGlue
- LoFTR

See the file `local_features/feature_matcher.py` for further details.

Supported global descriptors and local descriptor aggregation methods

Local descriptor aggregation methods

- Bag of Words (BoW): DBoW2 [16], DBoW3. [paper]
- Vector of Locally Aggregated Descriptors: VLAD [3]. [paper]
- Incremental Bags of Binary Words (iBoW) via Online Binary Image Index: iBoW, OBIndex2. [paper]
- Hyperdimensional Computing: HDC [36]. [paper]

NOTE: *iBoW* and *OBIndex2* incrementally build a binary image index and do not need a prebuilt vocabulary. In the implemented classes, when needed, the input non-binary local descriptors are transparently transformed into binary descriptors.

Global descriptors

Also referred to as *holistic descriptors*:

- SAD
- AlexNet
- NetVLAD [3]
- HDC-DELF
- CosPlace [9]
- EigenPlaces [10]

Different loop closing methods are available. These combines the above aggregation methods and global descriptors. See the file loop_closing/loop_detector_configs.py for further details.

Supported depth prediction models

Both monocular and stereo depth prediction models are available. SGBM algorithm has been included as a classic reference approach.

- SGBM: Depth SGBM from OpenCV (Stereo, classic approach) [17]
- Depth-Pro (Monocular) [11]
- DepthAnythingV2 (Monocular) [51]
- RAFT-Stereo (Stereo) [52]
- CREStereo (Stereo) [22]

Supported volumetric mapping methods

- TSDF with voxel block grid (parallel spatial hashing)
- Incremental 3D Gaussian Splatting. See here and MonoGS for a description of its backend [18].

Camera Settings

The folder `settings` contains the camera settings files which can be used for testing the code. These are the same used in the framework ORB-SLAM2 [35]. You can easily modify one of those files for creating your own new calibration file (for your new datasets).

In order to calibrate your camera, you can use the scripts in the folder `calibration`. In particular: 1. Use the script `grab_chessboard_images.py` to collect a sequence of images where the chessboard can be detected (set the chessboard size therein, you can use the calibration pattern `calib_pattern.pdf` in the same folder) 2. Use the script `calibrate.py` to process the collected images and compute the calibration parameters (set the chessboard size therein)

For more information on the calibration process, see this tutorial [29] or this other link [41].

If you want to **use your camera**, you have to: * Calibrate it and configure WEBCAM.yaml accordingly * Record a video (for instance, by using `save_video.py` in the folder `calibration`) * Configure the `VIDEO_DATASET` section of `config.yaml` in order to point to your recorded video.

Contributing to pySLAM

If you like pySLAM and would like to contribute to the code base, you can report bugs, leave comments and proposing new features through issues and pull requests on github. Feel free to get in touch at *luigifreda(at)gmail[dot]com*. Thank you!

Credits

- Pangolin
- g2opy
- ORBSLAM2 [35]
- SuperPointPretrainedNetwork [12]
- Tfeat [4]
- Image Matching Benchmark Baselines [59]
- Hardnet [33]
- GeoDesc [27]

- SOSNet [54]
- L2Net [53]
- Log-polar descriptor [15]
- D2-Net [14]
- DELF [37]
- Contextdesc [26]
- LFNet [40]
- R2D2 [44]
- BEBLID [50]
- DISK [58]
- Xfeat [42]
- LightGlue [23]
- Key.Net [5]
- Twitchslam
- MonoVO
- VPR_Tutorial [47]
- DepthAnythingV2 [61]
- DepthPro [11]
- RAFT-Stereo [24]
- CREStereo and CREStereo-Pytorch [21]
- MonoGS [31]
- Many thanks to Anathonic for adding the trajectory-saving feature and for the comparison notebook: pySLAM vs ORB-SLAM3.

References

[1] Pablo F Alcantarilla, Adrien Bartoli, and Andrew J Davison. Kaze features. *European conference on computer vision*, pages 214–227, 2012.

[2] Pablo F Alcantarilla, Jesús Nuevo, and Adrien Bartoli. Fast explicit diffusion for accelerated features in nonlinear scale spaces. *IEEE transactions on pattern analysis and machine intelligence*, 34(7):1281–1298, 2013.

