L-CNN Powered Junction-SLAM

Jianwei Zhang
University of California, Berkeley
jwzhang19@gmail.com

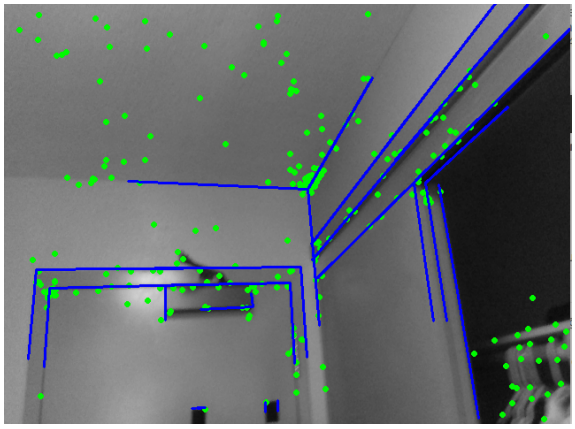
Junseok Lee
University of California, Berkeley
junseok_lee@berkeley.edu

Abstract

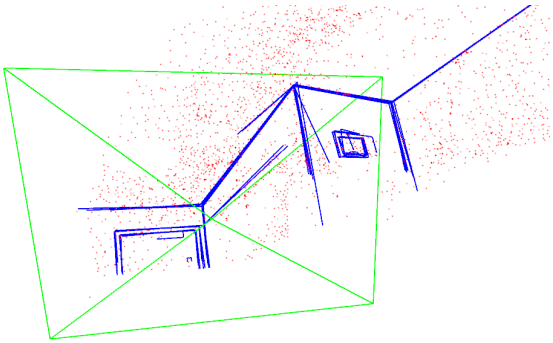
Simultaneous Localization and Mapping (SLAM) is key to robotics and augmented/virtual reality applications. Traditional visual SLAM approaches rely only on point features for pose estimation and mapping. Recent studies propose to combine line features and point features in visual SLAM for robust tracking and mapping. In this paper, we propose LCNN powered junction SLAM which integrates learning-based wireframe parsing using L-CNN into a visual odometry system. Junctions and wireframes detected by a neural network encode strong geometrical information and may provide improved robustness and accuracy especially in low texture environments. Since the LCNN powered junction SLAM is capable of obtaining a high-level representation of environments, it is potentially useful for robotics applications such as control and planning for obstacle avoidance. Furthermore, the reconstructed wireframe map from the LCNN powered junction SLAM provides structural information about the indoor scene, and that can benefit virtual object placement in augmented reality.

1. Introduction

Simultaneous Localization and Mapping (SLAM) is the task of building a map in an *unknown* environment while tracking the pose of the sensor. Both building a map and tracking the pose are essential for robotics application in state estimation, control, and planning. The popularity of the SLAM system emerges as one of the most important problems because of growth in indoor robotics applications, where it is hard to have an exact map *in prior*, and state estimation and planning by other sensors such as magnetometers and GPS are highly limited. SLAM is usually comprised of two parts, odometry and loop closure. The odometry estimates the current pose of a sensor and keeps the trajectory of the sensor pose in real-time. The loop closure is called when the system recognizes re-visiting a place to ensure global map consistency. The SLAM system can be realized by different sensors including cameras, depth cameras, sonar sensors, and LIDARs [1][2]. Due to mechanical simplicity and low power consumption, visual SLAM which is based on either monocular- or stereo-camera is one of the most common SLAM systems on robots [3].



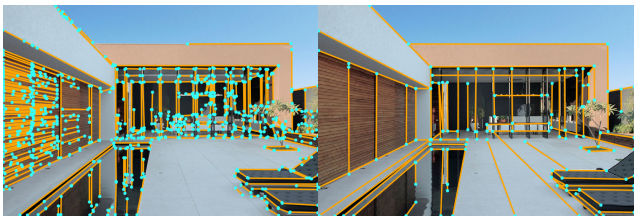
(a) Detected junctions and point features on a frame



(b) 3D map of feature points and wireframes

Figure 1: **L-CNN Powered Junction-SLAM** (a) Green points are the feature points by detected OpenCV GFTTDetector. Blue lines are the edges detected by L-CNN. (b) Red points are the feature points projected in 3D. Blue edges are the detected edges projected in 3D. The green camera frame represents the current estimated camera pose.

