Vision-Enhanced Lidar Odometry and Mapping

Daniel Lawrence Lu

August 1, 2016

Submitted in partial fulfilment of
the requirements for the degree of
Master of Science

Robotics Institute
Carnegie Mellon University
Pittsburgh, PA 15213

CMU-RI-TR-16-34

Thesis committee
Prof. George Kantor
Prof. Michael Kaess
Ji Zhang

Abstract

Vision-Enhanced Lidar Odometry and Mapping (VELO) is a new algorithm for simultaneous localization and mapping using a set of cameras and a lidar. By tightly coupling sparse visual odometry and lidar scan matching, VELO is able to achieve reduced drift error compared to using either one or the other method. Moreover, the algorithm is capable of functioning when either the lidar or the camera is blinded. Incremental Smoothing and Mapping is used to refine the pose-graph, further improving accuracy. Experimental results obtained using the publicly available KITTI data set reveal that VELO achieves around 1% translation error with respect to distance travelled, indicating it has comparable performance to state-of-the-art vision- and lidar-based SLAM methods.

Contents

1 Introduction

> If we could first know where we are, and whither we are tending, we could then better judge what to do, and how to do it.
>
> Abraham Lincoln (1858)

1.1 Motivation

Mobile robots and autonomous vehicles are poised to become a central part of our everyday lives. Self-driving cars can improve quality of life by preventing accidents and by freeing up time spent commuting. Unmanned mobile robots offer unprecedented utility in all manner of applications. Robots can explore remote or hazardous areas, transport goods, or perform manual labour such as cleaning, farming, and construction.

In order to navigate and to perform any task, an autonomous mobile robot must first be able to determine its own position and orientation (together, the pose) with respect to its surroundings. This problem is known as ego pose estimation or localization.

If there is no prior knowledge of the environment, the problem appears to be a chicken-and-egg problem. Without a map, one cannot know where one is; without knowing one's location, it seems difficult to build a map. The paradox is resolved by simultaneous localization and mapping (SLAM). The history of robotics is rich with different SLAM strategies using a variety of different sensors. This thesis builds upon work in this area by contributing a new method using vision and lidar which improves upon the state-of-the-art.

The camera and the lidar (light radar) are sensors with complementary properties. A camera takes images with high angular resolution and reliable intensity measurements, but does not provide any range information. Multiple cameras (stereo vision) can provide range information by triangulation, but the quality of the range measurement deteriorates rapidly as range increases.

A lidar is a range-and-bearing sensor which measures range in any direction by sending pulses of infrared light using lasers and timing the reflection. By rotating one or more lasers and performing measurements rapidly, a 3D point cloud of the surroundings is generated. A typical modern lidar suitable for mobile robots provides accurate range information up to 150 m and often a 360° field of view, but only at points which are sampled with sparse and possibly uneven angular resolution. Surveying lidars may be able to pro-

duce dense angular resolution, but do not have sufficient acquisition rate to be used in a mobile setting.

Whereas a camera is a passive sensor sensitive to ambient lighting conditions and incapable of functioning in darkness, a lidar works in most lighting conditions including complete darkness, as it emits its own light. Conversely a lidar is slightly less reliable in strong daylight when ambient light may overpower its lasers. Also, lidar data tends to have holes at regions beyond the maximum range and on dark or specular surfaces.

The strengths of each sensor may be used to compensate for the weaknesses of the other when the two are combined in an effective way. Ideally, we would like the camera's dense angular resolution to compensate for the sparsity of the lidar, and the accurate range information of the lidar to resolve the ambiguity in camera depth perception. Moreover, a visual-lidar system should perform well even when either one sensor fails. For example, in addition to failure in darkness, a vision-based system may struggle in a featureless environment such as an overcast day with snow on the ground, where it is difficult to determine scene geometry by vision alone as the colour is the same regardless of surface normal. On the other hand, a lidar approach that depends on aligning geometric point clouds may fail in geometrically degenerate scenes such as a straight tunnel, even though a camera may be able to pick up visual cues on the tunnel walls to help localize.

Figure 1: Degenerate situations. On the left, the scene is visually rich but geometrically symmetric, making it difficult to estimate pose by aligning lidar point clouds. On the right, the scene is visually featureless, even though the snow-covered parked cars are obstacles easily detected using lidar. (Image credit: Left, Thomas Wolf, CC-By-SA 3.0 Unported license; Right, Anna Kucsma, CC-By-SA 2.5 Generic license)

An alternative to the lidar is the radar, which uses radio waves instead of light pulses. Radar has the advantages of being cheap and impervious to weather conditions such as rain, snow, and fog which may obscure vision. However, the fidelity of radar in both the

angular and range direction is extremely poor compared to lidar. Consequently radar is usually employed in obstacle detection rather than pose estimation. Sonar is another alternative sensor, using sound waves instead of radio or light waves. The fidelity of sonar is even poorer, so sonar as a primary sensor is usually relegated to applications where radar and light are severely attenuated, such as undersea exploration. This notwithstanding, the low cost of ultrasonic range sensors means that such sensors can be found as obstacle detection sensors on terrestrial vehicles as well.

Apart from using cameras and range sensors, other sensors may be used for localization. The Global Positioning System (GPS) is a widely available constellation of satellites used for geolocation by multilateration. However, urban, mountainous, underground, forested, or extraterrestrial environments suffer from degraded or inaccessible GPS signals. Moreover, civilian GPS systems tend to have poor accuracy and update rate, and cannot usually be used to determine orientation.

An inertial measurement unit (IMU) is another popular sensor for localization. It measures linear acceleration and rate of rotation, which are integrated to provide an estimate of pose over time. Dead reckoning using an IMU is prone to more drift over time compared to other methods, as biases in the sensor accumulate quadratically. In addition, dead reckoning systems cannot construct a map of the environment to perform loop closure when revisiting known areas, or to localize when the robot has been unexpectedly moved. As such, inertial navigation systems tend to be integrated with other sensors.

To obtain the best possible pose estimate, it is desirable to construct a pose estimation system that does not fully rely on either the GPS or the IMU. The general problem of pose estimation using cameras integrated with range sensors is known as RGB-D pose estimation, as a camera typically produces colour data in the red, green, and blue channels (RGB) and the range sensor provides depth (D). Confusingly, the term “RGB-D” is sometimes used even if the camera is monochrome.

RGB-D pose estimation has been widely studied for visual-lidar systems as well as other sensors, such as time-of-flight cameras and structured light systems like the Microsoft Kinect. Compared to these alternative range sensors, lidar tends to have greater range and perform more accurately and reliably in outdoor environments.

1.2 Problem Statement

Given a sequence of images provided by one or more cameras and a sequence of point clouds generated by one or more lidars synchronized with respect to the cameras, the objective is to recover the pose of the robot in 6 degrees of freedom in each frame, with respect to its initial pose, without using a prior map. A secondary objective is to create a globally consistent 3D model of the perceived environment.

It is assumed that the environment is mostly static. Furthermore, it is assumed that the extrinsic calibration between the lidar and the cameras, as well as the intrinsic calibration within said lidar and cameras are known with high accuracy.

We note that the lidar point clouds may be distorted due to the motion of the vehicle while the lidar sensor is scanning. Although it is possible for some algorithms to remove this distortion [65], for the purpose of this thesis, we assume that such distortion has been mostly removed by a preprocessing step using an external pose source such as an inertial measurement unit. Further discussion to compensate for motion distortion is presented in section 8.1.

1.3 Contribution

I present a new pose estimation algorithm named Vision-Enhanced Lidar Odometry and Mapping (VELO). VELO tightly couples sparse visual odometry and lidar scan matching in a single optimization problem. As a natural consequence, VELO is capable of functioning if either the cameras or the lidar is completely disabled, and naturally takes advantage of the full field of view of both the lidar and the cameras.

A new feature tracking pipeline is proposed which combines Lucas-Kanade-Tomasi sparse feature tracking with fast retina keypoint descriptors to reject outliers. A new method is proposed to associate 2D keypoints detected in camera images with 3D position in the lidar point cloud.

Experiments with the public data set KITTI as well as our own data demonstrate the performance of VELO compared with the state of the art.

Real time performance is not a priority for the purposes of this thesis, however, a discussion of running time and efficiency considerations is presented. It is anticipated that, in the near future, computers will be able to run this algorithm in real time.

1.4 Notation

Unless otherwise specified, vectors are typeset in bold italic, e.g. $\boldsymbol{V}$ or $\boldsymbol{v}$, scalars are non-bold italic, e.g. k, and matrices are bold non-italic, e.g. $\mathbf{M}$. Sets are denoted in calligraphic, such as $\mathcal{S}$. A 3D point in space is written bold italic uppercase, e.g. $\boldsymbol{P}$, and its 2D projection is written in bold italic lowercase, e.g. $\boldsymbol{p}$. The 3×3 skew-symmetric matrix representation of the cross product by a 3×1 vector $\boldsymbol{t}$ is denoted as $[\boldsymbol{t}]_{\times}$.

The coordinate frame associated with a camera C is denoted in braces, i.e. $\{C\}$. The coordinate frame origin is centered on camera 0. The choice of coordinate system is such that, with respect to the vehicle, the z direction points forward, the y direction points downward, and the x direction points rightward.

2 Background

> The machine does not isolate us from the great problems of nature but plunges us more deeply into them.
>
> Antoine de Saint-Exupéry
> *Terre des Hommes* (1939)

Pose estimation systems can generally be separated into the front end and the back end. The front end, also referred to as *odometry* or *motion estimation*, involves estimating the relative spatial transformation between two frames. Methods to do this using range-and-bearing sensors such as lidar are described in section 2.1 and methods using cameras are described in section 2.2. Then, methods that combine the two sensors are discussed in section 2.3.

Although dead reckoning using frame-to-frame motion provides a good local estimate of the trajectory that the robot has taken, the gradual accumulation of error over time causes inconsistencies if the robot revisits a previously seen area. The back end of pose estimation resolves these inconsistencies to construct a globally consistent map from frame-to-frame measurements. These methods are discussed in section 2.4.

2.1 Lidar Motion Estimation

Lidar motion estimation refers to finding the relative spatial transformation between two frames, based on lidar data collected at those frames. Such techniques typically treat the data as sets of 3D points, also known as point clouds. The transformation between two frames is found by aligning the two point clouds using point set registration, also known in some cases as scan matching or point matching. Let $\mathcal{M}$ and $\mathcal{S}$ be two sets of 3D points: the moving "model" and the static "scene" respectively. Then, point set registration seeks to minimize $\mathrm{dist}(T(\mathcal{M}), \mathcal{S})$ where T is a transformation, and dist is some distance metric. Here, we are usually concerned with only 6 degree of freedom rigid transformations in SE(3), although we point out more complicated parametrizations of transformations may be needed to account for such effects as motion distortion.

Two types of point set registration algorithms exist: local ones that assign point correspondences by proximity, and global ones that establish point correspondences by constructing unique feature descriptors invariant with respect to translation and rotation.

2.1.1 Local point set registration

The most widely known algorithm for point set registration is iterative closest point (ICP) [5][11]. The algorithm operates in two steps: first, every point in $\mathcal{M}$ is assumed to correspond to its closest point in $\mathcal{S}$, and second, the sum of squared Euclidean distances between corresponding pairs of points is minimized through least squares. These two steps are repeated until convergence. To find the closest point, a KD-tree is a data structure which can be constructed from the points in $\mathcal{S}$, allowing the closest point to be queried in logarithmic time [49].

A primary drawback of ICP is that the assumption that each point corresponds to the nearest one can only hold when initial alignment is already very good. Consequently, it is prone to getting stuck in a local optimum, and it is highly sensitive to noise and outliers. To address these issues, a plurality of ICP variants have been proposed, changing every phase of the algorithm: point correspondence, transformation parametrization, cost function, and optimization strategy.

Instead of hard correspondences, that is, associating each point in $\mathcal{M}$ to a single point in $\mathcal{S}$, soft correspondences may be used, where each point in $\mathcal{M}$ has a weighted correspondence to multiple points in $\mathcal{S}$. One probabilistic approach to soft correspondence is to assume that each point in $\mathcal{S}$ is identically distributed according to a Gaussian distribution. While treating $\mathcal{S}$ as a Gaussian mixture model, the likelihood of sampling points $T(\mathcal{M})$ from this Gausian mixture model is maximized. In this sense, the algorithm is an expectation-maximization (EM) algorithm: the expectation step is equivalent to finding soft correspondences by evaluating the Gaussians at the points in $T(\mathcal{M})$, and the maximization step finds the transformation that maximizes the likelihood. Such algorithms typically include an annealing scheme that gradually decreases the variance of the Gaussians as the iteration proceeds: this is so that the algorithm is robust against noise and outliers at the beginning, but converges to an accurate result. EM-ICP [21] and coherent point drift (CPD) [43] are such algorithms. Instead of soft correspondences with Gaussian probability, other methods such as Softassign may be used, as in robust point matching [12].

