Autonomous Robots
https://doi.org/10.1007/s10514-018-9794-6

Efficient Velodyne SLAM with point and plane features

W. Shane Grant[1] · **Randolph C. Voorhies**[1] · **Laurent Itti**[1,2]

Received: 27 October 2016 / Accepted: 27 July 2018

Abstract
This paper develops and tests a plane based simultaneous localization and mapping algorithm capable of processing the uneven sampling density of Velodyne-style scanning LiDAR sensors in real-time. The algorithm uses an efficient plane detector to rapidly provide stable features, both for localization and as landmarks in a graph-based SLAM. When planes cannot be detected or when they provide insufficient support for localization, a novel constraint tracking algorithm selects a minimal set of supplemental point features to be provided to the localization solver. Several difficult indoor and outdoor datasets, totaling 6981 scans, each with ∼70,000 points, are used to analyze the performance of the algorithm without the aid of any additional sensors. The results are compared to two competing state-of-the-art algorithms, GICP and LOAM, showing up to an order of magnitude faster runtime and superior accuracy on all datasets, with loop closure errors of 0.14–0.95 m, compared to 0.44–66.11 m.

Keywords SLAM · Velodyne · Point cloud · ICP · Planes

1 Introduction

The recent availability of high performance sensors suitable for indoor range measurement such as RGB-D cameras and nodding LiDAR sensors has led to many successful simultaneous localization and mapping (SLAM) systems for indoor environments. Outdoor environments have proven challenging as many of the algorithms designed for these indoor sensors have difficulty when adapted to outdoor sensors, which often collect data at much greater distances and lower densities. As robotics applications move out of the laboratory and into the real world, the need for reliable outdoor-capable algorithms is rising.

This work develops a fast online SLAM solution around a Velodyne LiDAR, which is able to produce robust range measurements both indoors and outdoors. The algorithm leverages a fast plane detector to exploit structure in the environment while maintaining the capability of selectively sampling and integrating lower level features in less ordered environments. The planar features are used as landmarks in a global SLAM map to close the loop on several indoor and outdoor datasets with varying degrees of planar structure. The algorithm is able to operate on non-smooth trajectories, and is suitable for use on humanoid and aerial robots without the need for odometry measurements. An important contribution in this work is the development of a constraint tracking algorithm that minimizes the computational load of frame to frame matching.

W. Shane Grant and Randolph C. Voorhies have contributed equally to this work.

This work was supported by the National Science Foundation (Grant Numbers CCF-1317433 and CNS-1545089), C-BRIC (one of six centers in JUMP, a Semiconductor Research Corporation (SRC) program sponsored by DARPA), the Army Research Office (W911NF-12-1-0433), the Office of Naval Research (N00014-13-1-0563), and the Intel Corporation. The authors affirm that the views expressed herein are solely their own, and do not represent the views of the United States government or any agency thereof.

✉ W. Shane Grant
wgrant@usc.edu

Randolph C. Voorhies
voorhies@usc.edu

Laurent Itti
itti@pollux.usc.edu

[1] Department of Computer Science, University of Southern California, Los Angeles, CA 90089, USA

[2] Department of Psychology and Neuroscience Graduate Program, University of Southern California, Los Angeles, CA 90089, USA

Published online: 09 August 2018

2 Related work

Point cloud driven SLAM solutions have been a heavily researched topic in robotics for the past several years. Approaches to the problem have ranged on a spectrum from low level solutions, which continually match and integrate entire scans into a global point cloud, to feature-based solutions, which extract high level descriptors of the environment to use as landmarks.