Visual SLAM systems can be classified into direct and indirect systems. Direct systems try to minimize the pho-



(a) LSD line detection [7] (b) Wireframe Parsing [8]

Figure 2: **Line Detection** Deep learning-based provides more succinct line detection. Figures from [8].

tometric error of consecutive images while indirect systems first extract features and then solve for camera pose and map point position by minimizing projection errors. Typical features to use in indirect visual SLAM includes point features ranging from ORB [4], to SURF, and to BRISK, or instead curvature extrema [5]. They are detected and described using a local patch of an image.

Existing indirect SLAM systems typically rely on detecting and tracking lots of point features. However, if a frame contains large planar regions with few distinguished point features, these SLAM systems tend to fail. To address this issue, SLAM needs to make better use of geometric hints given by lines and junctions. Existing works have shown that combining line features and point features give more robust performance than point-only SLAM systems [6].

In this paper, we aim to push this further by using junctions detected by learning based wireframe detection methods. Junctions contain stronger geometry information than lines and have the potential to be more robust when few lines are in the scene. Furthermore, learning based wireframe detection gives less lines but obtains simpler representation of environments than typical line detection methods, as shown in 2. We also expect possibilities to speed up SLAM computation by reducing the number of features to process and robotics control and planning thanks to simpler mathematical representation of environments including obstacles.

2. Related Works

2.1. Visual SLAM

SLAM based on a single camera is common, due to simplicity in design, in many robotics applications, for example, aerial vehicles, whose payload is a critical engineering factor. As a consequence of losing one dimension in projection, recovering depth information is possible only up to scale, which means the monocular SLAM system inherently suffer from scale drift unless other components are added. Simply speaking, when using a single camera no one can differentiate between twice a bigger structure and twice closer to the structure. Compared to monocular visual SLAM, stereo visual SLAM is free from scale drift as the baseline of the stereo camera is already known in prior and allows to recover dimension. Similarly as monocular visual SLAM, stereo visual SLAM extracts point features from images and estimate pose from image correspondence [9]. However, that is not always the case that sufficient point features are available in real-world environments. To resolve the difficulty in finding correspondence, RGBD-SLAM takes advantage of RGBD cameras. The RGBD camera projects certain infrared patterns to ease the correspondence problem. However, the infrared patterns are washed out outdoors under sun-lights and have a limited range.

When it comes to human-made environments line features become more prevalent, such as edges of buildings, bridges, and rooms. PL-SLAM is developed to include line features as well as point features in stereo visual SLAM [6]. Leveraging both features PL-SLAM can provide more reliable estimation especially when point features are not well-distributed. PL-SLAM consists of several components that resemble point feature-based SLAM, i.e., visual odometry, keyframe selection, bundle adjustment, and loop closure. Similarly in ORB-SLAM loop closure in PL-SLAM is performed when place recognized using bag-of-visual-words (BoVW) that contain combined descriptors of point and line features. PL-SLAM shows a more robust performance compared to ORB-SLAM.

2.2. Wireframe Parsing

The traditional approach to wireframe parsing problem is a two-staged hierarchical system. First it generates pixel-wise junction and line heat maps using deep neural networks, and sequentially hand-crafted heuristic algorithms search over the line heat maps to localize junctions, line segments, and connectivity between the line segments. L-CNN enables end-to-end wireframe parsing by replacing hand-craft heuristics with deep neural networks to learn heuristics itself for better representation [10, 8]. L-CNN is composed of feature extraction, junction proposal, line sampling, and line verification.