Even with soft correspondences, minimizing point-to-point distance is prone to misalignments when used with scanning lidars that have nonuniform angular resolution. For example, a typical lidar with an array of lasers spinning in one axis may have much greater resolution in the azimuth direction as opposed to the altitude direction. This produces

artifacts such as rings on the ground that can bias a point set registration algorithm. One solution is to assume the scene to be locally planar. For every point in $\mathcal{M}$, a plane is fit to a small neighbourhood of points in $\mathcal{S}$, and the point-to-plane distance is minimized [11]. More generally, instead of a plane, a multivariate Gaussian may be fit to such a small neighbourhood around the corresponding point, such as in generalized ICP [52]. To further remove the effect of uneven lidar points, the normal distributions transform (NDT) [6][38] divides the space into a voxel grid and fits a multivariate Gaussian to each grid cell. Clustering algorithms such as in segmented greedy cluster NDT are an alternative to a voxel grid [15].

Instead of only treating $\mathcal{S}$ as a probability distribution and maximizing the likelihood of observing points in $\mathcal{M}$, one may treat both point sets as being probability distributions, and then minimize the distance between these two distributions. The choice of distance metric between two distributions includes L2 distance and Kullback-Leibler divergence. Such methods include kernel correlation [60] which aligns distance between various kernel mixture models using gradient descent, the method of Jian and Vemuri [27] that provides a closed form solution to minimize L2 distance between mixtures of spherical Gaussians, and distribution-to-distribution (D2D) NDT [56] which minimizes L2 distance between general multivariate Gaussians (computed in cells of a voxel grid) using gradient descent.

In addition to minimizing geometric spatial distance, the distance in feature space may also be minimized. For example, if colour is available, then three channel colour may be combined with 3D spatial distance to form a 6D minimization metric [28]. Other features include the normal direction and the feature descriptors discussed in the next section.

The transformation of a point set may be parametrized in many ways. Usually, we are concerned only with the 6 degree of freedom rigid transformation in SE(3). However, in some cases, non-rigid transformations are needed. In such cases, thin plate splines can be used [12]. When the lidar is in motion, there can be motion distortion as the points in the point set are perceived at different times. In lidar odometry and mapping (LOAM) [65], a linear interpolation is used to determine the transformation in between the start and the end of a scan.

Having established the cost function, several optimization schemes have been proposed. In the simplest case for ICP [5] as well as more advanced variants like CPD [43], the rigid transformation is found in closed form: the rotation can be found using singular

value decomposition and the translation is simply the mean translation. For additional robustness, ICP can be adapted to the Levenberg–Marquardt algorithm to form LM-ICP [17]. The Levenberg–Marquardt algorithm is also used in LOAM [65]. For distribution-to-distribution methods like kernel correlation [60] or D2D NDT [56], gradient descent with various line search strategies may be used.

If the goal is to register scans from consecutive time steps in a moving vehicle, certain physical constraints can be added to the cost function to improve convergence, such as smoothness of motion and the gravity direction [67].

2.1.2 Global point set registration

Local point set registration algorithms are prone to getting stuck in local optima if the initialization is poor or if the data is noisy. One way to adapt a local point set registration algorithm to achieve global optimality is through branch-and bound scheme, such as in Globally Optimal ICP [63], or stochastic optimization such as genetic algorithms, particle swarm optimization, particle filtering, random sample consensus, and simulated annealing. Another way is to engineer unique shape feature descriptors.

Feature engineering to globally register point sets can be divided into those that apply to the entire point set, or those that apply to specific interest points within the point set. The former category includes spin images [29] that directly align point clouds, NDT histograms [38] that are used to find relative rotation between two point clouds.

The latter category assigns a descriptor to certain interest points in a way that is invariant with respect to translation and rotation. These descriptors are designed to uniquely specify the geometry of a small neighbourhood around the interest point. Point correspondences are then found by matching interest points that have similar descriptors.

Interest point descriptors for point clouds include two general approaches: using a 2D depth map, or general point cloud. Depth map approaches include normal aligned radial feature (NARF) [55] and curvelet features [2]. These assume that the point cloud has been captured from a single viewpoint, and that range data is dense. As such, they may not perform well for a moving lidar which has sparse angular resolution. Feature-based global registration using general point clouds include fast point feature histograms (FPFH) [50] and integral volume descriptors (IVD) [20]. FPFH consists of computing statistics about a point's neighbours' relative positions and estimated surface normals. IVD

relies on finding the proportion of a small sphere around a point being occupied by the object's volume, by means of ray shooting in a voxel grid. However, these are relatively slow to compute, and also do not perform reliably for sparse point clouds.

Having established point correspondences using the feature descriptors, the transformation can be found in the same way as described in the previous section. The only difference is that point correspondences have been established using features instead of proximity. Three point correspondences are sufficient to recover the transformation. Typically, there are many outliers, so a random sample consensus (RANSAC) approach is used [50]. Localization accuracy of interest points is usually poor, so a local point set registration algorithm such as ICP is often needed as a postprocessing step to improve alignment [50] [20].

2.2 Visual Odometry

Visual odometry refers to methods that primarily rely on one or more cameras to recover the relative transformation between two frames. These can generally be classified as "dense" methods that reason about photometric error over all pixels the entire image, or "sparse" methods that track a relatively small set of interest points, or features. Compared to dense methods, sparse methods have a few drawbacks: tracking features may not be reliable as objects appear differently when viewed from different angles; the vast majority of the image data is not used; the environment can only be sparsely reconstructed. However, sparse systems have been studied for much longer, are often able to function better than dense methods when relative transformation between two frames is large, and can be more computationally efficient.

A main work for dense monocular visual odometry is large-scale direct simultaneous localization and mapping (LSD SLAM) [16]. When depth data is available, there are more methods such as ElasticFusion [62]. Very recently, deep machine learning techniques such as convolutional neural networks have also been applied to visual odometry by learning a direct mapping from an image to the six degree of freedom transformation [13].

In contrast to the relatively recent work on dense visual odometry, many methods for tracking visual features for sparse algorithms in computer vision have been proposed in past decades. In section 2.1.2 we have seen several methods for constructing point cloud features at interest points. Compared to these, visual features are more mature and re-

liable. To extract features, interest points are detected in the image. Then, to associate features across different frames, a unique descriptor of each feature is constructed, in a way that is mostly invariant with respect to translation, rotation, scale, and illumination, and also robust against noise. Algorithms to detect interest points include the Harris corner [23] and the Shi–Tomasi "Good Features To Track" [53].

Many techniques exist for constructing feature descriptors. These almost always have creative acronyms, including scale-invariant feature transform (SIFT) [36], speeded-up robust features (SURF) [4], binary robust invariant scalable keypoints (BRISK) [34], features from accelerated segment test (FAST) [47], binary robust independent elementary features (BRIEF) [9], oriented FAST and rotated BRIEF (ORB) [48], fast retina keypoint (FREAK) [3] and so on. They each have different trade-offs between computational efficiency and accuracy [39]. Some of these methods, such as SIFT and SURF, also include methods for detecting interest points. In general, modern binary features such as BRISK, ORB, and FREAK are orders of magnitude faster to compute than older, histogram-based features such as SIFT. Recall performance of these binary features is generally on par or better than SIFT-like features for the purpose of visual odometry as consecutive frames are very similar in appearance, although modern variants of SIFT-like features may be more robust under wildly different lighting, scale, and rotation.

Alternatively, if the motion is small, then interest points can be tracked using optical flow without having to compute their descriptors. A well-known method to do this is the Lucas–Kanade–Tomasi tracking [37][58][53]. A notable variant of LKT optical flow is the addition of an image pyramid, improving the efficiency and robustness for somewhat larger translations [7].

Having established point correspondences using the feature descriptors, different algorithms can be used to recover pose between two frames, depending on whether the depth data is known in one or both of the frames. Such methods are covered in depth in *Multiple View Geometry* [24].

2.2.1 2D-2D matching

When there are 2D features tracked across two different frames without depth data, the task of recovering the relative transformation is called 2D-2D matching. Algorithms that perform pose estimation using purely 2D-2D matching are known as monocular visual odometry, as a single camera has no depth data. In this case, rotation can be recovered

absolutely, but the translation is only determined up to scale. As such, a purely monocular approach is not sufficient to perform full pose estimation, unless there is prior knowledge about the contents of the scene.

In general, point correspondences in a scene viewed from two cameras are related by the fundamental matrix [24], from which it is possible to extract two possible rotation matrices and the normalized translation vector. When the camera intrinsic calibration is known, they are related by the essential matrix [24]. Five pairs of point correspondences are sufficient to recover the essential matrix. This is known as the five-point algorithm [44], which is a popular method for 2D-2D matching. The advantage of using as few points as possible is that RANSAC takes fewer samples on average to reach consensus.

2.2.2 3D-2D matching

Often, 3D positions of features in one frame is known. This can happen when 3D points are triangulated from past observations through bundle adjustment, or from stereo cameras, or using an external range sensor such as a lidar. When 2D projections of these known 3D points are observed in a second frame, the task of recovering the relative transformation is called 3D-2D matching, or perspective n-point (PnP) [24].

A popular algorithm for PnP is ePnP, which runs in linear time complexity with respect to the number of points [33].

When there are many outliers, RANSAC can be used. As with 2D-2D, using fewer points is advantageous as fewer RANSAC iterations are needed on average to reach consensus. However, when using too few points, the solutions may be ambiguous.

The minimal form of PnP is P3P. When using three points, there are at most four possible solutions. A plurality of P3P algorithms have been proposed [18]. In practice, a fourth correspondence is often used to resolve ambiguity. Numerous other algorithms have been proposed, using 4, 5, 6, or more points [26]. With six points, there are 12 linear equations, guaranteeing a unique solution as all 9 elements of the rotation matrix and 3 elements of the translation vector can be determined.

2.2.3 3D-3D matching

When the 3D positions of features in both frames are known (for example, if an RGB-D sensor is used), the problem becomes identical to global point set registration discussed in section 2.1.2. Typically, RANSAC can be used along with a closed form solution for the

transformation. The main advantage over a purely lidar method is that visual features tend to be much more reliable and fast to compute than point cloud features.

Note that in the case of stereo visual odometry, depth from stereo triangulation is generally considered unreliable, so 3D-3D matching is not usually used in that case. Instead, minimizing reprojection error in 3D-2D matching is typically used.

2.3 Sensor Fusion

When combining multiple sensors, there are two main concepts: recursive filtering and batch optimization [35]. The filtering approach recursively updates the state probabilistically using only the latest observations from the sensors, often using one of them for state propagation and the others for updates. Examples include the Kalman filter, its variants, and the particle filter. The batch optimization approach maintains a history of sensor observations and simultaneously performs a nonlinear optimization over the past states to give the most likely explanation of said observations. Batch optimization is more computationally intensive but produces more accurate results.

Strategies to combine multiple methods can further be classified as either loose coupling or tight coupling. The loosely coupled approach independently uses each method and then fuses them in a subsequent step: for example, it might use one method to preprocess another sensor's output, or to initialize a second method. The tightly coupled approach includes all measurements in a common problem where all states are jointly estimated. In general, a tightly coupled approach is more accurate than a loosely coupled one, but more computationally intensive. Then again, the accuracy improvement from tight coupling may not be significant if one method is significantly better than the other.

The method of RGB-D mapping by Henry *et al* [25] tightly couples the 3D-3D matching from sparse visual features with ICP for depth points with unknown correspondence. This method does not use camera data where depth is unavailable.

Depth-enhanced monocular odometry (DEMO) by Zhang *et al* [64] tightly couples the 2D-2D epipolar constraint as well as 3D-2D reprojection errors in a single cost function to estimate the frame-to-frame tranformation. It is able to outperform purely 3D-2D matching methods as well as purely 3D-3D methods by utilizing camera data outside of the field of view of the lidar. However, it cannot utilize lidar data outside of the field of view of the camera.

Visual-lidar odometry and mapping (V-LOAM) by Zhang *et al* [66] loosely couples DEMO [64] and LOAM [65] by using the former to mitigate motion distortion in lidar data, and subsequently using the latter which further removes motion distortion while performing point set registration. As a consequence of the loose coupling, the accuracy is ultimately primarily limited by the lidar and is less affected by uncertainty in the calibration between the lidar and camera.

2.4 Pose-Graph Optimization

The pose-graph is a representation of the robot's trajectory over time and the measurements observed between different points on this trajectory. Each vertex in the graph is a pose such as the pose of the robot at a different time, or a landmark. Edges represent constraints, such as a relative transformation between two frames found through the pose estimation frontend. Given these constraints, a large batch nonlinear optimization is used determine the robot's trajectory and poses of all relevant landmarks to optimally satisfy all of the constraints. Although the problem has many variables, it has a sparse structure, allowing efficient algorithms to perform this optimization.