[3] Relja Arandjelovic, Petr Gronat, Akihiko Torii, Tomas Pajdla, and Josef Sivic. Netvlad: Cnn architecture for weakly supervised place recognition. *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 5297–5307, 2016.

[4] Vassileios Balntas, Edgar Riba, Daniel Ponsa, and Krystian Mikolajczyk. Learning local feature descriptors with triplets and shallow convolutional neural networks. In *Bmvc*, volume 1, page 3, 2016.

[5] Axel Barroso-Laguna, Edgar Riba, Daniel Ponsa, and Krystian Mikolajczyk. Key.net: Keypoint detection by handcrafted and learned cnn filters. *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 5836–5844, 2020.

[6] Axel Barroso-Laguna, Edgar Riba, Daniel Ponsa, and Krystian Mikolajczyk. Aliked: A lightweight keypoint detector and descriptor. *arXiv preprint arXiv:2304.03608*, 2023.

[7] Axel Barroso-Laguna, Edgar Riba, Daniel Ponsa, and Krystian Mikolajczyk. Xfeat: A new feature detector and descriptor. *arXiv preprint arXiv:2404.19174*, 2024.

[8] Herbert Bay, Tinne Tuytelaars, and Luc Van Gool. Surf: Speeded up robust features. *European conference on computer vision*, pages 404–417, 2006.

[9] Gabriele Berton, Carlo Masone, and Barbara Caputo. Cosplace: Efficient place recognition with cosine similarity. *arXiv preprint arXiv:2304.03608*, 2023.

[10] Gabriele Berton, Carlo Masone, and Barbara Caputo. Eigenplaces: Learning place recognition with eigenvectors. *arXiv preprint arXiv:2404.19174*, 2023.

[11] Aleksei Bochkovskii, Amaël Delaunoy, Hugo Germain, Marcel Santos, Yichao Zhou, Stephan R Richter, and Vladlen Koltun. Depth pro: Sharp monocular metric depth in less than a second. *arXiv preprint arXiv:2410.02073*, 2024.

[12] Daniel DeTone, Tomasz Malisiewicz, and Andrew Rabinovich. Superpoint: Self-supervised interest point detection and description. In *CVPR Deep Learning for Visual SLAM Workshop*, 2018.

[13] Mihai Dusmanu, Ignacio Rocco, Tomas Pajdla, Marc Pollefeys, Josef Sivic, Akihiko Torii, and Torsten Sattler. D2-net: A trainable cnn for joint description and detection of local features. *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 8092–8101, 2019.

[14] Mihai Dusmanu, Ignacio Rocco, Tomas Pajdla, Marc Pollefeys, Josef Sivic, Akihiko Torii, and Torsten Sattler. D2-Net: A Trainable CNN for Joint Detection and Description of Local Features. In *Proceedings of the 2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2019.

[15] Patrick Ebel, Anastasiia Mishchuk, Kwang Moo Yi, Pascal Fua, and Eduard Trulls. Beyond Cartesian Representations for Local Descriptors. 2019.

[16] Dorian Galvez-Lopez and Juan D Tardos. Bags of binary words for fast place recognition in image sequences. *IEEE Transactions on Robotics*, 28(5):1188–1197, 2012.

[17] Heiko Hirschmuller. Stereo processing by semiglobal matching and mutual information. *IEEE Transactions on pattern analysis and machine intelligence*, 30(2):328–341, 2007.

[18] Bernhard Kerbl et al. Monogs: Monocular 3d gaussian splatting. *arXiv preprint arXiv:2312.06741*, 2023.

[19] Stefan Leutenegger, Margarita Chli, and Roland Y Siegwart. Brisk: Binary robust invariant scalable keypoints. *2011 International conference on computer vision*, pages 2548–2555, 2011.

[20] Gil Levi, Tal Hassner, and Ronen Basri. The latch descriptor: Local binary patterns for image matching. *IEEE transactions on pattern analysis and machine intelligence*, 38(8):1622–1634, 2016.