3. L-CNN Powered Junction-SLAM

3.1. System Overview

We identify four tasks that are important for building a visual odometry system with junctions as (i) data association, (ii) local bundle adjustment, (iii) pose estimation, and (iv) map building. (i) Data association is used to track features across frames. Existing methods for point features include optical flow tracking and descriptor matching. We showed that descriptor matching technique would be challenging with L-CNN detected junctions so we designed an optical flow tracker for junctions.(ii) Pose estimation refers

Algorithm 1 Initialize Map

Require: A RGBD image

procedure INITIALIZEMAP
  DETECTFEATURES ▷ GFTTDetector [13]
  DETECTJUNCTIONS ▷ IPC with L-CNN [Fig. 7]
  INITIALIZENEWPOINTS ▷ by depth from RGBD
  INITIALIZENEWJUNCTIONS ▷ by depth from RGBD
  SETKEYFRAME
  INSERTKEYFRAME
  UPDATEMAP
end procedure

Algorithm 2 Track Point and Junction Features

Require: A RGBD image

procedure TRACK
  TRACKFEATUREPOINTS ▷ calcOpticalFlowPyrLK [14]
  TRACKJUNCTIONS ▷ Subsection 3.3
  ESTIMATECURRENTPOSE ▷ Levenberg–Marquardt
  if enough tracking inliers **then**
    tracking_status ← tracking_good
  else
    tracking_status ← tracking_lost
  end if
end procedure

to the task of tracking the current camera pose with respect to a temporarily fixed map, for SLAM with point features, it is done by minimizing projected pixel coordinate error. For junction structure we model junction as a center point and several connected points. We do pose estimation by minimizing coordinate difference of center points and orientation difference of connected points. (iii) Local bundle adjustment minimizes projection error by jointly optimize poses of keyframes and landmark positions. This is done by non-linear optimization. We bundle adjustment of junctions for future work. (iv) Detected wireframes have a special incident relationship between junctions and lines. Leveraging those incident relationships to form a connected wireframe map with both topological and metric information would be an ideal map format for Junction SLAM and will be useful for robotics and AR applications. We leave this part for future work.

Our system framework is inspired by the Visual SLAM tutorial [11]. The visual odometry system is divided into parallel frontend and backend. The visual frontend is responsible for initializing, detecting, and tracking feature points and junctions while computing the camera pose [Alg. 1 and 2]. When the number of tracked features falls below a threshold, the frontend inserts a new keyframe, detects new features, and triggers backend optimization. The backend jointly optimizes the pose of several local keyframes and the positions of observed features by solving Levenberg–Marquardt non-linear optimization. We implemented the optimization in g2o [12]. We applied Huber Robust kernel to guard again outliers and reject outliers with large errors. We fix the size of the backend optimization window, when a new keyframe is inserted, an old keyframes and associated map points will be removed from local bundle adjustment.

3.2. Junction Representation

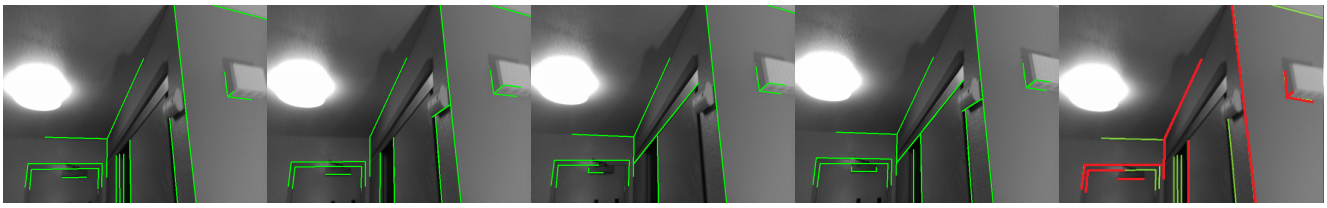
We represent junction as a center point and a vector of connected points. We only include vertexes with a degree more than one or if the connected point also has degree one. This scheme effectively prevents counting boundary points as junctions, which are not reliable to track and will not help pose estimation. We distinguish between 2D junctions and 3D junctions. 2D junctions are generated by L-CNN detection when a keyframe is inserted, and they are tracked within the keyframe. 3D junctions are generated upon keyframe insertion by projecting 2D junctions into 3D. We directly project center point and connected points to 3D using depth map provided by the depth sensor.