Incremental Smoothing and Mapping [30] provides an efficient and exact solution which takes advantage of the sparsity of pose-graphs as well as the incremental nature of SLAM. It updates a QR factorization of the information matrix, and avoids recalculating parts of the system that do not change. The g^{2}o library [32] and Ceres Solver [1] are both software suites that implement optimization techniques to efficiently solve sparse systems.

Bundle adjustment is a special case of pose-graph, where the positions of landmarks observed using cameras are estimated along with robot poses. A landmark is usually a sparse visual feature which has been observed in several frames. Bundle adjustment has been used in photogrammetry for a long time [59]. In computer vision, it is often used for structure from motion.

2.4.1 Loop closure detection

When using a pose-graph representation, a challenging problem is to discover edges connecting a current vertex to a past vertex. Returning to a previously encountered location is known as loop closure. Naively, one may attempt to run pose estimation front end between the current vertex and every past vertex to see if they match. Unfortunately, this is

prohibitively expensive and increases quadratically with time. A simpler approach is to only attempt loop closure between vertices close together. However, this does not work if odometry has drifted significantly. In the past, many researchers have simply resorted to manually tagging loop closures.

Several methods have been proposed to capture the scene in a compact representation so that it is easy to check whether the robot has been to the same place before.

In lidar-based methods, a histogram of point cloud normals in the normal distributions method [56] may be used as a unique scene descriptor for accomplishing loop closure.

For visual SLAM methods, appearance-based methods are typically used. In ORB-SLAM [42], a bag-of-words approach is used to determine if a camera image is similar to a previously seen image.

2.5 Relation to My Work

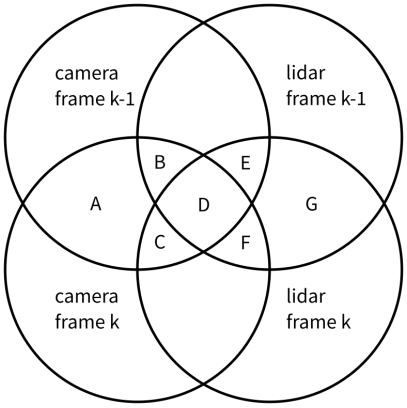

Figure 2: Venn diagram of available data for frame-to-frame pose estimation between frames k and $k-1$, and possible strategies. Refer to table 1.

My work, VELO, combines 2D-2D matching, 3D-2D matching, 3D-3D matching, and local point set registration in a deeply coupled fashion. It can be considered the deep coupling of [25] and [64]. Compared to [64], my method is able to utilize purely lidar data

Region (figure 2)	VELO	DEMO [64]	LOAM [65]	V-LOAM [66]	RGB-D m. [25]
A	2D-2D	2D-2D	-	2D-2D	-
B	3D-2D	3D-2D	-	3D-2D	3D-2D
C	2D-3D	-	-	-	2D-3D
D	3D-3D, ICP p2P	3D-2D	LOAM	LOAM	3D-3D, ICP
E, F	ICP p2P	-	LOAM	LOAM	ICP
G	ICP p2P	-	LOAM	LOAM	-

Table 1: Strategies for frame-to-frame motion estimation used by my method, VELO, compared to DEMO [64], LOAM [65], V-LOAM [66], and RGB-D mapping [25]. Refer to regions from figure 2. ICP p2P refers to point-to-plane iterative closest point. The ICP variant used by LOAM [65] and V-LOAM [66], written as LOAM in the table, first identifies reliable points in the point cloud and then uses either point-to-line or point-to-plane, while accounting for lidar motion distortion by assuming a linear interpolation. Here, 3D-2D matching refers to matching 3D points from frame k to 2D points in frame $k-1$ and 2D-3D refers to 3D points from $k-1$ and 2D points from k. In [25], the sensor does not produce data in region G.

in the case that few or no visual features are available. Compared to [25], my method is able to utilize mainly the camera in the case that lidar data is unavailable or degenerate.

Whereas [64], [65], and [66] do not perform loop closure, my method does, similar to [25].

3 Overview of Method

3.1 Extraction of Visual Features

My method employs sparse visual odometry, which depends on point correspondences between images. Point correspondences are found in two steps: first, features are detected, and second, correspondences are found between them. The quality of visual features has a large impact on the quality of the pose estimation pipeline. The proposed method uses Good Features To Track [53] to detect features.

The GFTT feature detection routine is briefly described here. For a point $\boldsymbol{p} = [u, v]$ in the image I, define the 2×2 matrix $\mathbf{Z}$:

$$\mathbf{Z} = \begin{bmatrix} I_u^2 & I_u I_v \\ I_u I_v & I_v^2 \end{bmatrix} \tag{1}$$

where the numeric gradient of the image is $\nabla I(\boldsymbol{p}) = [I_u, I_v]^T$. Let λ_1, λ_2 be the eigenvalues of $\mathbf{Z}$. Two small eigenvalues indicate that the intensity profile in the region is constant. A large and a small suggest the texture is unidirectional, permitting good tracking in one direction but not the other. If both eigenvalues are large, the pattern at $\boldsymbol{p}$ is above the noise level in both directions, allowing the point to be tracked accurately. Hence, a good feature to track must have both eigenvalues to be above a threshold λ.

It is preferable for the features to be distributed approximately uniformly within the image in order to achieve consistent visual odometry performance. Intuitively, little information is gained when using multiple features close together, compared to using only a single feature. To spread the features evenly within the image, a minimum distance d between features is enforced. Features are detected greedily: For m iterations, the pixel $\boldsymbol{p}$ with the maximum score $\min(\lambda_1, \lambda_2) > \lambda$ is selected such that $\|\boldsymbol{p} - \boldsymbol{q}\| < d$ for each previously selected feature $\boldsymbol{q}$. A grid of size d is used to efficiently check if a candidate feature is within a previously selected one, as only up to two previously selected features can exist in any given grid cell. The parameter λ is chosen to be some fraction (typically 0.01) of the score of the best feature.

3.2 Feature Tracking and Descriptor Matching

In VELO, two strategies are used to assign correspondences between keypoints: optical flow and binary descriptor matching. For consecutive frames t and $t-1$, the relative mo-

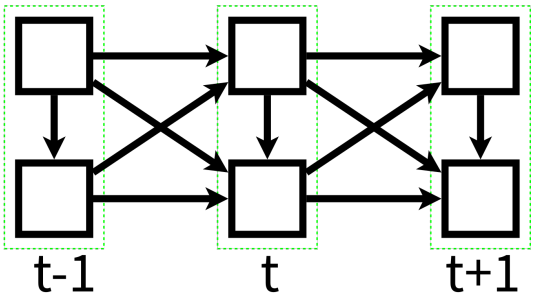

Figure 3: Feature tracking in frames $t-1$, t, $t+1$. Squares are camera images, arrows are tracked features using LKT sparse optical flow. If a feature is tracked from multiple images, the geometric median of the feature's position is used.

tion between the images is small. In this case, a pyramidal implementation of the Lucas–Kanade optical flow tracking is used [7]. Optical flow tracking assumes that the motion of a point seen by a camera is linearly related to the change in intensity and the local gradient at that point. As such, it is effective when the motion is sufficiently small that the appearance at the point is locally smooth. The pyramid approach uses successively downscaled images, so that, for larger motions, the linear assumption holds at coarser levels of the pyramid. Then, the tracked point's location is successively refined using the finer pyramid levels.

However, optical flow tracking tends to result in outliers in regions with complex textures and near the edge of the frame where a feature moving out of the frame cannot be found. To combat outliers, we require a tracked feature to be associated not only between different frames, but also between different cameras (i.e. stereo correspondence). At a frame t, for a camera c, a feature is tracked using optical flow from all camera images at frame $t-1$. Moreover, if $c > 0$, the feature is also tracked from camera 0 at frame t. Where there are n_c cameras, this can result in up to $n_c + 1$ candidate positions for each feature. The feature's final position is set to be the geometric median of these $n_c + 1$ hypotheses. The geometric median is robust against outliers by minimizing the sum of distances to the $n_c + 1$ points, and is efficiently calculated using Weiszfeld's algorithm [61].

By tracking features between cameras using optical flow instead of more conventional sliding-window approaches to stereo correspondence, it is agnostic with respect to incorrect calibration (as pointed out in [14]), and also works even when the cameras

are rotated or have different distortion parameters.

Another weakness of optical flow is that a tracked point slowly drifts over time. After a few frames, it loses coincidence with the initial real-world point being tracked. To remove this effect, the Fast Retina Keypoint (FREAK) feature descriptor [3] is computed for each feature point. The original descriptor of a tracked feature is stored, and if the descriptor computed at a newly tracked position is sufficiently different, the feature is considered to be expired and removed. Compared to other feature descriptors, FREAK is very fast, and relatively robust and accurate when used in visual odometry.

Computing the FREAK descriptors also allows points to be associated when the relative pose between frames is large. This occurs during loop closures.

3.3 Camera Projection

For this thesis, a typical pinhole camera model is used [24]. Generally, a 3×4 projection matrix $\mathbf{C}$ relates a 3D point in homogeneous coordinates $\boldsymbol{P} = [x, y, z, 1]^T$ to its 2D projection matrix $\boldsymbol{p} = [u, v, 1]^T$,

$$\boldsymbol{p} \equiv \mathbf{C}\boldsymbol{P} \tag{2}$$

In the following sections we shall consider the camera to be calibrated such that its projection matrix which projects a 3D point to canonical camera coordinates is $[\mathbf{I}|\mathbf{0}]$. If the physical projection matrix is $[\mathbf{K}|\mathbf{0}]$ where $\mathbf{K}$ is the 3×3 intrinsict matrix, then a point $\boldsymbol{p}_p$ in pixel coordinates is related to the point $\boldsymbol{p}_c$ in canonical coordinates by $\boldsymbol{p}_c = \mathbf{K}^{-1}\boldsymbol{p}_p$.

When there are multiple cameras, all cameras except camera 0 are translated with respect to the coordinate frame. Consequently the assumption that the projection matrix is $[\mathbf{I}|\mathbf{0}]$ no longer holds. Suppose our offset calibrated camera has a 3×4 physical projection matrix of the form:

$$\mathbf{C} = \Big[\mathbf{K}|\boldsymbol{t}_k\Big] \tag{3}$$

where $\boldsymbol{t}_k$ is a 3×1 vector. We can isolate the extrinsic translation $\boldsymbol{t}_e$ by noting

$$\boldsymbol{t}_e = \mathbf{K}^{-1}\boldsymbol{t}_k. \tag{4}$$

Then, we may write the projection matrix as

$$\mathbf{C} = \mathbf{K}[\mathbf{I}|\boldsymbol{t}_e]. \tag{5}$$

Note that any rotation in the extrinsic calibration may be absorbed into the intrinsic matrix $\mathbf{K}$. Suppose the transformation between two frames $\{C_0^m\}$ and $\{C_0^s\}$ is $\mathbf{T}_m^s$ consisting of rotation $\mathbf{R}$ and translation $\boldsymbol{t}$:

$$\mathbf{T}_m^s = \begin{bmatrix} \mathbf{R} & \boldsymbol{t} \\ \mathbf{0}_{1\times 3} & 1 \end{bmatrix} \tag{6}$$

where $\mathbf{R}$ is a 3×3 rotation matrix and $\boldsymbol{t}$ is a 3×1 translation vector.

Then the transformation $\mathbf{T}_m^{s\,\prime}$ of the offset camera $c \neq 0$ between frames $\{C_c^m\}$ and $\{C_c^s\}$ is:

$$\mathbf{T}_m^{s\,\prime} = \begin{bmatrix} \mathbf{I} & \boldsymbol{t}_e \\ \mathbf{0} & 1 \end{bmatrix} \mathbf{T}_m^s \begin{bmatrix} \mathbf{I} & \boldsymbol{t}_e \\ \mathbf{0} & 1 \end{bmatrix}^{-1} = \begin{bmatrix} \mathbf{R} & \boldsymbol{t} - \mathbf{R}\boldsymbol{t}_e + \boldsymbol{t}_e \\ \mathbf{0} & 1 \end{bmatrix}. \tag{7}$$

This fact is used when implementing frame-to-frame motion estimation for multiple cameras in section 4, as the visual odometry algorithms operate within the canonical coordinate frame of each camera independently.

3.4 Feature Depth Association

For each visual feature, its depth is found using lidar points if possible. Two key challenges exist: first, the lidar points are sparse and may not cover the whole frame; second, parallax between the camera and the lidar can cause some lidar points to be occluded from the camera's perspective.

To overcome sparsity, it is necessary to interpolate between the nearest lidar points projected into the camera image. In depth-enhanced monocular odometry (DEMO) [64], a 2D KD-tree is constructed to query for the three closest lidar points, forming a planar interpolation. A drawback is that the triangle formed from these three points may be degenerate (if the points are nearly collinear) and does not always contain the feature point.

