# Implementing Large-Scale Direct Monocular SLAM

Raven Pillmann [redacted]

University of Minnesota, Twin Cities

**Abstract.** Simultaneous localization and mapping (SLAM) refers to a class of algorithms that discern information of an agents location and surrounding environment from visual imagery, used largely in autonomous robotics. Several approaches to SLAM are first discussed and compared. LSD-SLAM is referenced in greater detail, and key components to LSD-SLAM are implemented to varying degrees of success. Finally, future implementation details are discussed.

**Keywords:** SLAM, Robotics, Navigation, Autonomous

## 1 Introduction

Robots have and will continue to become a large part of our everyday experience. From assembling parts on the manufacturing line to transporting passengers, robots are a massive portion of the technological boom over the last few decades. Self-driving vehicles, including cars, trucks, and drones, are set to come to the forefront within the next decade. The need for robots to understand and interact with their environment, therefore, is higher than ever.

Simultaneous localization and mapping, also known as SLAM, is a set of algorithms that attempt to both discern the location of an agent as well as its surrounding environment from visual imagery. Self-driving cars, for example, may employ cameras to collect series of images as they drive. These images can be used to produce information pertaining to the car's location – where is the car in relation to the world – as well as its surrounding environment – how far away are the other cars, where is the stop sign, has the car driven in a loop, and so on. This information is essential for any autonomous vehicle to navigate unknown spaces.

SLAM becomes more important as information about the environment becomes more sparse. A sole self-driving car on a test road that has been completely mapped beforehand may be able to rely on GPS to determine which direction to turn at certain points, how far away it is from the sides of the road, and if it has completed a full loop of the track. A drone flying through a heavily wooded mountainside looking for a lost hiker, on the other hand, needs to rely on the visual information surrounding it to determine whether there are trees in the way and whether it has already searched a space.

The difficulties surrounding SLAM are what make it an interesting problem. To a certain extent, localization and mapping are two problems that rely on the solution of the other. How can an object be localized without knowing the distance of itself to its surrounding environments? How can the positions of surrounding objects be known without knowing where the agent is in relation to them? These complexities require an iterative approach, which is the basis of a number of SLAM approaches.

## 2 Background

Various SLAM approaches have been used historically, beginning in the 1980s. According to [2], SLAM began in 1986, when researchers at the IEEE Robotics and Automation Conference began discussing probabilistic estimation of localization in robotics. [14] proposed the stochastic map, a representation of spatial information that could be filtered and improved upon with the presence of more information, which would form the basis for probabilistic filtering in SLAM. In this paper, a class of SLAM algorithms known as EKF-SLAM was introduced, which took advantage of extended Kalman filters to create the stochastic map. Further, [1] shows that SLAM will converge to a correct solution for the stochastic map, and that the accuracy of depth estimation is solely based on the initial values of the estimation.

SLAM approaches can be distinguished by a number of factors, including the features they use to construct maps and the number of cameras used. Early SLAM algorithms were based around feature-tracking. These kinds of SLAM algorithms would first find features of interest through some feature extraction algorithm like SIFT and estimate the depth using these features. In [12], FastSLAM was introduced as a computationally efficient improvement to previous SLAM algorithms, which took advantage of Kalman filtering. This new algorithm scaled logarithmically with the number of features, making it more efficient than previous approaches.

Feature-based approaches allow for computationally efficient methods. The number of points in an image that need to be considered is reduced from every pixel in the image to a subset of features based on any number of feature-detection algorithms. However, these methods, by definition, lose information by limiting the number of pixels in an image that are used to reconstruct the relative position of two images, and therefore limit the amount of information used in depth estimation. The resulting maps may be very sparse as a result.

Direct, or feature-less, approaches to SLAM attempt to combat this loss by estimating pose with all pixels shared between two images. [10] proposes a dense SLAM algorithm using RGB-D cameras, using photometric and depth error of all pixels to create a more accurate pose estimation algorithm.

SLAM algorithms also differ by whether one or two cameras are used, referred to as monocular and stereo approaches. Generally speaking, two mounted cameras with a fixed rotation and translation between them simplify the depth estimation problem, as depth can be estimated once focal length, baseline, and disparity are known. [8] succeeds at using a single stereo camera rig to recreate a map of its environment without the use of additional lasers or sensors, a technique which is robust to varying terrain.