3.3. Junction Tracking

L-CNN wireframe detection is sensitive to small image change as shown in Figure 3. We marked edges that exist in all four frames red as shown in Figure 3e. This effect poses a challenge for the detection-description-matching workflow. Matching with the previous frame strategy will lose track of a large portion of junctions and edges quickly.

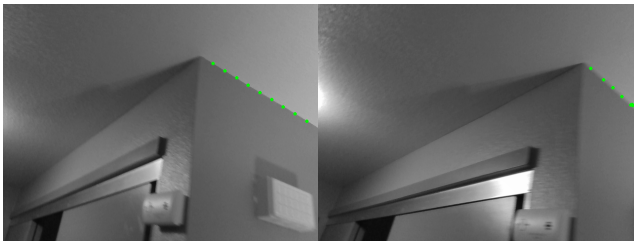
To address this challenge, we used optical flow tracking. Traditionally optical flow tracking is used on a local image patch. We designed a junction tracking scheme based on image patch tracking. We leveraged Lucas-Kanade (LK) optical flow to track local patch [15]. For the junction center, we directly track it with LK optical flow. For edges, we interpolate several middle points and track those middle points as shown in Figure 4.

Lucas-Kanade optical flow can produce reliable results if both eigenvalues of image patch hessian are large [16]. This condition would usually be true for the center point but untrue for interpolated middle points on edges. For interpolated points on edges, we expect it to have one large eigenvalue, whose eigenvector is approximately orthogonal to line direction and one small eigenvalue, whose eigenvector is along the line direction. Even though we can not reliably recover the image patch movement along the line direction, the fraction of image patch displacement perpendicular to the line direction will make the tracked image patch still stay on the edge, making edge orientation reliable to track as seen in Figure 4. To resolve potential numerical issues, when the small eigenvalue is too small or zero, we only calculate image displacement along the direction of the large eigenvector. Since tracking does not preserve relative distancing of interpolated points, we re-interpolate middle



(a) (b) (c) (d) (e)

Figure 3: **L-CNN Output Examples** (a-d) L-CNN detection (green lines) of four consecutive frames. (e) Common edges (red lines) in the all frames.



(a) interpolated middle points (b) tracked after 15 frames

Figure 4: **Junction Tracking** We track an edge by interpolating middle points and track those middle points using the Lucas-Kanade optical flow.

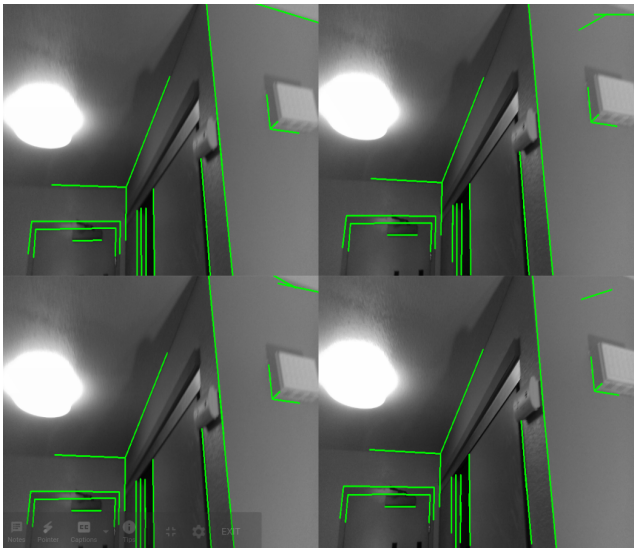


Figure 5: **Junction Tracking Results** Most junctions are successfully tracked across four frames. Consistent tracking can be useful for pose estimation and fusion of detected wireframes.

points at each frame. We show our junction tracking result of four consecutive frames in Figure 5.