Another common way to interpolate between sparse point data is to construct a Delaunay triangulation. The triangulation can be efficiently computed in either in the planar camera frame, or in spherical geometry [10]. Then, a fast point location algorithm such as [40] queries the triangle containing the feature point. This has the advantage that the triangle always contains the query point, and is therefore more accurate. Both constructing a KD-tree and a Delaunay triangulation are possible in $O(n \log n)$ time complexity, and querying each point is possible in $O(\log n)$ time complexity.

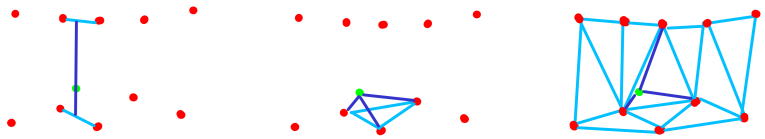

Figure 4: Depth association using three methods. From left to right: my method; finding the three nearest points using a KD-tree; point location within a Delaunay triangulation. Red points are projected lidar points. The green point is the 2D feature.

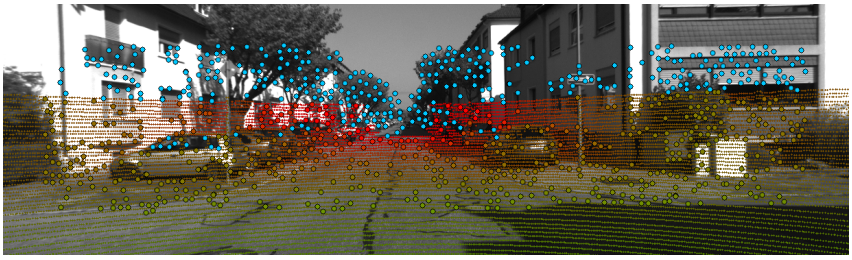

Figure 5: Example of depth association using lidar points. Small points are lidar points projected into camera image, colour coded from near (green) to far (red). Circles with black outlines are tracked visual features, likewise colour-coded if they are associated with depth, and blue otherwise.

My approach to associating visual features with depth accounts for both sparsity and and most of occlusion. It is also both faster and simpler to implement compared to the aforementioned methods. This is made possible by making two observations about the geometric properties of a typical lidar sensor. First, an interpolation on a line segment (as opposed to a planar patch) is sufficient since a typical lidar sensor has very dense angular resolution in one axis despite sparse resolution in the other. For example, the Velodyne HDL-64E has very dense azimuth resolution, so interpolation in the altitude direction is much more important. In fact, for the KITTI data set, the median pixel distance between points in the azimuth direction is less than 3 pixels.

Second, a KD-tree or such data structure is unnecessary because the lidar points arrive in nearly sorted order due to the scanning motion of the lidar sensor. The only out-of-order points arise due to occlusion caused by parallax and can be removed in a single pass. A lidar spinning in one axis with k lasers will produce k sorted "rings", or layers, of points. Each ring projects to a "row" of points in the image frame. Equivalently, a li-

dar with a single laser spinning in two axes will also produce k "rings" and "rows", if the rotation rate in one axis is k times the other.

When iterating over each ring, a lidar point is projected if it is in front of the camera and within the camera's field of view. Projected points are stored in a stack. Suppose that the points project to mostly left-to-right in the camera image. While processing a new point, points in the stack to the right of and having greater depth than the new point are assumed to be occluded and popped from the stack. Conversely, if the new point is to the left of and having greater depth than the last point in the stack, it is assumed to be occluded and ignored. Otherwise, it is pushed to the stack. Thus, projecting points into the camera image and removing occluded points runs in $O(n)$ time complexity. This method only handles occlusion within each ring, but does not account for the possibility of two different rings occluding each other. However, the sparse nature of the lidar means that these rings are far apart and are unlikely to occlude each other.

For any row of lidar points, a binary search suffices to find the closest lidar point in the x-direction to any query point. Another binary search over the k rows suffices to find the two closest rows in the positive and negative y direction. Then a segment is formed from the two points from these two rows. The query point is then associated with the interpolated position on this segment. As the query algorithm relies only on binary search, the time complexity is in $O(\log n)$.

The running time of the entire pose estimation pipeline is not dominated by depth association, but rather by the later steps discussed in the next section. However, the running time of depth association becomes more important if there are many cameras, if the camera operates at a much higher framerate than lidar, or if expensive steps like point set registration are only run at keyframes. For more discussion, refer to section 6.1.

4 Front End: Frame-to-Frame Motion Estimation

> The fulfilment of what exists potentially,
> in so far as it exists potentially, is motion.
>
> Aristotle
> *The Basic Works of Aristotle* (1941)

Let us consider two frames $\{C^m\}$ and $\{C^s\}$, where C^m is located at the origin and C^s is located with pose expressed by the homogeneous rigid transformation as in equation 6. The goal of frame-to-frame motion estimation is to recover rigid transformation $\mathbf{T}_m^s$, consisting of rotation $\mathbf{R}$ and translation $\boldsymbol{t}$.

The parametrization of rotation in $SO(3)$ is the angle-axis representation. Given a 3×1 rotation vector of the form $\theta = [\theta_x, \theta_y, \theta_z]^T$, the rotation matrix is obtained using the Rodrigues formula [45],

$$\mathbf{R}(\theta) = e^{[\theta]_\times} = \mathbf{I} + \frac{[\theta]_\times}{\|\theta\|} \sin \|\theta\| + \frac{[\theta]_\times^2}{\|\theta\|^2}(1 - \cos \|\theta\|), \tag{8}$$

where $[\theta]_\times$ is the 3×3 skew symmetric matrix of θ. Geometrically, the rotation vector represents a rotation by an angle equal to $\|\theta\|$ about the axis θ. We note that $(\mathbf{R}(\theta))^{-1} = \mathbf{R}(-\theta)$.

The optimization problem therefore has 6 parameters, which can be written as $\boldsymbol{x} = [\theta, \boldsymbol{t}]$.

4.1 3D-3D Matching

Suppose that the camera-lidar system at $\{C^m\}$ observes the 3D point $\boldsymbol{M} = [x_m, y_m, z_m]^T$ associated with the point $\boldsymbol{S} = [x_s, y_s, z_s]^T$ observed from $\{C^s\}$.

The residual arising from each 3D-3D correspondence is simply the distance between the two 3D points,

$$\boldsymbol{f}_{\text{3D-3D}}(\boldsymbol{M}, \boldsymbol{S}; \boldsymbol{x}) = \boldsymbol{S} - \mathbf{R}\boldsymbol{M} - \boldsymbol{t}. \tag{9}$$

The function $\boldsymbol{f}_{\text{3D-3D}}$ is vector-valued with size 3.

4.2 3D-2D Matching

Suppose that the camera-lidar system at $\{C^m\}$ observes the 3D point $\boldsymbol{M} = [x_m, y_m, z_m]^T$ associated with the 2D point $\boldsymbol{s} = [u_s, v_s]^T$ observed from $\{C^s\}$.

The residual arising from each 3D-2D correspondence is simply the distance between the 2D point and the 2D projection of the 3D point,

$$\boldsymbol{f}_{\text{3D-2D}}(\boldsymbol{M}, \boldsymbol{s}; \boldsymbol{x}) = \boldsymbol{s} - \begin{bmatrix} x'_m / z'_m \\ y'_m / z'_m \end{bmatrix} \tag{10}$$

where

$$\boldsymbol{M}' = \begin{bmatrix} x'_m \\ y'_m \\ z'_m \end{bmatrix} = \mathbf{R}\boldsymbol{M} + \boldsymbol{t}. \tag{11}$$

The function $\boldsymbol{f}_{\text{3D-2D}}$ is vector-valued with size 2.

In the converse case, the camera at $\{C^m\}$ observes the 2D point $\boldsymbol{m} = [u_m, v_m]^T$ associated with the 3D point $\boldsymbol{S} = [x_s, y_s, z_s]^T$ observed from $\{C^s\}$.

The residual arising from each 2D-3D correspondence is

$$\boldsymbol{f}_{\text{2D-3D}}(\boldsymbol{m}, \boldsymbol{S}; \boldsymbol{x}) = \boldsymbol{m} - \begin{bmatrix} x'_s / z'_s \\ y'_s / z'_s \end{bmatrix} \tag{12}$$

where

$$\boldsymbol{S}' = \begin{bmatrix} x'_s \\ y'_s \\ z'_s \end{bmatrix} = \mathbf{R}^{-1}(\boldsymbol{M} - \boldsymbol{t}). \tag{13}$$

The function $\boldsymbol{f}_{\text{2D-3D}}$ is vector-valued with size 2.

4.3 2D-2D Matching

Suppose that the camera at $\{C^m\}$ observes the 2D point $\boldsymbol{m} = [u_m, v_m, 1]^T$ associated with the 2D point $\boldsymbol{s} = [u_s, v_s, 1]^T$ observed from $\{C^s\}$.

The residual arising from each 2D-2D correspondence is due to the epipolar constraint between the two cameras,

$$\boldsymbol{f}_{\text{2D-2D}}(\boldsymbol{m}, \boldsymbol{s}; \boldsymbol{x}) = \boldsymbol{s}^T \mathbf{E} \boldsymbol{m} \tag{14}$$

where $\mathbf{E}$ is the essential matrix, defined as:

$$\mathbf{E} = [\boldsymbol{t}]_\times \mathbf{R}. \tag{15}$$

As $\boldsymbol{f}_{\text{2D-2D}}$ is a dot product, it is scalar-valued (but it is here typeset in boldface for consistency with other residual functions). When computing $[\boldsymbol{t}]_\times$, the translation is normalized first. This prevents the algorithm from incorrectly biasing the translation towards zero when 2D-2D matches make up the majority of available data.

4.4 Point Set Registration

Suppose two lidar point clouds corresponding to the desired frames are $\mathcal{M}$ and $\mathcal{S}$. For a point $\boldsymbol{M} \in \mathcal{M}$, we would like to find three points in $\boldsymbol{S}_0, \boldsymbol{S}_1, \boldsymbol{S}_2 \in \mathcal{S}$ that are close to $\mathbf{R}'\boldsymbol{M} + \boldsymbol{t}'$, forming a plane. The transformation $\mathbf{R}', \boldsymbol{t}'$ is the estimate of the transformation from the previous iteration. Then the point-to-plane distance is the residual.

As we have discussed in section 3.4, lidar point cloud data are organized in "rings". If we simply query the three closest points in $\mathcal{S}$ which are the closest to $\mathbf{T}\boldsymbol{M}$, then these points are likely to belong to the same ring. Consequently, they are likely to be near-collinear, resulting in an ill-conditioned situation. To overcome this, two of the points are constrained to belong in different rings.

To speed up querying the closest point in a ring, a KD-tree is constructed for each ring. The three points are checked to be within a threshold distance of the query point. Let $\boldsymbol{N} = (\boldsymbol{S}_1 - \boldsymbol{S}_0) \times (\boldsymbol{S}_2 - \boldsymbol{S}_0)$ be the normal vector. The residual is:

$$\boldsymbol{f}_{\text{ICP}}(\boldsymbol{M}, \boldsymbol{S}_0, \boldsymbol{S}_1, \boldsymbol{S}_2; \boldsymbol{x}) = \frac{1}{\|\boldsymbol{N}\|}\boldsymbol{N}^T(\mathbf{R}\boldsymbol{M} + \boldsymbol{t} - \boldsymbol{S}_0) \tag{16}$$

Similarly to the 2D-2D residual, $\boldsymbol{f}_{\text{ICP}}$ is a dot product and is scalar-valued. Near-degenerate planes where $\boldsymbol{N}$ is close to zero are ignored.

In the interest of reducing computational cost, it is not necessary to compute the residual for every point $\boldsymbol{M} \in \mathcal{M}$, as the query step is expensive. As such, only one in every 100 points in $\mathcal{M}$ is used. Compared to methods such as LOAM [65], which uses fewer than 10 points per ring (albeit with heuristics to select and classify said points), one in every 100 is still quite a large number of points.