[11] proposed a constant-time stereo SLAM algorithm. Specifically, this paper introduces the representation of a global map as a series of relative locations, and in this process reduces the computational complexity associated with SLAM. Loop closure, a difficult problem faced by SLAM algorithms, is shown to be attainable without global map optimization: rather, this paper detected a loop closure by comparing new images to previous images, which is a more efficient approach.

An essential component of SLAM techniques is depth estimation. Many early techniques revolved around trying to find metric information pertaining to depth. [7] addresses this issue by showing that relative depth information can be obtained without the need of a stereo camera system. Indeed, by using two sequential images with shared points, different transformations (such as affine or projective transformations) can be recovered that reveal information pertaining to the relative rotation and translation between the two images. This information can then be used to recover depth. The implication of this work is that depth can be obtained not just from calibrated stereo camera rigs, but indeed from sequential images taken with a single camera. This is the basis of monocular SLAM algorithms. [13] additionally succeeds at creating a supervised learning technique in which depth is estimated from a single image using a Markov Random Field.

[6] proposes a technique for depth estimation refinement. In particular, this paper introduces a technique in which the pixels with a non-negligible gradient in a certain image are refined continuously by additional images. The technique introduced in this paper is computationally efficient, with the authors demonstrating its ability to run real-time on a CPU, making it a potentially key technique for real-world SLAM applications.

LSD-SLAM, as introduced in [5], produces a large-scale map of the world through direct techniques and a single camera. The algorithm runs in real-time on a CPU, as demonstrated in the paper by the authors success in running the algorithm live on a smartphone. Its efficiency, semi-dense map creation, and use of a single camera make LSD-SLAM a popular SLAM algorithm.

## 3 LSD-SLAM's Approach to SLAM

Large Scale Direct Monocular SLAM is distinguished from other forms of SLAM in that it attempts to build a large model of the environment using only one camera by creating a semi-dense map that relies in gradient information rather than features, which allows it to take advantage of more pixels than feature-based SLAM when deriving transformations between two images.

LSD-SLAM generally consists of two components, which can be broken down into several sub-components. The goal of LSD-SLAM is to take a series of consecutive images and produce keyframes. Each keyframe consists of depth information of the pixels of the image that keyframe represents, as well as its relative rotation and translation from the previous keyframe.

The first image in the input sequence is taken to be the first keyframe, with identity rotation and translation and randomly assigned inverse depth for each pixel. Using Gauss-Newton Minimization, the relative rotation and translation from the current keyframe to the next image in the sequence is obtained. If the magnitude of the rotation and translation is above a certain threshold – if the current image is too far away from the previous image – the current image is designated as the new keyframe. Otherwise, we can continue the process of finding the relative rotation and translation of each sequential image from its current image and using it to find the position of the current image from the last keyframe by composing all transformations in between.

The second component of LSD-SLAM is depth refinement. During each iteration, LSD-SLAM attempts to refine the depth information of the current keyframe by estimating the mean and variance of the inverse depth of each pixel in the current keyframe. This is done by first searching for corresponding points in the current image to points in the current keyframe, finding the disparity between them, and triangulating the distance from the current keyframe. This means that the current keyframe's depth estimate is refined only until a new keyframe is created, after which the new keyframe's depth estimate is refined.

Finally, once all images are used to create a series of keyframes, the keyframes are placed into a global map, which is created by using the relative position of each sequential keyframe to the last and taking the depth estimates of each pixel in those keyframes.

### 3.1 The SE3 Lie Algebra

Lie groups and algebras are used to represent transformations in many robotics applications. Several lie groups and algebras exist for different purposes. In the LSD-SLAM case, the SE3 lie group and algebra is used, which consists of six variables (three for rotation and three for translation) and is suitable for rigid body motion. The benefit of using a lie algebra to represent the transformation between two images rather than a rotation and translation matrix is that, whereas rotation matrices require additional constraints to ensure that all columns are orthonormal, the lie algebra representation can easily be calculated without this check.