Given enough computational power, the former approach has been shown to work well for small and medium sized environments when using structured light sensors. For example, the KinectFusion algorithm of Newcombe et al. (2011) can create dense reconstructions of small indoor environments using a Microsoft Kinect (or equivalent RGB-D sensor) and an array of high powered graphics processing units (GPU). Later extensions by Whelan et al. (2012) extend the map size by continually offloading map data from GPU memory to system RAM. Although such algorithms have set a high bar for indoor localization, the relatively short range of current RGB-D sensors and their sensitivity to ambient infrared interference limits their applicability to outdoor mapping. Additionally, such algorithms require significant processing power in the form of high-end GPUs that are currently rare on mobile robots due to their size and power requirements. Other low-level solutions, such as using sparse features, also exist for RGB-D sensors (e.g., Henry et al. 2012; Endres et al. 2012). While these algorithms lower the computational burden by first extracting keypoints from each frame before performing matching on a limited subset of locations, they are still hindered by the limitations of RGB-D sensors, which limits their applicability to outdoor environments. Another recent RGB-D SLAM implementation (Salas-Moreno et al. 2014) uses planes as features. However, in addition to RGB-D sensor limitations, it is intended primarily for indoor augmented reality applications and lacks capabilities (e.g., loop closure) that become necessary in larger environments.

LiDAR sensors, which provide much longer range distance readings and have lower sensitivity to infrared interference from sunlight, have proven popular for general indoor and outdoor applications. Planar 2D LiDARs, which are able to capture a slice of the environment in a single pass, have proven very successful in real-time 2D SLAM (Hahnel et al. 2003; Grisetti et al. 2005, 2007). Such sensors can also be mounted on a 'nodding' servo mechanism that allows for full 3D scans of the environment. These scans can offer very high resolution, though often with very slow scanning speed. Additionally, these scans are typically captured in a stop-scan-go fashion which hinders their ability to be used for activities requiring real-time interaction with the environment. Planar features have proven very useful for these sensors, and have been integrated into several SLAM solutions (Weingarten and Siegwart 2005, 2016; Pathak et al. 2010). However, due to the slow scanning speed of nodding LiDAR sensors, which can take on the order of seconds per scan, previous efforts have not required high speed algorithms.

Continuously rotating 2D LiDAR, which capture a single scanline in up to 360° degrees of rotation, can provide a large benefit over their planar counterparts to real-time applications such as autonomous driving. These sensors were widley used in in the DARPA Grand Challenge (Behringer et al. 2005; Urmson et al. 2007). However, the limited resolution provided by a single scanline often necessitates combining multiple 2D LiDAR along with a variety of other sensor modalities (e.g.,Choi et al. 2012).

As an alternative to 2D LiDAR sensors, 3D scanning LiDARs have recently become popular, due in part to their applicability to autonomous driving, as demonstrated in the DARPA Grand Challenges. By rotating several laser/receiver units with fixed inclination around a central vertical axis, these sensors can acquire full 3D scans at a rate of roughly 100 milliseconds per scan. The trade-off for this high speed is a relatively low resolution in the inclination direction, with one sample per laser/receiver pair, compared to their nodding counterparts. Current models comprise 16, 32, or 64 lasers spanning approximately 40° in inclination. Because of this uneven sampling density, most systems designed for RGB-D or nodding sensors, which collect uniformly dense measurements, do not adapt well. The work of Moosmann and Stiller (2011) has been directed specifically to Velodyne sensors and operates in a spirit similar to the KinectFusion family of algorithms, in that each current scan is matched against and then integrated into a global point cloud map. Generalized ICP (GICP) (Segal et al. 2009), when used with graph based SLAM formulations, has shown some promise on Velodyne derived datasets (Trevor et al. 2014). However, as will be shown in Sect. 4, this strategy is often too slow for real-time processing and requires additional information from other sensors (e.g., IMU-derived odometry or GPS) to work well. More recently, Ceriani et al. (2015) have developed a Velodyne-capable SLAM algorithm using a point-plane ICP that is refined by a sliding window optimizer, though when using a global trajectory optimizer for loop closure, it is not suitable for real-time processing. Additionally, LOAM (Zhang and Singh 2014a) is a fast algorithm that uses line and planar features extracted from point clouds to perform mapping. Real time performance is achieved by leveraging a fast frame to frame odometry estimate along with a slower alignment to a global map. However, as will be seen in Sect. 4, their approach seems sensitive to fast rotational motion as well as less structured environments, and may need to be supplemented with inertial data for robust operation.