4.5 Optimization Strategy

Having established the cost functions, the goal is to solve for the parameters that minimize the sum of squares of residuals:

$$\boldsymbol{x}^* = \arg\min_{\boldsymbol{x}} \frac{1}{2}\|\boldsymbol{F}\|^2 \tag{17}$$

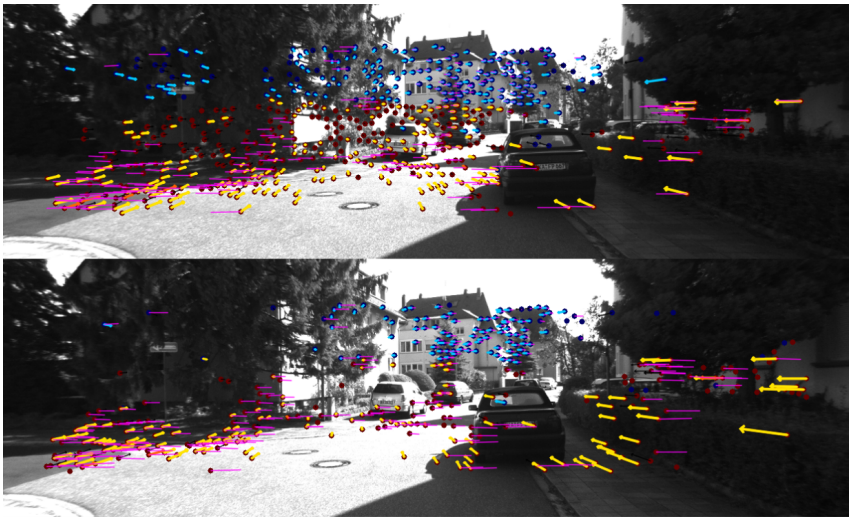

Figure 6: Estimated motion of tracked keypoints. Top: left camera image, Bottom: right camera image. Yellow arrows: 3D-2D matches; Red arrows: 3D-3D matches; Teal arrows: 2D-2D matches; Black arrows: outlier matches. Dark red points: points with depth; Dark blue points: points without depth. Magenta segments: stereo correspondences.

where the cost function is the concatenation of all residuals.

$$\boldsymbol{F} = \begin{bmatrix} [\boldsymbol{\rho}_{\text{3D-3D}}(\boldsymbol{f}_{\text{3D-3D}}(\boldsymbol{M},\boldsymbol{S};\boldsymbol{x});\rho_{\text{3D-3D}})\forall \boldsymbol{M},\boldsymbol{S} \in \text{3D-3D matches}] \\ [\lambda_{\text{3D-2D}}\boldsymbol{\rho}_{\text{3D-2D}}(\boldsymbol{f}_{\text{3D-2D}}(\boldsymbol{M},\boldsymbol{s};\boldsymbol{x});\rho_{\text{3D-2D}})\forall \boldsymbol{M},\boldsymbol{s} \in \text{3D-2D matches}] \\ [\lambda_{\text{3D-2D}}\boldsymbol{\rho}_{\text{3D-2D}}(\boldsymbol{f}_{\text{2D-3D}}(\boldsymbol{m},\boldsymbol{S};\boldsymbol{x});\rho_{\text{3D-2D}})\forall \boldsymbol{m},\boldsymbol{S} \in \text{2D-3D matches}] \\ [\lambda_{\text{2D-2D}}\boldsymbol{\rho}_{\text{2D-2D}}(\boldsymbol{f}_{\text{2D-2D}}(\boldsymbol{m},\boldsymbol{s};\boldsymbol{x});\rho_{\text{2D-2D}})\forall \boldsymbol{m},\boldsymbol{s} \in \text{2D-2D matches}] \\ [\lambda_{\text{ICP}}\boldsymbol{\rho}_{\text{ICP}}(\boldsymbol{f}_{\text{ICP}}(\boldsymbol{M},\boldsymbol{S}_0,\boldsymbol{S}_1,\boldsymbol{S}_2;\boldsymbol{x});\rho_{\text{ICP}})\forall \boldsymbol{M},\boldsymbol{S}_0,\boldsymbol{S}_1,\boldsymbol{S}_2 \in \text{ICP matches}] \end{bmatrix}. \quad (18)$$

The scaling constants $\lambda_{\text{3D-2D}}$, $\lambda_{\text{2D-2D}}$, and λ_{ICP} are needed to compensate for the different units of the different residuals, and are selected by hand. The functions $\boldsymbol{\rho}_{\text{3D-3D}}$, $\boldsymbol{\rho}_{\text{3D-2D}}$, $\boldsymbol{\rho}_{\text{2D-2D}}$, and $\boldsymbol{\rho}_{\text{ICP}}$ are robust loss functions to minimize the impact of outliers, and are controlled by parameters ρ respectively.

To ensure robust performance even in the presence of outliers, the following choice

of loss functions is used:

$$\boldsymbol{\rho}_{\text{3D-3D}}(\boldsymbol{x}; \rho_{\text{3D-3D}}) = \rho_{\text{3D-3D}} \arctan\left(\frac{\boldsymbol{x}}{\rho_{\text{3D-3D}}}\right) \tag{19}$$

$$\boldsymbol{\rho}_{\text{3D-2D}}(\boldsymbol{x}; \rho_{\text{3D-2D}}) = \rho_{\text{3D-2D}} \arctan\left(\frac{\boldsymbol{x}}{\rho_{\text{3D-2D}}}\right) \tag{20}$$

$$\boldsymbol{\rho}_{\text{2D-2D}}(\boldsymbol{x}; \rho_{\text{2D-2D}}) = \rho_{\text{2D-2D}} \arctan\left(\frac{\boldsymbol{x}}{\rho_{\text{2D-2D}}}\right) \tag{21}$$

$$\boldsymbol{\rho}_{\text{ICP}}(\boldsymbol{x}; \rho_{\text{ICP}}) = \rho_{\text{ICP}} \log\left(1 + \frac{|\boldsymbol{x}|}{\rho_{\text{ICP}}}\right) \tag{22}$$

When the residuals are much smaller than ρ, the arctangent loss function is linear, and for values much larger than ρ, the value asymptotically approaches the constant $\rho\pi/2$. As such, outliers with a large residual will have little or no effect on the optimization, whereas inliers with a small residual would behave as though there were no loss function. The consequence of using the arctangent loss function is that the optimization problem may have many local minima. However, with a good initial guess, it is unlikely for the algorithm to converge to an incorrect local minimum. As the motion of a moving vehicle is generally smooth, a good initial guess can be obtained with a first order extrapolation of motion. Apart from relying on a good initial guess, one method to widen the basin of convergence to the correct minimum is to adjust the parameters ρ during the optimization process, in a way similar to annealing.

The Cauchy loss function used for $\boldsymbol{\rho}_{\text{ICP}}$ is less robust against outliers than the arctangent loss function. However, ICP correspondences are only established for nearby points anyway, so it is nearly impossible for the $\boldsymbol{f}_{\text{ICP}}$ to be large. In contrast, matching using feature descriptors can result in outliers arbitrarily far apart, necessitating the arctangent loss function.

Each ρ is set to approximately the median value of $\boldsymbol{f}$ as determined through experimentation. The λ values are chosen to ensure that the residuals arising from each type of matching contributes equally to the optimization problem.

In general, when performing nonlinear optimization, it is necessary to compute the derivative of the objective function $\boldsymbol{F}$ in the form of the Jacobian matrix. Fortunately, the Ceres Solver library [1] provides automatic differentiation, which computes the exact derivative within machine accuracy. Automatic diferentiation has asymptotically optimal performance and tends to be more efficient in practice than manually or symbolically finding the derivative, especially with respect to many inputs, although manually optimizing the derivative can be faster in performance-critical situations. The alternative to

computing the exact Jacobian is to approximate it using numerical methods, which can be slow and numerically unstable.

Having found the Jacobian matrix $\mathbf{J}$, the solution of equation 17 is found iteratively using the Levenberg–Marquardt method. Nonlinear optimization is usually based on solving a sequence of linear approximations of the original problem:

$$\boldsymbol{F}(\boldsymbol{x} + \Delta\boldsymbol{x}) \approx \boldsymbol{F}(\boldsymbol{x}) + \mathbf{J}(\boldsymbol{x})\Delta\boldsymbol{x} \tag{23}$$

which allows equation 17 to be expressed as:

$$\min_{\Delta\boldsymbol{x}} \frac{1}{2}\|\mathbf{J}(\boldsymbol{x})\Delta x + \boldsymbol{F}(\boldsymbol{x})\|^2. \tag{24}$$

The Levenberg–Marquardt algorithm belongs to a class of optimization algorithms known as trust region solvers, which, given some trust region radius μ, attempts to solve

$$\arg\min_{\Delta\boldsymbol{x}} \frac{1}{2}\|\mathbf{J}(\boldsymbol{x})\Delta x + \boldsymbol{F}(\boldsymbol{x})\|^2 \tag{25}$$

$$\text{such that } \|\mathbf{D}(x)\Delta\boldsymbol{x}\|^2 \leq \mu, \boldsymbol{L} \leq \boldsymbol{x} + \Delta\boldsymbol{x} \leq \boldsymbol{U} \tag{26}$$

The solution to this can be obtained by solving an unconstrained optimization of the form

$$\arg\min_{\Delta\boldsymbol{x}} \frac{1}{2}\|\mathbf{J}(\boldsymbol{x})\Delta x + \boldsymbol{F}(\boldsymbol{x})\|^2 + \frac{1}{\mu}\|\mathbf{D}(\boldsymbol{x})\Delta\boldsymbol{x}\|^2 \tag{27}$$

where $\mathbf{D}$ is typically chosen to be the square root of the diagonal of $\mathbf{J}^T\mathbf{J}$. In this case, this is solved using factorization, implemented in the DENSE_SCHUR method in Ceres Solver [1].

4.6 Triangulation of Features

As a visual feature is tracked over several frames and is observed by multiple cameras, we can estimate its 3D position through triangulation. Given a set of 3D observations and 2D observations from known camera poses, the position of the point may be optimized using the same set of cost functions as discussed in sections 4.1 and 4.2, where the poses of the cameras are held constant.

5 Back End: Loop Closure and Pose-Graph Optimization

> We are at a moment, I believe, when our experience of the world is less that of a long life developing through time than that of a network that connects points and intersects with its own skein.
>
> Michel Foucault
> "Of Other Spaces" (1967)

Although the focus of this thesis is the novel motion estimation front end, a robust back end is necessary in any pose estimation system to deliver a globally consistent map. Here, I do not apply bundle adjustment. The only edges in the pose graph are estimated between frames using the front end.

For now, a loop closure is simply deemed to be two vertices that are closer together in space than a certain threshold, yet far apart in time. No appearance data is used to determine loop closures. If such a loop closure is detected, the front end frame-to-frame relative pose estimation is run between the two frames. Should the result fail to converge or result in an estimate significantly different from the initial guess, this is removed from the pose graph.

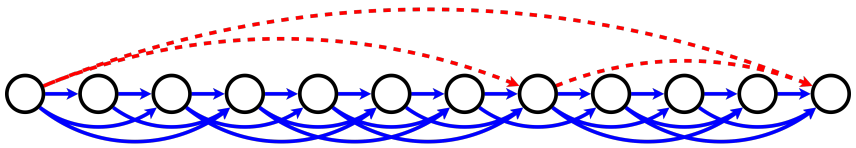

Figure 7: Pose-graph example. Circles indicate robot poses at different time steps. Blue arcs indicate consecutive frame-to-frame transformations using tracked features. Red dashed arcs indicate loop closure frame-to-frame transformations using matched features.

An advantage of using ICP in VELO is that, during a loop closure, the robot does not have to be facing in the same direction for frame-to-frame pose estimation to function. Whereas the field of view of a camera is narrow and requires the robot to be facing the same way for visual features to be associated, the lidar point cloud has 360 $^\circ$ field of view, allowing frame-to-frame relative transformation to be estimated regardless of orientation. That notwithstanding, visual features are still attempted to be matched between potential loop closures using the FREAK descriptors, in order to provide even more infor-

mation during loop closures.

In addition, relative transformations are estimated between each frame t and $t-1$, $t-2$, and so on, until insufficient tracked visual features remain. This reduces drift over time. As visual odometry is very fast, and a band matrix is a form of sparse system that iSAM is very efficient at solving, doing so only results in using a constant factor more time.

6 Implementation

The pose estimation system is implemented in C++11 and compiled with the GNU Compiler Collection (GCC) version 6.1.1. The following libraries are used:

- Ceres Solver [1] is used to perform nonlinear optimization for the frontend.
- iSAM [30][31][46] version 1.7 is used to perform pose-graph optimization.
- OpenCV [8] version 3.1.0 is used to process images, including detection and extraction of visual features.
- Point Cloud Library (PCL) [51] version 1.8.0 is used to process 3D point clouds.
- Eigen [22] version 3.2.8-4 is used to provide linear algebra.

To obtain the best performance, Ceres Solver and OpenCV are compiled from source using the latest version in the Git repository as of 2016-07-06. OpenCV and PCL are compiled against with NVidia CUDA version 7.5 to provide graphics processing unit (GPU) acceleration, and OpenBLAS-Lapack 0.2.18 to provide optimized linear algebra.

The source code which implements the algorithm described in this thesis can be found on Bitbucket at:

- https://bitbucket.org/dllu/vision-enhanced-lidar-odometry

Computing hardware consists of:

- Intel Core i7 6700K, four cores at 4.0 GHz
- 32 GiB DDR4 memory at 2800 MHz
- Two NVidia GeForce GTX 980 with 4 GiB of memory each
- Samsung 850 Evo 500 GB solid state drive

To process multiple data sets in parallel, GNU Parallel version 20160622 [57] is used.