Lie groups and algebras consist of logarithmic and exponential mappings. Exponential mappings map from a rotation and translation matrix to its corresponding 6x1 lie-algebra, and an logarithmic mapping maps from the 6x1 lie-algebra to its corresponding rotation and translation matrix. More information can be found at [3].

## 3.2 Gauss-Newton Minimization

Gauss-Newton Minimization is an iterative optimization algorithm used to approximate the best warping transformation to get from one image to the other. In the case of LSD-SLAM, the warping is represented by the SE3 lie-algebra $\xi$. Each iteration, the current rotation and translation values are used to warp each of the previous image's pixel locations to their corresponding location in the current image. The difference in pixel intensity between the corresponding points (the intensity at the original location in the first image and the intensity of the mapped location in the second image) is defined as the residual $r$, which is then used to compute the jacobian $J_r$ and calculate the derivative of rotation and translation $\delta\xi$ (1). This derivative is then applied to the current lie-algebra representation of the rotation and translation to make a new approximation (2). This is repeated until convergence.

$$\delta\xi = -(J_r^T J_r)^{-1} J_r^T r \tag{1}$$

$$\xi =: \delta\xi \circ \xi \tag{2}$$

One main problem associated with monocular SLAM algorithms is scale drift. Because exact depth is not known but rather only relative depth is recovered, the scale of depth is important to keep relatively constant. To prevent scale drift from occurring, and to be robust to errors that may be caused by reflections or other lighting obscurities, [5] employs a Huber norm which penalizes both the photometric and depth residuals.

$$Error = \sum_{pixel} \| \frac{r_{photometric}^2}{\sigma_{photometric}^2} + \frac{r_{depth}^2}{\sigma_{depth}^2} \|_\delta \tag{3}$$

Where $r_{photometric}$ are the photometric residuals, $\sigma_{photometric}^2$ are the photometric variances, $r_{depth}$ are the depth residuals, and $\sigma_{depth}^2$ are the depth variances. In [5], this Huber norm is calculated after every warping update, and warping updates are finished once this error converges.

### 3.3 Inverse Depth Estimation

Given two corresponding pixels, each from a different image, it is possible to triangulate the depth of the point the pixels represent. This can intuitively be found as

$$Z = \frac{f * b}{d} \tag{4}$$

Where $Z$ is the depth estimate, $f$ is the camera's focal length, $b$ is the baseline between two camera positions, and $d$ is the disparity between corresponding pixels (defined as the pixel distance between the pixel in one image and the corresponding pixel in the second image).

In LSD-SLAM, inverse depth ($\frac{1}{Z}$) is calculated rather than proper depth, which prevents issues arising from infinite depth objects, which are represented as an inverse depth of 0 rather than infinity.

Inverse depth estimation follows several steps, as outlined in [6], and relies on epipolar geometry. As a note, an epipolar line is defined as a line in one image that the corresponding pixel of a point seen in the other image must lie upon. Therefore, by searching along the epipolar line (in image B) produced by a pixel in image A, one can find image A's pixel's corresponding pixel in image B.

To estimate depth, a subset of "good pixels" is found by looking at the geometric and photometric errors, which makes depth estimation much more computationally efficient by reducing the overall number of calculations required. Geometric error measures how far off the epipolar line is from the true corresponding point, and photometric error is the difference in intensity between a pixel in the first image and the pixel it maps to in the second image.

Once a subset of "good" pixels in the first image is found, the epipolar lines in the second image are computed by using a fundamental matrix, defined by the intrinsic camera parameters, rotation, and translation. For each pixel in the first image, its corresponding point is found in the second image by looking at each pixel along the epipolar line in the second image, as well as the pixel's surrounding four pixels, and finding the closest match in terms of intensity values.

Finally, the depth estimate for each pixel in the keyframe, which is represented as a normal distribution, is updated with a Kalman filter, where both inverse depth and variance of inverse depth are updated based only on the previous estimates. This will be covered in more detail in the next section.

## 4 Implementation

Implementing LSD-SLAM requires many components to cover all of the various warping and depth estimation tasks. As such, a version of LSD-SLAM was implemented in which components were broken into smaller, more testable pieces.