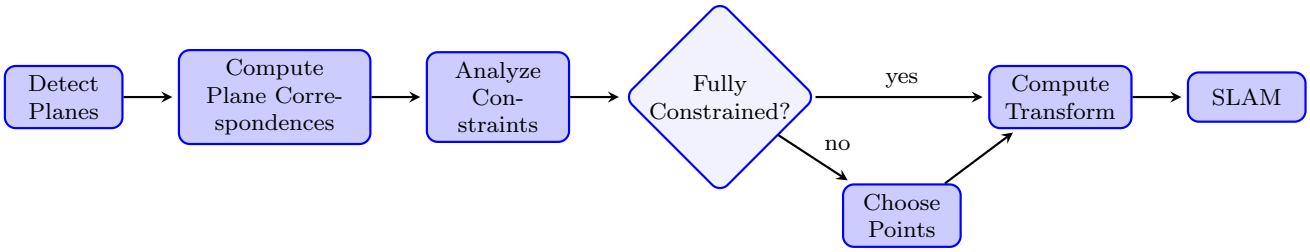


Fig. 1 System overview

3 Approach

The algorithm developed in this section is designed to address the shortcomings of previous works: it is capable of running indoors and outdoors, it can tolerate the uneven sampling density of 3D scanning LiDAR sensors, and it is fast enough to operate in real-time for use on mobile robotics platforms. This is chiefly accomplished by observing that in many human accessible environments, planar surfaces are abundant. The algorithm thus exploits these planes to both speed up processing and provide strong constraints on motion, while maintaining the ability to integrate points when planes are sparse or provide insufficient constraints.

3.1 System overview

The algorithm aims to reduce the computational load of traditional point based alignment schemes by leveraging planar features extracted from structured environments. As such, the first step of the system is to extract these features via a fast and robust plane detector (Sect. 3.2) tuned specifically for scanning LiDAR sensors (Grant et al. 2013). These plane correspondences form the basis on which the rest of the algorithm is computed.

Once all planar features have been extracted from two consecutive frames, the constraint imposed by their correspondences is evaluated by a unique constraint tracking algorithm (Sect. 3.3). If the planes alone can satisfy the constraints and provide a correspondence solution, a fast closed form algorithm is used to compute the transformation (Sect. 3.5.1). If the constraint tracking algorithm determines that the solution would be under-constrained given only the available plane correspondences, a minimal number of additional point correspondences are found (Sect. 3.4) to fulfill the remaining alignment constraints. Both the plane and point correspondences are then used to robustly estimate a rigid body transformation between the two frames (Sect. 3.5.2). This transformation estimation, along with the plane correspondences, is used to build an online pose graph SLAM map (Sect. 3.6). A diagram of these operations is shown in Fig. 1, and each operation is detailed separately in the following sections.

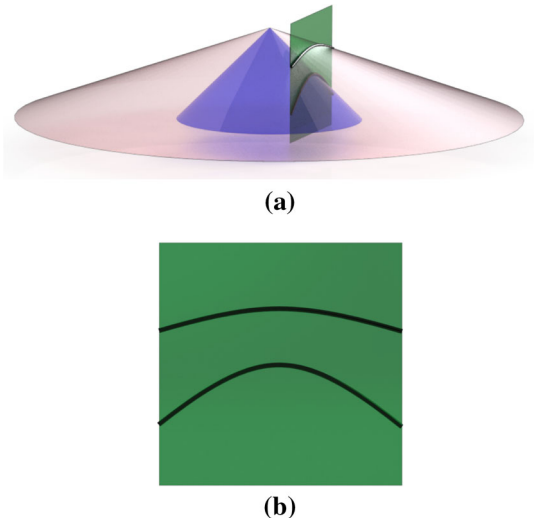


Fig. 2 The physical design of the sensor causes it to create virtual laser cones as it sweeps each of its lasers through θ (**a**). When a plane intersects one of these cones, it generates a conic section of varying curvature depending upon the inclination angle of the sensor and the relative orientation of the plane (**b**). **a** Perspective view of laser cones (red and blue) intersecting an actual, physical vertical plane (green). **b** Frontal view (through plane) of laser scan-lines (black) intersecting the physical plane (green) (Color figure online)