6.1 Time Complexity

In this thesis, I have prioritized accuracy over running time. However, in order for a pose estimation system to be useful, it has to be able to function in real time. It must moreover be asymptotically fast, so as to scale well when future robots are expected to have faster response times, and sensors improve in resolution.

Therefore, a theoretical analysis of time complexity is discussed in this section. Where there are n lidar points, m visual features, and p pixels, the time complexity of each step in the pose estimation pipeline is summarized in table 1. Typical values of n and p are 10^5 and a typical value of m is 10^3.

Step	Time complexity
Detecting m visual features in image of p pixels	$O(p \log p)$
Tracking m visual features using optical flow	$O(m)$
Constructing m feature descriptors	$O(m)$
Feature depth association	$O(m \log n + n)$
Feature correspondence matching (single threaded)	$O(m^2)$
Feature correspondence matching (g threads, $g \leq m$)	$O(m^2/g)$
Constructing KD-tree of lidar points	$O(n \log n)$
Lidar point association in point set registration	$O(n \log n)$
One iteration of frame-to-frame motion estimation	$O(n + m)$
iSAM batch optimization, t frames	$O(t^3)$ worst case, $O(t^{1.5})$ typical

Table 2: Summary of time complexity for the steps in pose estimation pipeline

The time complexity of detecting GFTT keypoints is dominated by sorting the list of eigenvalue scores, which is $O(p \log p)$. Note that the OpenCV implementation of GFTT is highly efficient and uses OpenCL to harness the massively parallel capabilities of the GPU to accelerate finding the eigenvalues. This gives a constant time speedup over the single threaded $O(p)$ time it would take to do so. As each of the m features is added, enforcing the minimum distance constraint is constant time due to the grid strategy discussed in section 3.1. Hence the features are added in $O(m)$. The total time complexity of GFTT keypoint detection is $O(p \log p + m) = O(p \log p)$ as m is necessarily much smaller than p.

Tracking m features using sparse LKT optical flow uses linear time $O(m)$.

The time taken to construct the FREAK descriptors is linear in m because they are

computed independently, each taking a constant amount of time.

The time complexity of feature depth association has been discussed at length in section 3.4. In practice, the binary search over the k rings can be replaced with a linear search as $k \leq 64$ is quite small.

The time complexity of feature correspondence matching between two frames, each containing m features, is $O(m^2)$ because the distance of every feature in one frame to every feature in the other frame must be computed. In the case of FREAK descriptors, the distance metric is the Hamming distance of 64 bytes. Modern processors that support SSE4.2 instruction set can use the 64 bit POPCNT instruction to efficiently compute Hamming distance, reducing the running time by a constant factor. Even then, an $O(m^2)$ time complexity is the slowest step in the pipeline. However, OpenCV provides a CUDA implementation of feature matching which leverages the 10^3 cores of a GPU to massively speed up this embarrassingly parallel problem. Since the number of cores is similar to the number of features, the running time for feature matching becomes linear in m. Note that feature *matching* is only used for running frame-to-frame pose estimation in a loop closure setting. For consecutive frames, LKT optical flow tracking is relied upon, as desribed in section 3.2.

To perform point set registration using ICP, a KD-tree must be constructed. The Point Cloud Library provides an efficient approximate implementation which runs in $O(n \log n)$. It uses the Fast Library for Approximate Nearest Neighbors (FLANN) [41]. Next, the KD-tree will be queried up to $O(n)$ times. This also runs in $O(n \log n)$, as each of the n queries will be found in $O(\log n)$. However, a large constant factor in the running time exists, because of having to find three points from two different rings. This is mitigated by downsampling the moving point cloud by only querying a small fraction of points sampled evenly in each of the k rings.

During one iteration of the frame-to-frame motion estimation, there are $O(m)$ elements in the cost function arising from visual features and $O(n)$ elements arising from ICP. The optimization problem is only of six parameters so the time taken to solve is only proportional to the number of residuals, which is $O(n + m)$.

Batch optimization using iSAM, should it be necessary to run, can take up to $O(t^3)$ for t frames, in the absolute worst case where an edge exists betwen every pair of frames. In practice, the pose-graph is sparse, and the adjacency matrix has only $O(t)$ nonzero elements, with only very few loop closure edges. Under certain conditions, e.g. when the pose graph is planar, iSAM is able to efficiently solve the system in $O(t^{1.5})$ [30]. We

note that, while pose estimation is running, generally only an incremental update, such as a Givens rotation, is needed [30].

6.2 Experimental Performance

To analyze the practical performance of the algorithm, Valgrind version 3.11.0 with Callgrind is used to profile the pipeline to determine the relative time taken by each function.

As table 3 shows, with single threaded feature correspondence matching, the quadratic time complexity results in this step being the most expensive. Experiments reveal that switching to the CUDA implementation improves performance by 100 times, effectively rendering this step instant.

With CUDA-based feature correspondence matching, the most expensive step is KD-tree construction and querying.

Frame-to-frame motion estimation tends to take approximately 0.5 s with 1500 visual features. Point set registration using ICP accounts for around 0.4 s out of that.

Each iSAM incremental update takes less than 0.1 s. At the end of each run, a batch optimization is performed, which takes about 10 s.

Step	Proportion of time taken
Feature correspondence matching (single threaded)	30.62%
Lidar KD-tree querying in point set registration	24.22%
Detecting visual features	0.88%
Autodifferentiation of cost function	0.72%
Projecting lidar points to camera	0.67%
Feature depth association	0.59%

Table 3: Profiling data for running frame-to-frame motion estimation using single threaded feature correspondence matching and one iteration of ICP per frame. Steps not shown are negligible in time consumed.

7 Evaluation

The VELO pipeline is evaluated on the KITTI data set [19]. This data set is collected by a car equipped with a Velodyne HDL-64E lidar and two Point Grey Flea 2 cameras driving in a variety of environments, including urban, highway, and orchard environments.

7.1 Front End Dead Reckoning Performance

The effectiveness of frame-to-frame motion estimation is tested without applying any form of pose-graph optimization, bundle adjustment, or the use of keyframes. The algorithm from section 4 is run on every consective pair of frames t and $t-1$. Even when using only dead reckoning, VELO performs reasonably well.

The translation error as reported by the KITTI evaluation tool is 1.48%.

Distance travelled (m)	Translation error (%)	Rotation error (rad)
100	1.1182	0.000176
200	1.2600	0.000147
300	1.4171	0.000131
400	1.5501	0.000119
500	1.6701	0.000109
600	1.7078	0.000101
700	1.6982	0.000095
800	1.6469	0.000089

Table 4: Error in translation and rotation evaluated over different lengths of travel, averaged over data sets 00 to 10.

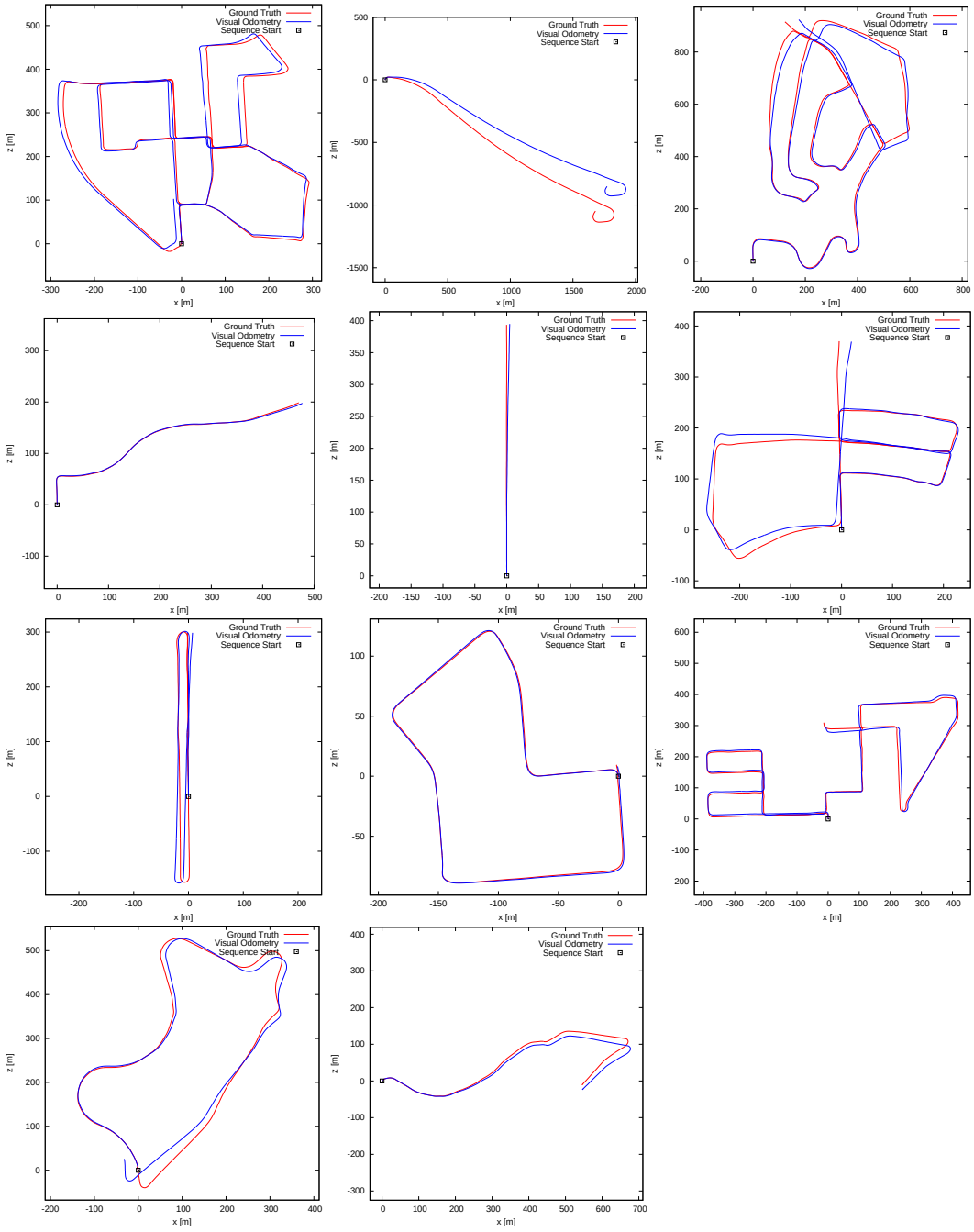

Figure 8: Dead reckoning trajectories of KITTI data sets 00 to 10.

7.2 Full Closed Loop Performance

The effectiveness of the full VELO pipeline is tested on the KITTI data set. Overall performance using the KITTI evaluation tool is 1.0199% mean translation error per unit distance travelled and 0.0051 $^\circ$ per meter mean rotation error.

Distance (m)	00	01	02	03	04	05
100	0.9177	0.8235	0.7316	0.7092	0.4799	0.3804
200	1.0556	0.7779	0.8327	0.9850	0.5472	0.4087
300	1.1010	0.8507	0.9414	1.6568	0.4844	0.3803
400	1.0974	0.9710	0.9957	2.4049	×	0.3470
500	1.0356	1.1181	1.0138	3.1633	×	0.2961
600	0.9464	1.3433	0.9980	×	×	0.2360
700	0.8666	1.6019	0.9689	×	×	0.1999
800	0.7922	1.8639	0.9520	×	×	0.1687
Distance (m)	06	07	08	09	10	Avg
100	0.4686	0.5625	1.3127	0.6870	0.6976	0.8083
200	0.7001	0.5191	1.6256	0.7387	1.0016	0.9536
300	0.8165	0.3859	1.9438	0.7769	1.1678	1.0758
400	0.7600	0.3063	2.0104	0.8241	1.3008	1.1250
500	0.4179	0.2275	1.9304	0.9115	1.4806	1.0998
600	0.2099	0.1125	1.9031	0.8893	1.5472	1.0588
700	0.1359	×	1.9693	0.7456	1.6687	1.0550
800	0.0557	×	2.0245	0.6245	1.4934	1.0428

Table 5: Translation error (%) for each data set evaluated over different lengths of travel.