3.4. Pose Estimation with Junction

We applied optimization to find optimal camera pose and landmark positions. The camera pose can be represented by a Special Euclidean group $\mathrm{SE}(3)$, which comprises a rotation matrix $R \in \mathrm{SO}(3)$ and translation vector $t \in \mathbf{R}^3$. The associated Lie Algebra can be represented by $\omega, t \in \mathbf{R}^3$. The rotation matrix R and Lie algebra representation ω are related by the exponential map and the log map as follows.

$$e^{\hat{\omega}} = R \tag{1}$$

$$\omega = \widehat{logR} \tag{2}$$

$\hat{\omega}$ is the skew symmetric matrix of vector ω.

$$\hat{\omega} = \begin{bmatrix} 0 & -\omega_3 & \omega_2 \\ \omega_3 & 0 & -\omega_1 \\ -\omega_2 & \omega_1 & 0 \end{bmatrix} \tag{3}$$

For point features, the error can be defined as coordinated error in the camera space. This error can also work for junction center position.

$$error = RX_{map} + t - X_{observed} \tag{4}$$

The Jacobian matrix is

$$\frac{\partial error}{\partial(\delta\omega, \delta t)} = \begin{bmatrix} 0 & -\omega_3 & \omega_2 & 1 & 0 & 0 \\ \omega_3 & 0 & -\omega_1 & 0 & 1 & 0 \\ -\omega_2 & \omega_1 & 0 & 0 & 0 & 1 \end{bmatrix} \tag{5}$$

However, as shown in the previous section, the length of edges cannot be reliably tracked so we only leverage edge orientation for pose estimation. 3D Orientation can be represented by a unit vector in 3D so we define the error as difference of unit orientation vector in the camera space.

$$error = \frac{RX_{map}}{\|X_{map}\|} - \frac{X_{observed}}{\|X_{observed}\|} \tag{6}$$

where

$$X_{map} = X_{\text{connected point in map}} - X_{\text{center point in map}} \tag{7}$$

$$X_{observed} = X_{\text{tracked connected point}} - X_{\text{tracked center point}} \tag{8}$$

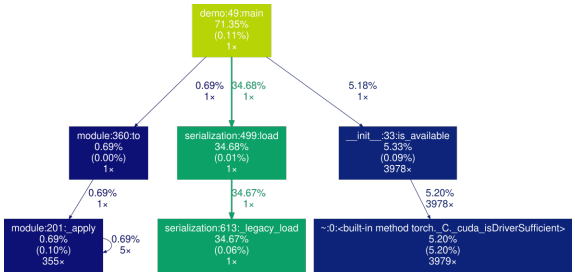


Figure 6: **Bottleneck Analysis** The profiling shows L-CNN has a huge overhead. 35% of the processing time (green nodes) is used for loading pretrained weights in processing a single image. Irrelevant edges pruned.

The Jacobian matrix is

$$\frac{\partial error}{\partial(\delta\omega, \delta t)} = \begin{bmatrix} 0 & -\omega_3 & \omega_2 & 0 & 0 & 0 \\ \omega_3 & 0 & -\omega_1 & 0 & 0 & 0 \\ -\omega_2 & \omega_1 & 0 & 0 & 0 & 0 \end{bmatrix} \quad (9)$$

where $(\delta\omega, \delta t)$ is the left-multiply update on the original $\mathrm{se}(3)$.

$$\omega_{updated} = \widehat{e^{\delta\omega} \cdot e^{\omega}} \quad (10)$$

$$t_{updated} = e^{\widehat{\delta\omega}} \cdot st + \delta t \quad (11)$$

Having obtained the Jacobian and error formulation, we can leverage Levenberg–Marquardt non-linear optimization to solve for optimal pose.

3.5. Profiling L-CNN for Bottleneck Analysis

When L-CNN is called on every image frame, a huge delay from L-CNN makes the SLAM unusable in real-time. In an effort to optimize the delay, we first profile the off-the-shelf L-CNN on a single image pass to analyze the bottleneck [Fig. 6]. The profiling shows the bottleneck is because of loading pretrained weights, which spends 35% of the total processing time.

3.6. Interprocess Communication Design

This leads us to keep L-CNN on memory while the SLAM is running and design L-CNN to communicate with the SLAM using TCP/IP. As reducing the latency is our top priority, we use the lightest-weight libraries for interprocess communication (IPC) and serialization [Fig. 7] rather than JSON or YAML. The IPC is realized by ZeroMQ [17], a high-performance asynchronous messaging library, and serialization is performed using Protocol Buffer [18], a fast and simple library for serializing structured data. Minimal data of rows, columns, channels, image type, and binary data for an image and an array of endpoints of lines for detected junctions is transferred between SLAM and L-CNN.