3.2 Plane detection and correspondence

The plane finding algorithm exploits knowledge of the sensor geometry to quickly find planes: the 3D LiDAR sensor's rotating lasers, which form cones in space, inscribe conic sections onto planar objects, as illustrated in Fig. 2. Each conic section found in a scanline of a sensor sweep is then allowed to vote for all possible planes that could have produced it. These votes are cast in a layered spherical accumulator, which is parameterized with the inclination (ϕ) and azimuth (θ) of the normal vector and the offset (ρ) of the plane as described in Borrmann et al. (2011).

Given a single conic section, the set of explanatory planes is found by walking across all values of ρ in the accumulator and solving for the two $\langle\theta, \phi\rangle$ pairs at each value of ρ. A vote is then placed in the accumulator for each value weighted by how well the putative plane fits into the curvature of the section. A highly curved conic section will give a high score

to planes that fit its points well, and a penalty to all others. A conic section with lower curvature will give a more uniform score to all planes. Once all voting has been completed, the accumulator is thresholded to find candidate planes, and a set of simple filtering steps is used to clean up the results. Planes are considered in correspondence if for two planes $\boldsymbol{p_a}$ and $\boldsymbol{p_b}$, $\|\rho_a \hat{\mathbf{n}}_a - \rho_b \hat{\mathbf{n}}_b\|$ is minimized (where $\hat{\mathbf{n}}$ is the plane normal), and the following two constraint equations are satisfied:

$$\hat{\mathbf{n}}_a \cdot \hat{\mathbf{n}}_b > t_{dot} \tag{1}$$

$$\frac{|\rho_a - \rho_b|}{(\rho_a + \rho_b)} < t_{dist} \tag{2}$$

A fully detailed description and analysis of this plane detection algorithm, as well as of the plane correspondence matcher and its parameters, can be found in Grant et al. (2013).

3.3 Constraint analysis

While it has been shown that the above plane detection and correspondence algorithms can be used to compute frame to frame registration in well constrained indoor environments (Grant et al. 2013), outdoor environments prove to be more challenging when planes are sparse. By detecting when an alignment problem is under-constrained by the available planes, a minimal set of points can be selected from the remaining points (points not constituent of corresponding planes) in the input cloud to fill in the missing constraints.

Fully constraining a 6D alignment problem using only infinite planes requires at least three non-coplanar planes. The level of constraint of the system can be visualized by a 3D ellipsoid, as shown in Fig. 3, in which the length of each axis represents the amount of constraint along that direction. For example, a circle in the X-Y plane would represent an equal amount of constraint in the X and Y directions, with no constraint in the Z, while a perfect sphere would represent equal constraint in all directions. This ellipsoid can be parameterized by a 3×3 matrix that is called the Constraint Matrix ($\mathcal{C}$). Once the plane correspondences are discovered in the scene, $\mathcal{C}$ is constructed from each plane normal as follows:

$$\mathcal{C} = \sum_{i=0}^{N} \gamma_i n_i n_i^T \tag{3}$$

where γ_i is the number of points contained in the plane, and n_i is the plane normal vector.

After the constraint matrix is built, the bounding ellipsoid can be parameterized by a rotation matrix $\mathcal{R}_c$ and a set of three radii $r_{c_0}, r_{c_1}, r_{c_2}$. This parameterization is extracted