Retrospectively, this was a difficult way to approach the problem: due to the iterative nature of the algorithm, depth estimates are somewhat random for the first few keyframes, as noted in [5]. This means that testing becomes somewhat difficult as a result, as components built before multiple keyframe integration is completed may be performing correctly despite randomness in the output. Therefore, testing this implementation was difficult and remains incomplete.

Because SLAM at its core revolves around depth estimation, I first worked to implement a classic monocular depth estimation technique suggested by [7]. This was done by creating a pipeline that, given two images, first found corresponding features between both images using the SIFT feature extraction algorithm. Using RANSAC, four optimal correspondences were found between the two images. Shaping these correspondences into $A$, the null space of $x$ was found to solve for the components for the fundamental matrix.

$$Ax = 0 \tag{5}$$

Singular value decomposition was used to deconstruct the fundamental matrix found with the null space, remove the third singular value, and reconstruct to find a new fundamental matrix with rank 2. This fundamental matrix represents the projective transformation between both images. Then, depth could be estimated with equation (4).

With this as a proof-of-concept, I worked to implement warping via Gauss-Newton Minimization, which would provide the projective transformation between two images using all shared pixels in the image rather than a handful of features. Per [5], this began by implementing the lie-algebra transformations for pose concatenation, exponential mapping, and logarithmic mapping. I wrote tests to ensure that certain lie-algebra pose concatenations would result in the same transformation as the multiplication their corresponding matrices.

With the lie-algebra transformations implemented, I implemented error calculation, which required implementing photometric residuals and their jacobian. The residuals were calculated on a pixel-by-pixel basis, in which the pixels from the previous image were transformed from pixel to real-world coordinates, projected to a three-dimensional point by taking into account the current depth estimate for that pixel, applying the current estimate for rotation and translation, and projecting back down to the current image by dividing the $x'$, $y'$, and $z'$ components by $z'$. The residual then becomes the difference in the intensity of the resulting pixel in the current image and its corresponding pixel in the previous image. Additionally, each row of the jacobian is provided in [4]

$$J_r = \frac{1}{z'}(\nabla I_x f_x \nabla I_y f_y) \begin{pmatrix} 1 & 0 & -\frac{x'}{z'} & -\frac{x'y'}{z'} & (z' + \frac{x'^2}{z'}) & -y' \\ 0 & 1 & -\frac{y'}{z'} & -(z' + \frac{y'^2}{z'}) & \frac{x'y'}{z'} & x' \end{pmatrix} \tag{6}$$

Where $\nabla I_x$ and $\nabla I_y$ are the x and y direction image gradients and $f_x$ and $f_y$ are the focal length in the x and y direction.

With the residuals and jacobian computed, I implemented the change in pose as

$$\delta\xi = -(J_r^T J_r)^{-1} J_r^T r \tag{7}$$

Where $J_r$ is the stacked jacobians for each pixel and $r$ is the stack residuals for each pixel. $\delta\xi$ is then left-multiplied with the current lie-algebra approximation of the rotation and translation to get the new approximation, per equation (2). Due to time constraints, I did not implement the Huber norm for the error function as outlined in the paper. Rather, I used the sum of squares of residuals as the error function to determine when this warping procedure was completed.

To implement depth estimation, I relied largely on the work of [6], which was recommended in [5]. Additionally, I referenced an external implementation for depth estimation [9]. First, I implemented code to determine good pixels based on the criteria set forth in [6]. Then, with the current estimate of rotation and translation from the warping procedure, as well as the intrinsic camera parameters $K$, I found the fundamental matrix as

$$F = {K^{-1}}^T [t]_X R K^{-1} \tag{8}$$

Where $[t]_X$ is the skew symmetric matrix for the translation from image to keyframe and $R$ is the rotation from image to keyframe. With the fundamental matrix $F$, the epipolar line in the keyframe, $l_u$, for each point in the image, $u$, can be found as

$$l_u = Fu \tag{9}$$

With these epipolar lines, for each pixel, I implemented a sequential search along the line for the best matching pixel. For matching, I followed the suggestion of [5] to compare the two points on both sides of the pixel being searched for.