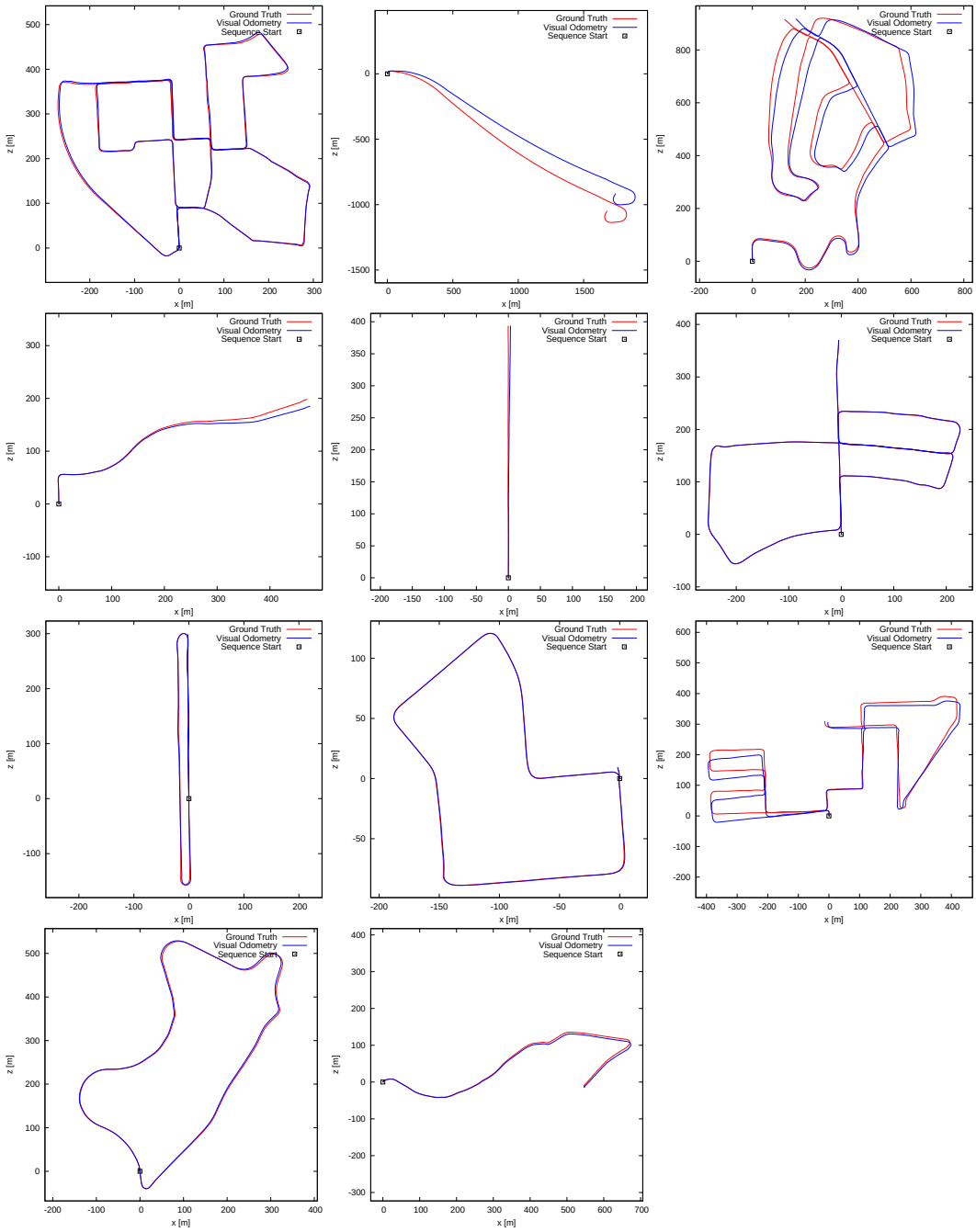

Figure 9: Loop closed, pose graph optimized KITTI data sets 00 to 10.

Distance (m)	Rotation error (rad)
100	0.000170
200	0.000113
300	0.000089
400	0.000075
500	0.000066
600	0.000060
700	0.000055
800	0.000050

Table 6: Error in rotation evaluated over different lengths of travel, averaged over data sets 00 to 10.

7.3 Discussion

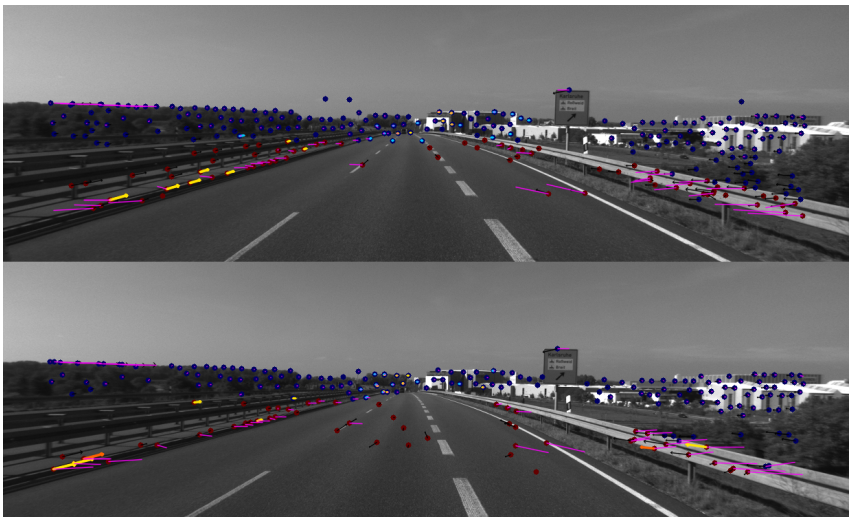

Figure 10: Failure case in visual odometry on data set 01 when ICP is disabled.

At a glance, performance on data set 01 seems to be worse than the others. The visual odometry component of the VELO pipeline appears to fail on the highway scene, for two reasons. First, the smooth appearance of the road means that few or no good features to track can be extracted. Second, many of the features occur on the railings on the side of the road. The railing has a repetitive appearance and suffers from the stroboscopic

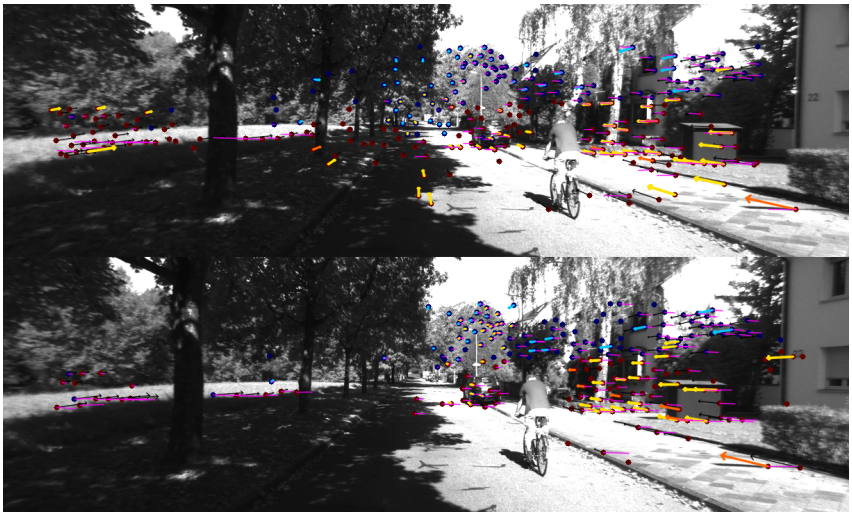

Figure 11: Failure case in visual odometry on data set 08 when ICP is disabled.

effect. The remaining features are found on distant objects, which are beyond the range of lidar, and also impossible to be reliably triangulated.

Fortunately, the tightly coupled approach of VELO allows the point-to-plane ICP to converge to a reasonable solution despite the poor performance of visual odometry. It is unclear how DEMO [64], which also relies on the OpenCV implementation of LKT sparse optical flow tracking, manages to perform well without using lidar scan matching.

The performance on data set 08 is also poor. From figure 11 appears that the dynamic range of the camera is not sufficient to correctly expose different parts of the image. As a consequence of the GFTT implementation used here, few or no reliable features are extracted in one half of the image. It is possible that using different GFTT thresholds for different grid cells in the image, such as in DEMO [64], will improve results by having a more even distribution of features in the image. It is possible that the large amount of foilage in the orchard scenes causes lidar measurements to be less reliable. More careful lidar feature selection, such as in LOAM [65], may improve results.

Finally, VELO relies on de-warped lidar point clouds provided by the KITTI data set, whereas LOAM [65] and V-LOAM [66] apply their own de-warping. It is possible that the KITTI point clouds contain distortions which cause VELO's ICP-based point set registration to perform worse than anticipated, although it appears that ICP-based loop closure is working as intended.

8 Conclusion

A new algorithm called Vision-Enhanced Lidar Odometry and Mapping (VELO), which fuses visual odometry and lidar odometry, is presented. Compared to previous approaches, VELO is able to function even when either lidar or the camera is not functioning reliably. Tests on the publicly available KITTI data set yields results comparable to state-of-the-art pose estimation systems.

Despite a sophisticated approach to feature tracking, inadequate visual feature reliability in certain data sets cause VELO to behave worse than the best state-of-the-art algorithms on the KITTI data set leaderboards. Nonetheless, thanks to pose graph optimization and lidar scan matching, VELO is able to output a reasonable solution even when feature tracking fails.

Lidar scan matching is advantageous in estimating transformations in loop closure due to the 360 $^\circ$ field of view of the lidar.

8.1 Future Work

Currently, VELO assumes that motion distortion in lidar point clouds have been adequately removed using an external pose source. This may not be the case in reality. For future work, motion distortion can be removed using the method of V-LOAM [66], which operates in multiple steps: first, 3D-2D and 2D-2D matching are used to estimate motion distortion, and then point set registration may be performed afterwards. Since VELO already uses 3D-2D and 2D-2D matching, it will be straightforward to implement this enhancement.

The feature tracking in VELO currently uses available implementations in OpenCV. However, other algorithms have managed to achieve impressive performance without lidar by tracking features in other ways [14][42]. By incorporating such approaches into the feature tracking pipeline of VELO, surely the performance will improve too.

The nonlinear optimization in VELO is prone to local minima. This is an inherent drawback of the Levenberg-Marquardt algorithm, and is exacerbated by the use of the arctangent loss function. As such, it requires a good initial guess. Random sample consensus may be used to improve frame-to-frame pose estimation even in the absence of a good initial guess.

Much of VELO is parallelizable. For example, during point set registration, each of the closest-point queries can be executed independently in parallel. Using n processors

(for example, in a graphics processing unit), the time complexity may be improved from $O(n \log n)$ to $O(\log n)$. As this step is the bottleneck in performance, such a drastic improvement will surely allow VELO to operate in real time. In addition, the back end and the front end may be run in parallel.

The cost functions used in frame-to-frame motion estimation do not take into account the different uncertainty in a 3D point in the azimuth, altitude, and range direction. Angular localization tends to be much better than range localization of a point. If these uncertainties were taken into account, the algorithm will be more robust.

For a self-driving vehicle in an urban environment, many cues may be used to improve pose estimation. Buildings tend to have many vertical lines, and this can be used to correct for rotational drift.

As further future work, it will be interesting to explore fusing dense stereo odometry methods, such as those similar to LSD-SLAM [16], with lidar data. Alternatively, deep learning methods such as [13] have recently become popular. Combining these with lidar-based pose estimation will be a unique challenge.

9 Acknowledgements

This thesis would not have been possible without the help of many people. Most of all I thank my advisor Dr. George Kantor for providing valuable insight and direction throughout my studies at Carnegie Mellon.

I also thank my thesis committee members Dr. Michael Kaess and Ji Zhang for overseeing my research and providing useful feedback.

I am grateful for my classmates and colleagues. I enjoyed collaborating with Guan-Horng Liu, Masa Uenoyama, and Srinivasan Vijayarangan on the Yamaha Viking project. I also thank my friends Jonathan Shen, Aram Ebtekar, Junxing Wang for the fun conversations and activities.

I also thank Dr. Danny Sleator for allowing me to participate in practice sessions for competitive programming, as well as two trips to the Association of Computing Machinery International Collegiate Programming Contest regionals as an unofficial contestant.

Finally, I thank my parents Ding Lu and Qing Li, and my sister Stephanie Lu for their ceaseless love and encouragement.

References

[1] Agarwal, Sameer, and Keir Mierle. “Ceres Solver”. http://ceres-solver.org.

[2] Ahuja, Satyajeet, and Steven Lake Waslander. “3D Scan Registration Using Curvelet Features.” *Computer and Robot Vision (CRV), 2014 Canadian Conference on.* IEEE, 2014.

[3] Alahi, Alexandre, Raphael Ortiz, and Pierre Vandergheynst. “Freak: Fast retina keypoint.” *Computer Vision and Pattern Recognition (CVPR), 2012 IEEE Conference on.* IEEE, 2012.

[4] Bay, Herbert, Tinne Tuytelaars, and Luc Van Gool. “Surf: Speeded up robust features.” *Computer vision—ECCV 2006.* Springer Berlin Heidelberg, 2006. 404-417.

[5] Besl, Paul J., and Neil D. McKay. “Method for registration of 3-D shapes.” *Robotics-DL tentative.* International Society for Optics and Photonics, 1992.

[6] Biber, Peter, and Wolfgang Straßer. “The normal distributions transform: A new approach to laser scan matching.” *Intelligent Robots and Systems, 2003.(IROS 2003).* Proceedings. 2003 IEEE/RSJ International Conference on. Vol. 3. IEEE, 2003.