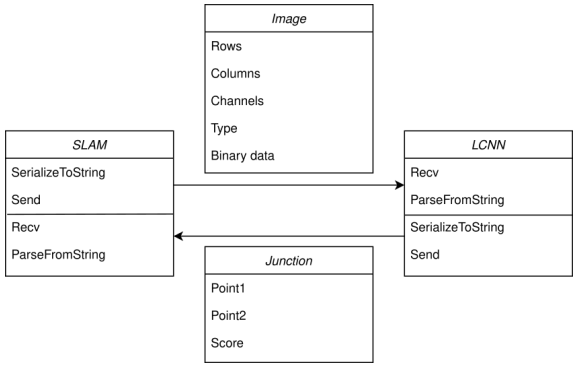


Figure 7: **Interprocess Communication** IPC is introduced to reduce the overhead of loading pretrained weights in L-CNN. Reducing latency is considered top-priority in designing IPC.

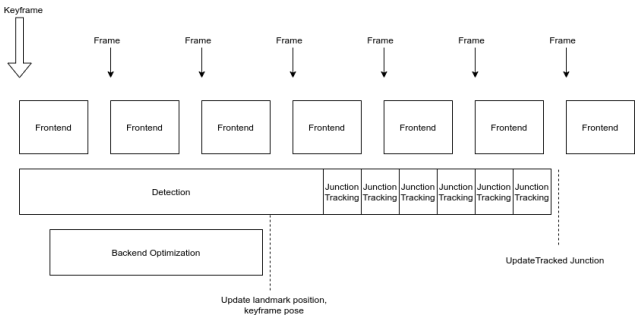


Figure 8: **Suggested Framework to Hide Detection Latency** By running neural network detection on a separate thread in parallel, tracking junctions of lagged frames, and merging them with the frontend LCNN latency can be hidden.

4. Results

Our code and demo video can be found here.

5. Future Works

SLAM with L-CNN can be potentially realized in real-time in several ways. A recent study claims that pruning synapses and neurons of convolutional neural networks can enable efficient inference and results in a few times speed-up [19]. We can apply pruning to L-CNN to reduce latency itself.

Tracking and pose estimation in the frontend depend on the detection result to proceed. This will cause large latency when junction detection is called at the frontend. To resolve the detection latency issue, latency hiding mechanism can be used. We illustrate a potential framework in Figure 8. We can put neural network detection on a separate thread and let the frontend only track previously detected junctions. We can buffer input images while doing neural

network detection and track them afterward. Since junction tracking can run faster than the frontend it will catch up with the frontend and merge tracked junctions with the frontend.

Furthermore, because of the GPU architecture, processing a single image or multiple images does not change the processing time significantly. Processing frames of different past time steps altogether, and registering all detected junctions in different time steps in a map leads to higher efficiency.

6. Conclusion

In this paper, we integrated L-CNN on a SLAM system as a proof-of-concept. We added junction detection, 3D map initialization, junction tracking, and pose estimation into a visual SLAM system. We identified the challenge in data association caused by the sensitivity of L-CNN to small image change and designed an optical tracking technique to address this. We also proposed a pose estimation method compatible with our junction tracking method.

As the bottleneck analysis shows that the off-the-shelf L-CNN has unacceptable latency in loading pretrained weights, we keep L-CNN on memory after loading pretrained weights and use IPC between SLAM and L-CNN. The IPC is designed keeping minimal latency in mind. It turned out that the L-CNN powered SLAM system still has unacceptable latency for real-time use, but it has possibilities for improvement. One way is to prune neurons and synapses of L-CNN for faster inference. The other suggestion is to use non-blocking IPC and to detect junctions for past frames, which are registered on the map later. As the GPU architecture has higher efficiency in processing images as a batch, a batch of past images can be transferred and processed by L-CNN altogether.