Once inverse depths were found, I either created or updated the inverse depth estimate based on whether the keyframe already had inverse depth estimates or not. To create a depth estimate, I used the inverse of equation (4). To update an inverse depth estimate, I used the Kalman filter update step provided in [6]

$$d = \frac{\sigma_p^2 d_o + \sigma_o^2 d_p}{\sigma_p^2 + \sigma_o^2} \tag{10}$$

Where $d$ is the new depth estimate of the pixel, $\sigma_p^2$ and $d_p$ refer to the observed variance and depth of the pixel in the current iteration of depth refinement, and $\sigma_o^2$ and $d_o$ refers to the previous estimate of variance and depth. Similarly, the new variance is estimated with

$$\sigma^2 = \frac{\sigma_p^2 \sigma_o^2}{\sigma_p^2 + \sigma_o^2} \tag{11}$$

Using these filter updates, the depth is refined with each new rotation and translation. Iteratively, Gauss-Newton Minimization and inverse depth estimation are applied until the rotation and translation between the keyframe and image is too large. The size of rotation and translation is taken as the norm of the lie-algebra associated with the rotation and translation, and compared to a pre-defined threshold. I used $10^{-7}$ as the threshold, simply from observing a jump in error around that magnitude of lie-algebra.

Ultimately, I did not add functionality to incorporate new keyframes once the threshold is met. Future work will be discussed in part 6 of this paper.

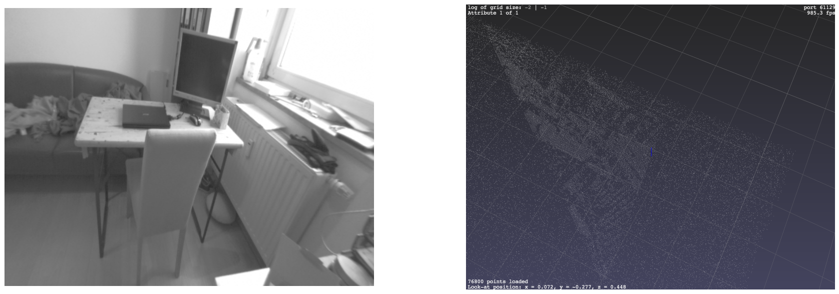
## 5 Experimental Results

To test my implementation, I used a dataset of images provided by [5]. Overall, I had varying degrees of success, but more work needs to go into testing the resulting depth estimates. Due to the large amount of code involved, how interwoven each step of the process is, and that [5] says that depth estimates are not very reliable until a few keyframes in, it was very difficult to debug my code.

I visually verified my 8-point algorithm based depth estimation by checking a depth-intensity map. This was the most success part of my project, but because it is not part of LSD-SLAM, I do not consider it a success for my implementation.

I wrote several tests for my lie-algebra transformations, each of which passed. Image warping, however, faced several issues. In particular, warping from the first to second image of the sample video went well in each case, with error steadily decreasing for several iterations before error increased. However, in all subsequent warpings (second to third image, third to fourth, and so on), error always increased during the first iteration. I was never able to find the source of error in this, but it resulted in poor transformations from the keyframe to the current image as the current image deviated further away. To test the warpings, I computed the fundamental matrix from the rotation and translation as described in equation (6) and checked visually whether the corresponding points from the first image laid along the epipolar line in the second image as defined by the fundamental matrix. However, this type of testing was subjective and failed when checking the transformation between two images relatively far apart temporally.

Finally, I checked the general ability of the algorithm to estimate the depth of various points. For the single keyframe I recovered, I projected the points into 3D by using the depth information estimated from the algorithm. The resulting point clouds were somewhat poorly formed: although discernable shapes were recovered from the algorithm, the depth estimated for each pixel has not been verified. Figure 1 consists of the original image in the keyframe, as well as the resulting point cloud. Note that the grating of the furnace can be seen in the bottom left portion of the image, and that surfaces in the original image (such as the desk) are visible as dense point cloud regions as well. Because I only implemented depth estimation for a single keyframe, and because, according to [5], this algorithm converges to true estimates only after a few keyframes are added, this point cloud formation may, in part, be improved by adding more keyframes.