[21] Jiankun Li, Peisen Wang, Pengfei Xiong, Tao Cai, Ziwei Yan, Lei Yang, Jiangyu Liu, Haoqiang Fan, and Shuaicheng Liu. Practical stereo matching via cascaded recurrent network with adaptive correlation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 16263–16272, 2022.

[22] Zhengfa Li, Yuhua Liu, Tianwei Shen, Shuaicheng Chen, Lu Fang, and Long Quan. Crestereo: Cross-scale cost aggregation for stereo matching. *arXiv preprint arXiv:2203.11483*, 2022.

[23] Philipp Lindenberger, Paul-Edouard Sarlin, and Marc Pollefeys. Lightglue: Local feature matching at light speed. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 17627–17638, 2023.

[24] Lahav Lipson, Zachary Teed, and Jia Deng. Raft-stereo: Multilevel recurrent field transforms for stereo matching. In *International Conference on 3D Vision (3DV)*, 2021.

[25] David G Lowe. Object recognition from local scale-invariant features. *Proceedings of the seventh IEEE international conference on computer vision*, 2:1150–1157, 1999.

[26] Zixin Luo, Tianwei Shen, Lei Zhou, Jiahui Zhang, Yao Yao, Shiwei Li, Tian Fang, and Long Quan. Contextdesc: Local descriptor augmentation with cross-modality context. *Computer Vision and Pattern Recognition (CVPR)*, 2019.

[27] Zixin Luo, Tianwei Shen, Lei Zhou, Siyu Zhu, Runze Zhang, Yao Yao, Tian Fang, and Long Quan. Geodesc: Learning local descriptors by integrating geometry constraints. In *Proceedings of the European conference on computer vision (ECCV)*, pages 168–183, 2018.

[28] Zixin Luo, Lei Zhou, Xiang Bai, Alan Yuille, and Jimmy Ren. Contextdesc: Local descriptor augmentation with cross-modality context. *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 2527–2536, 2020.

[29] Satya Mallick. Camera calibration using opencv, 2016.

[30] Jiri Matas, Ondrej Chum, Martin Urban, and Tomas Pajdla. Robust wide-baseline stereo from maximally stable extremal regions. *Proceedings of the British Machine Vision Conference*, 1(502):384–393, 2002.

[31] Hidenobu Matsuki, Riku Murai, Paul H. J. Kelly, and Andrew J. Davison. Gaussian Splatting SLAM. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2024.

[32] Anastasiia Mishchuk, Dmytro Mishkin, Filip Radenovic, and Jiri Matas. Working hard to know your neighbor's margins: Local descriptor learning loss. *Advances in neural information processing systems*, 30, 2017.

[33] Anastasiya Mishchuk, Dmytro Mishkin, Filip Radenovic, and Jiri Matas. Working hard to know your neighbor's margins: Local descriptor learning loss. In *Proceedings of NeurIPS*, December 2017.

[34] Marius Muja and David G Lowe. Fast approximate nearest neighbors with automatic algorithm configuration. *VISAPP (1)*, 2(331-340):2, 2009.

[35] Raul Mur-Artal and Juan D. Tardos. Orb-slam2: An open-source slam system for monocular, stereo and rgb-d cameras, 2017.

[36] Peer Neubert and Peter Protzel. Hyperdimensional computing as a framework for systematic aggregation of image descriptors. *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 9067–9076, 2021.

[37] Hyeonwoo Noh, Andre Araujo, Jack Sim, Tobias Weyand, and Bohyung Han. Large-scale image retrieval with attentive deep local features. In *Proceedings of the IEEE international conference on computer vision*, pages 3456–3465, 2017.

[38] Hyeonwoo Noh, Andre Araujo, Joonseok Sim, Tobias Weyand, and Bohyung Han. Large-scale image retrieval with attentive deep local features. *Proceedings of the IEEE international conference on computer vision*, pages 3456–3465, 2017.

[39] Yoshitaka Ono, Eduard Trulls, Pascal Fua, and Kwang Moo Yi. Lf-net: Learning local features from images. *Advances in neural information processing systems*, 31, 2018.

[40] Yuki Ono, Eduard Trulls, Pascal Fua, and Kwang Moo Yi. Lf-net: Learning local features from images. *Advances in neural information processing systems*, 31, 2018.