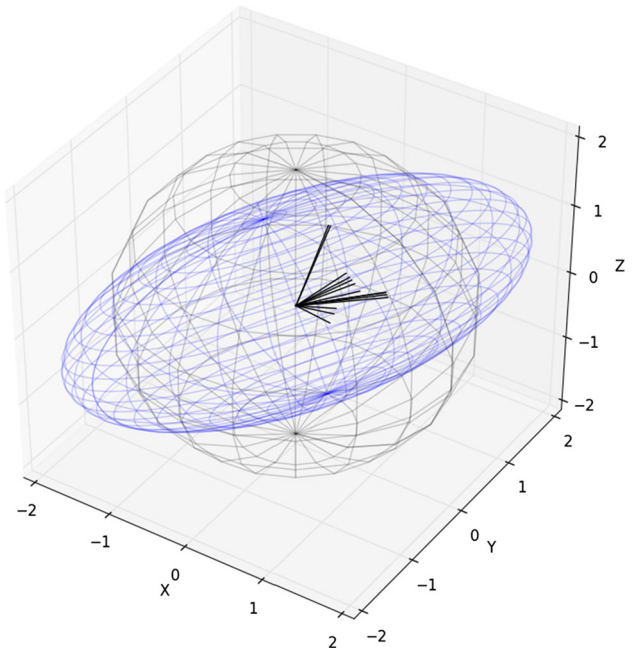


Fig. 3 A visualization of the constraint matrix $\mathcal{C}$ (blue ellipsoid) for an arbitrary collection of scaled plane normals (black vectors). The black sphere, parameterized by $\hat{\tau}$, represents the threshold required for the system to be considered well constrained. In this example, the system is well constrained (ellipsoid exceeds volume of sphere) along the X and Y dimension, but lacks constraint in the Z dimension. Equation 8 provides a numerical way of calculating how well constrained any particular query vector is (Color figure online)

from $\mathcal{C}$ through singular value decomposition:

$$\mathcal{C} \overset{\text{SVD}}{=} U \begin{bmatrix} r_{c_0}^2 & 0 & 0 \\ 0 & r_{c_1}^2 & 0 \\ 0 & 0 & r_{c_2}^2 \end{bmatrix} \mathcal{R}_c \tag{4}$$

This formulation is similar to Principal Component Analysis, except that, rather than using a true normalized covariance matrix, a weighted scatter matrix is used. Once the constraint ellipsoid has been calculated, the system can (if needed) be queried to determine how well constrained the system is in the direction of a new candidate data point. If the system is already fully constrained in that direction, the data point can be skipped to save computation time. This constraint checking is performed as follows: given a query normal n of unit length, it is first transformed into the axis-aligned frame of the ellipsoid:

$$\tilde{n} = \mathcal{R}_c^T n \tag{5}$$

and is then converted into polar coordinates:

$$\langle \theta, \phi \rangle = \langle \arctan(\tilde{n}_1, \tilde{n}_0), \arccos(\tilde{n}_2) \rangle \tag{6}$$

The spherical coordinate ellipsoid equation can then be used to find the distance from the origin to the edge of the ellipsoid in the direction of the query normal:

$$d_q = \sqrt{\frac{\cos(\theta)^2 \sin(\phi)^2}{r_{c_0}^2} + \frac{\sin(\theta)^2 \sin(\phi)^2}{r_{c_1}^2} + \frac{\cos(\phi)^2}{r_{c_2}^2}} \quad (7)$$

If a fully constrained system is considered as a sphere of radius $\hat{\tau}$, then a score between 0 and 1 can be assigned to any query vector to represent the utility of adding that vector to the system:

$$s_q = 1 - \min\left(\frac{d_q}{\hat{\tau}}, 1\right) \quad (8)$$

The value of $\hat{\tau}$ (as well as d_q) can also be thought of as representing the number of points supporting one dimension of the constraint matrix C, due to the scaling by γ_i in its construction. This provides an intuitive guide for selecting a value for $\hat{\tau}$, which can be interpreted to mean the minimum number of points from a sensor scan required to constrain one dimension. In the later experiments, $\hat{\tau}$ is set to 1.4% (1000) of the input points per scan (70,000). The ultimate value chosen for this parameter depends mostly upon the sensor. Choosing too large of a constraint threshold will mean the inclusion of more points than necessary, increasing run-time, while too small of a constraint limits the accuracy of the solution. Although this constraint matrix algorithm provides no theoretical guarantee of optimality, Sect. 3.4 presents a detailed analysis of its practical value.