(a) Original Image (b) Point Cloud Projection

Fig. 1: Original Image and Point Cloud

## 6 Conclusion and Future Work

LSD-SLAM is a popular SLAM algorithm for its efficiency, use of a single camera, and semi-dense mapping. However, it is also an intricate and complex algorithm to implement. In this paper, key components of LSD-SLAM were implemented to varying degrees of success, although issues persist. First, lie-algebra transformations were implemented. Then, warping was implemented using [5], the introductory paper for LSD-SLAM. Depth estimation was also implemented following [6].

My implementation was able to produce depth information for a single keyframe, to a certain extent. However, a single keyframe is not useful for SLAM purposes. Future work will begin with implementing the addition of sequential keyframes. This will involve finding corresponding points between two keyframes, initializing inverse depth estimates from the previous keyframe's estimates, and continuing to warp images and estimate depths in the new keyframe. Additional considerations will need to be taken into account concerning each subsequent keyframe's rotation and translation from the last.

Furthermore, [5] employs a global optimization technique to detect loops. Future work will involve detecting loops, which will necessitate robustness to scale drift. To solve for this, I will implement the Huber norm error function, defined in (3), after implementing geometric residual and variance calculations. The geometric residuals are fairly straightforward to calculate, but I found [5]'s description of calculating geometric variance confusing, although I understand that they use propagation of uncertainty, a technique that estimates the covariance of a function with the covariance of the input to that function.

Finally, SLAM is most helpful when it is efficient enough to run in real-time. In the future, I will implement LSD-SLAM with this in mind, and plan to use C++ rather than Python, which is a slower language to run. Ideally, the performance gains by this will make it possible to run LSD-SLAM on a smartphone, as [5] suggests is possible.

LSD-SLAM continues to be a popular version of the SLAM algorithm. However, a plethora of approaches to SLAM exists, each with varying usefulness and implementation difficulties. Regardless, SLAM continues to be an exciting component of autonomous navigation, and will maintain its role as a key component of future robotics.

## References

1. Dissanayake, M., Newman, P., Clark, S., Durrant-Whyte, H., Csorba, M.: A solution to the simultaneous localization and map building (slam) problem. IEEE Transactions on Robotics and Automation **17**(3), 229–241 (2001)
2. Durrant-Whyte, H., Bailey, T.: Simultaneous localization and mapping: part i. IEEE Robotics and Automation Magazine **13**(2), 99–110 (2006)
3. Eade, E.: Lie groups for 2d and 3d transformations (2017), `http://ethaneade.com/lie.pdf`
4. Engel, J.: Direct real-time slam. University Lecture (2016)
5. Engel, J., Schöps, T., Cremers, D.: Lsd-slam: Large-scale direct monocular slam. European Conference on Computer Vision pp. 834–849 (2014)
6. Engle, J., Sturm, J., Cremers, D.: Semi-dense visual odometry for a monocular camera. IEEE International Conference on Computer Vision (2013)
7. Faugeras, O.: What can be seen in three dimensions with an uncalibrated stereo rig? European Conference on Computer Vision (1992)
8. Garcia, M., Solanas, A.: 3d simultaneous localization and modeling from stereo vision. IEEE International Conference on Robotics and Automation (2004)
9. Haubenstock, M.: Prob depth (2017), `https://github.com/geoeo/Prob_Depth`
10. Kerl, C., Sturn, J., Cremers, D.: Dense visual slam for rgb-d cameras. IEEE/RSJ International Conference on Intelligent Robots and Systems (2013)
11. Mei, C., Sibley, G., Cummins, M., Newman, P., Reid, I.: A constant-time efficient stereo slam system (2009)
12. Montemerlo, M., Thrun, S., Koller, D., Wegbreit, B.: Fastslam: A factored solution to the simultaneous localization and mapping problem. AAAI pp. 593–598 (2002)
13. Saxena, A., Chung, S., Ng, A.: Learning depth from single monocular images (2006)
14. Smith, R., Self, M., Cheeseman, P.: Estimating uncertain spatial relationships in robotics. Autonomous Robot Vehicles pp. 167–193 (1990)