[41] OpenCV. Camera calibration, 2021.

[42] Guilherme Potje, Felipe Cadar, André Araujo, Renato Martins, and Erickson R Nascimento. Xfeat: Accelerated features for lightweight image matching. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 2682–2691, 2024.

[43] Jerome Revaud, Philippe Weinzaepfel, Cedric R De Souza, Nicolas Pion, Gabriela Csurka, Yohann Cabon, and Martin Humenberger. R2d2: Repeatable and reliable detector and descriptor. *Advances in neural information processing systems*, 32, 2019.

[44] Jerome Revaud, Philippe Weinzaepfel, César Roberto de Souza, and Martin Humenberger. R2D2: repeatable and reliable detector and descriptor. In *NeurIPS*, 2019.

[45] Edward Rosten and Tom Drummond. Machine learning for high-speed corner detection. *European conference on computer vision*, pages 430–443, 2006.

[46] Ethan Rublee, Vincent Rabaud, Kurt Konolige, and Gary Bradski. Orb: An efficient alternative to sift or surf. *2011 International conference on computer vision*, pages 2564–2571, 2011.

[47] Stefan Schubert, Peer Neubert, Sourav Garg, Michael Milford, and Tobias Fischer. Visual place recognition: A tutorial. *IEEE Robotics & Automation Magazine*, 2023.

[48] Jianbo Shi and Carlo Tomasi. Good features to track. *1994 Proceedings of IEEE conference on computer vision and pattern recognition*, pages 593–600, 1994.

[49] Karen Simonyan, Andrea Vedaldi, and Andrew Zisserman. Learning local feature descriptors using convex optimisation. *IEEE transactions on pattern analysis and machine intelligence*, 36(8):1573–1585, 2014.

[50] Iago Suárez, Ghesn Sfeir, José M Buenaposada, and Luis Baumela. Beblid: Boosted efficient binary local image descriptor. *Pattern recognition letters*, 133:366–372, 2020.

[51] DepthAnything Team. Depthanythingv2: A monocular depth prediction model. *arXiv preprint arXiv:2406.09414*, 2024.

[52] Zachary Teed and Jia Deng. Raft-stereo: Recurrent all-pairs field transforms for stereo matching. *arXiv preprint arXiv:2109.07547*, 2021.

[53] Yurun Tian, Bin Fan, and Fuchao Wu. L2-net: Deep learning of discriminative patch descriptor in euclidean space. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 661–669, 2017.

[54] Yurun Tian, Xin Yu, Bin Fan, Fuchao Wu, Huub Heijnen, and Vassileios Balntas. Sosnet: Second order similarity regularization for local descriptor learning. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 11016–11025, 2019.

[55] Engin Tola, Vincent Lepetit, and Pascal Fua. Daisy: An efficient dense descriptor applied to wide-baseline stereo. *IEEE transactions on pattern analysis and machine intelligence*, 32(5):815–830, 2010.

[56] Tomasz Trzcinski, Marios Christoudias, Pascal Fua, and Vincent Lepetit. Boosting binary keypoint descriptors. *2013 IEEE Conference on Computer Vision and Pattern Recognition*, pages 2874–2881, 2013.

[57] Maciej Tyszkiewicz, Pascal Fua, and Eduard Trulls. Disk: Learning local features with policy gradient. *Advances in neural information processing systems*, 33:14254–14265, 2020.

[58] Michał Tyszkiewicz, Pascal Fua, and Eduard Trulls. Disk: Learning local features with policy gradient. *Advances in Neural Information Processing Systems*, 33:14254–14265, 2020.

[59] vcg uvic. Image matching benchmark baselines, 2020.

[60] Yannick Verdie, Kwang Moo Yi, Pascal Fua, and Vincent Lepetit. Tilde: A temporally invariant learned detector. *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 5279–5288, 2015.

[61] Lihe Yang, Bingyi Kang, Zilong Huang, Zhen Zhao, Xiaogang Xu, Jiashi Feng, and Hengshuang Zhao. Depth anything v2. *arXiv:2406.09414*, 2024.