The proposed visual SLAM system with detected junctions has the potential to be more robust and accurate in low texture environments. The reconstructed wireframe map can be useful for AR and VR applications. It can also produce a simpler mathematical representation of obstacles, and thus, it is beneficial for robotics applications such as control and planning for obstacle avoidance in real-time.

References

[1] Baichuan Huang, Jun Zhao, and Jingbin Liu. A survey of simultaneous localization and mapping with an envision in 6g wireless networks. *arXiv:1909.05214 [cs]*, Feb 2020. arXiv: 1909.05214. 1

[2] Cesar Cadena, Luca Carlone, Henry Carrillo, Yasir Latif, Davide Scaramuzza, Jose Neira, Ian Reid, and John J. Leonard. Past, present, and future of simultaneous localization and mapping: Towards the robust-perception age. *IEEE Transactions on Robotics*, 32(6):1309–1332, Dec 2016. arXiv: 1606.05830. 1

[3] Takafumi Taketomi, Hideaki Uchiyama, and Sei Ikeda. Visual slam algorithms: a survey from 2010 to 2016. *IPSJ Transactions on Computer Vision and Applications*, 9(1):16, Dec 2017. 1

[4] Raul Mur-Artal and Juan D. Tardos. Orb-slam2: An open-source slam system for monocular, stereo, and rgb-d cameras. *IEEE Transactions on Robotics*, 33(5):1255–1262, Oct 2017. 2

[5] Masashi Yokozuka, Shuji Oishi, Thompson Simon, and Atsuhiko Banno. Vitamin-e: Visual tracking and mapping with extremely dense feature points. *arXiv:1904.10324 [cs]*, Dec 2019. arXiv: 1904.10324. 2

[6] R. Gomez-Ojeda, F. Moreno, D. Zuñiga-Noël, D. Scaramuzza, and J. Gonzalez-Jimenez. Pl-slam: A stereo slam system through the combination of points and line segments. *IEEE Transactions on Robotics*, 35(3):734–746, 2019. 2

[7] R. Grompone von Gioi, J. Jakubowicz, J. Morel, and G. Randall. Lsd: A fast line segment detector with a false detection control. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 32(4):722–732, 2010. 2

[8] Y. Zhou, H. Qi, and Y. Ma. End-to-end wireframe parsing. In *2019 IEEE/CVF International Conference on Computer Vision (ICCV)*, pages 962–971, 2019. 2

[9] Raul Mur-Artal, Jose Maria Martinez Montiel, and Juan D Tardos. Orb-slam: a versatile and accurate monocular slam system. *IEEE transactions on robotics*, 31(5):1147–1163, 2015. 2

[10] K. Huang, Y. Wang, Z. Zhou, T. Ding, S. Gao, and Y. Ma. Learning to parse wireframes in images of man-made environments. In *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 626–635, 2018. 2

[11] Qinrui Yan Xiang Gao, Tao Zhang and Yi Liu. *14 Lectures on Visual SLAM: From Theory to Practice*. Publishing House of Electronics Industry, 2017. 3

[12] R. Kümmerle, G. Grisetti, H. Strasdat, K. Konolige, and W. Burgard. G2o: A general framework for graph optimization. In *2011 IEEE International Conference on Robotics and Automation*, pages 3607–3613, 2011. 3

[13] https://docs.opencv.org/4.3.0/df/d21/classcv_1_1GFTTDetector.html. 3

[14] https://docs.opencv.org/4.3.0/dc/d6b/group__video__track.html. 3

[15] Bruce D Lucas, Takeo Kanade, et al. An iterative image registration technique with an application to stereo vision. 1981. 3

[16] Jianbo Shi and Carlo Tomasi. Good features to track. In *Computer Vision and Pattern Recognition*, pages 593–600, 1994. 3

[17] https://zeromq.org/. 5

[18] https://developers.google.com/protocol-buffers. 5

[19] Pavlo Molchanov, Stephen Tyree, Tero Karras, Timo Aila, and Jan Kautz. Pruning convolutional neural networks for resource efficient inference. *arXiv preprint arXiv:1611.06440*, 2016. 5