3.4 Choosing points

Once the constraint matrix has been constructed for all plane correspondences in the current frame, point features may need to be added to the system to provide sufficient constraints. To do so, points are treated as planelet features using the covariance of local neighborhoods, with a matching function inspired from Generalized ICP (Segal et al. 2009). Due to the high computational cost of this covariance computation and cost function, a method utilizing the constraint ellipse of Sect. 3.3 is used to keep the set of selected points is kept to the minimum necessary to provide alignment constraint.

The method for this selection is shown in Algorithm 3.1. In this algorithm, a very fast normal vector estimation is first computed for each point in the input point set $\mathcal{Q}$ using the Fast Approximate Least Squares (FALS) method of Badino et al. (2011). This method was chosen over computing normals precisely (i.e., using principle component analysis) because it gives a dramatic runtime performance improvement at little to no cost in the accuracy of computed normals. Next, for each point in the input a utility score s is calculated for these normals given the input constraint matrix $\mathcal{C}$ and threshold $\hat{\tau}$

Algorithm 3.1 The choosePoints algorithm - a subfunction of IC3PO. This method chooses a subset of point features to provide full rigid body constraints.

inputs
  $\mathcal{C}$ a constraint matrix computed for all plane correspondences
  $\mathcal{Q}$ an input point cloud
  $\hat{\tau}$ a constraint threshold
local variables
  $\mathcal{N}$ a list of normal vectors
  S a list of scores
  $\tilde{\mathcal{Q}}$ the chosen points
1: $\mathcal{N} \leftarrow$ FALS($\mathcal{Q}$)
2: **S** $\leftarrow \varnothing$
3: **for** i **in** `indices`($\mathcal{N}$) **do**
4: Calculate initial s given $\mathcal{C}$, $\mathcal{N}_i$, and $\hat{\tau}$ ▷ Eq's 5–8
5: $\mathbf{S}_i \leftarrow s$
6: **end for**
7: Sort $\mathcal{Q}$ and $\mathcal{N}$ in increasing order of **S**
8: $\tilde{\mathcal{Q}} \leftarrow \varnothing$
9: **for** i **in** `indices`($\mathcal{N}$) **do**
10: Recalculate s given $\mathcal{C}$, $\mathcal{N}_i$, and $\hat{\tau}$ ▷ Eq's 5–8
11: **if** BERNOULLI(s) **then**
12: add $\mathcal{N}_i$ to $\mathcal{C}$ with $\gamma = 1$ ▷ Eq. 3
13: CALCULATECOVARIANCE($\mathcal{Q}_i$)
14: $\tilde{\mathcal{Q}} \leftarrow \tilde{\mathcal{Q}} \cup \mathcal{Q}_i$
15: Calculate r_2 from $\mathcal{C}$ ▷ Eq. 4
16: **if** $r_2 > \hat{\tau}$ **then**
17: **break**
18: **end if**
19: **end if**
20: **end for**
21: **return** $\tilde{\mathcal{Q}}$

using Eqs. (5)–(8) on Lines 3–6. This provides an initial estimate of the utility of each point. The points are then iterated through in order of this initial utility estimate.

For each point, a new utility score s is calculated before deciding whether the point should be included in the returned set $\tilde{\mathcal{Q}}$. This new score is necessary because the algorithm updates $\mathcal{C}$ with each included point. The decision to include a point in $\tilde{\mathcal{Q}}$ could be done in multiple ways, such as choosing points exceeding a threshold or accepting the top N points. A Bernoulli distribution with $p = s$ is sampled to determine if the point is to be chosen - this is faster than repeatedly recomputing scores for sorting and gives preference to high scoring points. If a point is chosen, it is inserted into $\tilde{\mathcal{Q}}$ and a neighborhood covariance matrix is computed for it and attached as metadata on Algorithm Lines 13 and 14. The constraint matrix $\mathcal{C}$ is also updated with the point. If the resulting smallest radius of the constraint ellipse is greater than the constraint threshold $\hat{\tau}$, then the set of selected points is finalized and returned.