[7] Bouguet, Jean-Yves. “Pyramidal implementation of the affine Lucas Kanade feature tracker description of the algorithm.” *Intel Corporation* 5.1-10 (2001): 4.

[8] Bradski, Gary. “The OpenCV library.” Doctor Dobb’s Journal of Software Tools 25.11 (2000): 120-126.

[9] Calonder, Michael, Vincent Lepetit, Christoph Strecha, and Pascal Fua. “Brief: Binary robust independent elementary features.” Computer Vision–ECCV 2010 (2010): 778-792.

[10] Caroli, Manuel, Pedro MM de Castro, Sébastien Loriot, Olivier Rouiller, Monique Teillaud, and Camille Wormser. “Robust and efficient delaunay triangulations of points on or close to a sphere.” *Experimental Algorithms.* Springer Berlin Heidelberg, 2010. 462-473.

[11] Chen, Yang, and Gérard Medioni. “Object modelling by registration of multiple range images.” *Image and vision computing* 10.3 (1992): 145-155.

[12] Chui, Haili, and Anand Rangarajan. “A new point matching algorithm for non-rigid registration.” *Computer Vision and Image Understanding* 89.2 (2003): 114-141.

[13] Costante, Gabriele, Michele Mancini, Paolo Valigi, and Thomas A. Ciarfuglia. “Exploring Representation Learning With CNNs for Frame-to-Frame Ego-Motion Estimation.” *Robotics and Automation Letters, IEEE* 1.1 (2016): 18-25.

[14] Cvišić, Igor, and Ivan Petrović. “Stereo odometry based on careful feature selection and tracking.” *Mobile Robots (ECMR), 2015 European Conference on*. IEEE, 2015.

[15] Das, Aruneema, James Servos, and Steven Lake Waslander. “3D scan registration using the normal distributions transform with ground segmentation and point cloud clustering.” *Robotics and Automation (ICRA), 2013 IEEE International Conference on.* IEEE, 2013.

[16] Engel, Jakob, Thomas Schöps, and Daniel Cremers. “LSD–SLAM: Large-scale direct monocular SLAM.” *Computer Vision—ECCV 2014.* Springer International Publishing, 2014. 834-849.

[17] Fitzgibbon, Andrew W. “Robust registration of 2D and 3D point sets.” *Image and Vision Computing* 21.13 (2003): 1145-1153.

[18] Gao, Xiao-Shan, Xiao-Rong Hou, Jianliang Tang, and Hang-Fei Cheng. *IEEE transactions on pattern analysis and machine intelligence* 25.8 (2003): 930-943.

[19] Geiger, Andreas, Philip Lenz, Christoph Stiller, and Raquel Urtasun. “Vision meets robotics: The KITTI dataset.” *The International Journal of Robotics Research* (2013): 0278364913491297.

[20] Gelfand, Natasha, Niloy J. Mitra, Leonidas J. Guibas, and Helmut Pottmann. “Robust global registration.” *Symposium on geometry processing.* Vol. 2. No. 3. 2005.

[21] Granger, Sébastien, and Xavier Pennec. “Multi-scale EM-ICP: A fast and robust approach for surface registration.” *Computer Vision—ECCV* (2002): 69-73.

[22] Guennebaud, Gaël, Benoît Jacob. *Eigen v3*, http://eigen.tuxfamily.org. 2010.

[23] Harris, Chris, and Mike Stephens. “A combined corner and edge detector.” *Alvey vision conference.* Vol. 15. 1988.

[24] Hartley, Richard, and Andrew Zisserman. *Multiple view geometry in computer vision.* Cambridge university press, 2003.

[25] Henry, Peter, Michael Krainin, Evan Herbst, Xiaofeng Ren, and Dieter Fox. "RGB-D mapping: Using Kinect-style depth cameras for dense 3D modeling of indoor environments." *The International Journal of Robotics Research* 31.5 (2012): 647-663.

[26] Horaud, Radu, Bernard Conio, Olivier Leboulleux, and Bernard Lacolle. "An analytic solution for the perspective 4-point problem." *Computer Vision and Pattern Recognition, 1989. Proceedings CVPR'89., IEEE Computer Society Conference on*. IEEE, 1989.

[27] Jian, Bing, and Baba C. Vemuri. "Robust point set registration using gaussian mixture models." *Pattern Analysis and Machine Intelligence, IEEE Transactions on* 33.8 (2011): 1633-1645.

[28] Johnson, Andrew Edie, and Sing Bing Kang. "Registration and integration of textured 3D data." *Image and vision computing* 17.2 (1999): 135-147.

[29] Johnson, Andrew E., and Martial Hebert. "Using spin images for efficient object recognition in cluttered 3D scenes." *Pattern Analysis and Machine Intelligence, IEEE Transactions on* 21.5 (1999): 433-449.

[30] Kaess, Michael, Ananth Ranganathan, and Frank Dellaert. "iSAM: Incremental smoothing and mapping." *IEEE Transactions on Robotics* 24.6 (2008): 1365-1378.

[31] Kaess, Michael, and Frank Dellaert. "Covariance recovery from a square root information matrix for data association." *Robotics and autonomous systems* 57.12 (2009): 1198-1210.

[32] Kümmerle, Rainer, Giorgio Grisetti, Hauke Strasdat, Kurt Konolige, and Wolfram Burgard. "g^{2}o: A general framework for graph optimization." *Robotics and Automation (ICRA), 2011 IEEE International Conference on*. IEEE, 2011.

[33] Lepetit, Vincent, Francesc Moreno-Noguer, and Pascal Fua. "EPnP: An accurate $O(n)$ solution to the PnP problem." *International Journal of Computer Vision* 81.2 (2009): 155-166.

[34] Leutenegger, Stefan, Margarita Chli, and Roland Y. Siegwart. "BRISK: Binary robust invariant scalable keypoints." *Computer Vision (ICCV), 2011 IEEE International Conference on.* IEEE, 2011.

[35] Leutenegger, Stefan, Simon Lynen, Michael Bosse, Roland Siegwart, and Paul Furgale. “Keyframe-based visual–inertial odometry using nonlinear optimization.” *The International Journal of Robotics Research* 34.3 (2015): 314-334.

[36] Lowe, David G. “Object recognition from local scale-invariant features.” *Computer vision, 1999. The proceedings of the seventh IEEE international conference on.* Vol. 2. Ieee, 1999.

[37] Lucas, Bruce D., and Takeo Kanade. “An iterative image registration technique with an application to stereo vision.” *Artificial Intelligence, 1981. Proceedings of the 7th International Conference on (IJCAI).* Vol. 81. 1981.

[38] Magnusson, Martin. “The three-dimensional normal-distributions transform: an efficient representation for registration, surface analysis, and loop detection.” (2009).

[39] Miksik, Ondrej, and Krystian Mikolajczyk. “Evaluation of local detectors and descriptors for fast feature matching.” *Pattern Recognition (ICPR), 2012 21st International Conference on.* IEEE, 2012.

[40] Mücke, Ernst P., Isaac Saias, and Binhai Zhu. “Fast randomized point location without preprocessing in two-and three-dimensional Delaunay triangulations.” *Proceedings of the twelfth annual symposium on Computational geometry.* ACM, 1996.

[41] Muja, Marius, and David G. Lowe. “Flann, fast library for approximate nearest neighbors.” *International Conference on Computer Vision Theory and Applications* (VISAPP’09). INSTICC Press, 2009.

[42] Mur-Artal, Raul, J. M. M. Montiel, and Juan D. Tardós. “Orb-slam: a versatile and accurate monocular slam system.” *IEEE Transactions on Robotics* 31.5 (2015): 1147-1163.

[43] Myronenko, Andriy, and Xubo Song. “Point set registration: Coherent point drift.” *Pattern Analysis and Machine Intelligence, IEEE Transactions on* 32.12 (2010): 2262-2275.

[44] Nistér, David. “An efficient solution to the five-point relative pose problem.” *Pattern Analysis and Machine Intelligence, IEEE Transactions on* 26.6 (2004): 756-770.

[45] Rodrigues, Olinde. *De l’attraction des sphéroïdes*. Diss. 1815.

[46] Rosen, David M., Michael Kaess, and John J. Leonard. “An incremental trust-region method for robust online sparse least-squares estimation.” *Robotics and Automation (ICRA), 2012 IEEE International Conference on*. IEEE, 2012.

[47] Rosten, Edward, and Tom Drummond. “Machine learning for high-speed corner detection.” *Computer Vision–ECCV 2006.* Springer Berlin Heidelberg, 2006. 430-443.

[48] Rublee, Ethan, Vincent Rabaud, Kurt Konolige, and Gary Bradski. “ORB: an efficient alternative to SIFT or SURF.” *Computer Vision (ICCV), 2011 IEEE International Conference on.* IEEE, 2011.

[49] Rusinkiewicz, Szymon, and Marc Levoy. “Efficient variants of the ICP algorithm.” *3-D Digital Imaging and Modeling, 2001. Proceedings. Third International Conference on.* IEEE, 2001.

[50] Rusu, Radu Bogdan, Nico Blodow, and Michael Beetz. “Fast point feature histograms (FPFH) for 3D registration.” *Robotics and Automation, 2009. ICRA’09. IEEE International Conference on.* IEEE, 2009.

[51] Rusu, Radu Bogdan, and Steve Cousins. “3d is here: Point cloud library (pcl).” *Robotics and Automation (ICRA), 2011 IEEE International Conference on.* IEEE, 2011.

[52] Segal, Aleksandr, Dirk Haehnel, and Sebastian Thrun. “Generalized-ICP.” *Robotics: Science and Systems.* Vol. 2. No. 4. 2009.

[53] Shi, Jianbo, and Carlo Tomasi. “Good features to track.” *Computer Vision and Pattern Recognition, 1994. Proceedings CVPR’94., 1994 IEEE Computer Society Conference on.* IEEE, 1994.

[54] Smith, Randall, Matthew Self, and Peter Cheeseman. “Estimating uncertain spatial relationships in robotics.” *Autonomous robot vehicles.* Springer New York, 1990. 167-193.

[55] Steder, Bastian, Radu Bogdan Rusu, Kurt Konolige, and Wolfram Burgard. “NARF: 3D range image features for object recognition.” *Workshop on Defining and Solving Realistic Perception Problems in Personal Robotics at the IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS).* Vol. 44. 2010.

[56] Stoyanov, Todor Dimitrov, Martin Magnusson, Henrik Andreasson, and Achim Lilienthal. “Fast and accurate scan registration through minimization of the distance between compact 3D NDT representations.” *The International Journal of Robotics Research* (2012): 0278364912460895.

[57] Tange, Ole. “Gnu parallel-the command-line power tool.” *The USENIX Magazine* 36.1 (2011): 42-47.

[58] Tomasi, Carlo, and Takeo Kanade. “Shape and motion from image streams under orthography: a factorization method.” *International Journal of Computer Vision* 9.2 (1992): 137-154.

[59] Triggs, Bill, Philip F. McLauchlan, Richard I. Hartley, and Andrew W. Fitzgibbon. “Bundle adjustment–a modern synthesis.” *International workshop on vision algorithms.* Springer Berlin Heidelberg, 1999.

[60] Tsin, Yanghai, and Takeo Kanade. “A correlation-based approach to robust point set registration.” *Computer Vision—ECCV* 2004. Springer Berlin Heidelberg, 2004. 558-569.

[61] Weiszfeld, Endre. “Sur le point pour lequel la somme des distances de n points donnés est minimum.” *Tohoku Math. J* 43.355-386 (1937): 2.

[62] Whelan, Thomas, Stefan Leutenegger, Renato F. Salas-Moreno, Ben Glocker, and Andrew J. Davison. “ElasticFusion: Dense SLAM without a pose graph.” *Proceedings of Robotics: Science and Systems (RSS).* 2015.

[63] Yang, Jiaolong, Hongdong Li, and Yunde Jia. “GO-ICP: Solving 3d registration efficiently and globally optimally.” *Proceedings of the IEEE International Conference on Computer Vision.* IEEE, 2013.

[64] Zhang, Ji, Michael Kaess, and Sanjiv Singh. “Real-time depth enhanced monocular odometry.” *Intelligent Robots and Systems (IROS 2014), 2014 IEEE/RSJ International Conference on.* IEEE, 2014.

[65] Zhang, Ji, and Sanjiv Singh. “Loam: Lidar odometry and mapping in real-time.” *Robotics: Science and Systems Conference (RSS).* 2014.

[66] Zhang, Ji, and Sanjiv Singh. “Visual-lidar odometry and mapping: Low-drift, robust, and fast.” *Robotics and Automation (ICRA), 2015 IEEE International Conference on.* IEEE, 2015.

[67] Zlot, Robert, and Michael Bosse. “Efficient large-scale 3D mobile mapping and surface reconstruction of an underground mine.” *Field and service robotics.* Springer Berlin Heidelberg, 2014.