Figure 4 provides a visualization of how the selection of points adapts to the presence of planes, and how it differs from a uniformly random selection of points. Algorithm 3.1 predominantly selects points that maximize the constraint along the least constrained dimension, though the random

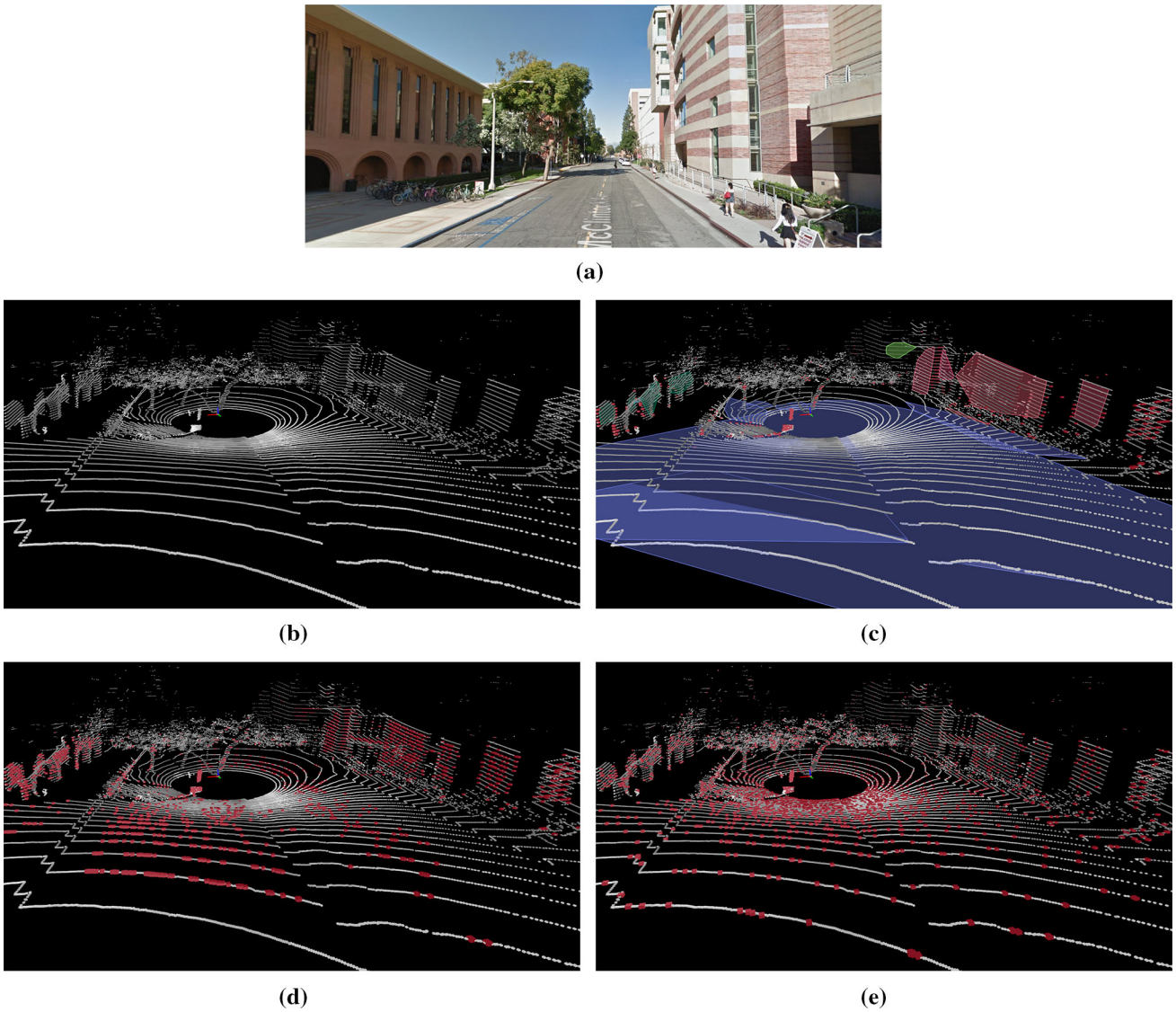


Fig. 4 Comparison of points selected for iterative closest point in an outdoor street scene for various algorithms. Points are shown in red for all figures. Best viewed electronically with figure enlarged. **a** Picture of scene. **b** Raw Velodyne data. Sensor position indicated by tricolor axis. The sensor is located in front of the light pole on the left sidewalk, visible in **a**. **c** Points selected by Algorithm 3.1 given the restraints of detected planes (shaded polygons in figure). In this frame, the plane detector has only provided a single plane (green plane in background) along the X axis (red axis of sensor), which provides insufficient constraint, so additional points are selected predominantly to constrain this direction. This can be seen by the large number of point selections on surfaces along this axis, which is perpendicular to the camera viewpoint. Points can be seen on the walls of buildings to the right, as well as along the tree trunk, light pole, and sign post near the bike rack to the left. **d** Points selected by Algorithm 3.1 given no initial plane constraints. Selected points most occur on flat, orthogonal surfaces of objects. The majority of points occur either on the street (ground plane), the walls facing the street, or on the same surfaces as selected in **c**. **e** A uniformly random selection of points equal in number to those chosen by Algorithm 3.1 in **d**. Given the structure of the sensor data, the majority of these points are chosen close to the origin. In addition, many of the points selected occur on less structured portions of the data, such as on the leaves of trees, and distant features are under represented due to the sparsity of points on them

component to the algorithm does allow a small amount of other points to be selected. Compare Fig. 4c, d to see how planes provide constraints which drastically reduce the number of points that must be selected. Fig. 4d contrasts strongly with Fig. 4e, which is a uniform selection of points. Due to the structure of the raw data (Fig. 4b), uniformly subsampling points leads to an overabundance of points around the origin, points selected on 'noisy' landmarks such as the leaves of trees, and an undersampling of distant but stable features such as walls.

To test the efficacy of Algorithm 3.1 in isolation, it was used to select points to perform alignment between two point clouds using GICP. One hundred frames were randomly selected from an outdoor dataset used in the later experi-

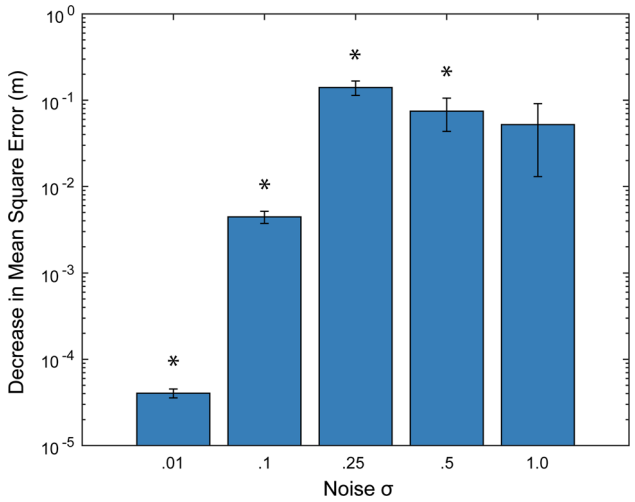


Fig. 5 The decrease of mean squared error (MSE) in alignment when using Algorithm 3.1 to select points versus a uniformly random selection of points. The error was calculated by comparing the alignment solution using either algorithm to a ground-truth solution, generated by transforming 100 randomly sampled frames from the Parkside dataset (an outdoor dataset, see Sect. 4). Error bars depict the standard error, with markers indicating significance for a one-tailed t-test checking whether the difference of MSE was significantly above zero

ments (Parkside, see Sect. 4). The frames were subjected to a random transformation to establish a ground truth correspondence; frames were rotated and then translated along the X, Y, and Z axes independently (sampled from $\mathcal{N}(0, 5°)$ and $\mathcal{N}(0, 0.15\,\text{m})$, respectively), and Gaussian noise was added independently to each point. A uniformly random selection of points was used as a baseline for comparison. Fig. 5 shows the average difference in mean square error between the randomly selected points and those selected via Algorithm 3.1. The experiment was run over increasing amounts of additive Gaussian noise, showing that the algorithm gives a significant improvement in mean squared error (MSE) until the noise obscures the original data (1 m standard deviation).

3.5 Computing transforms

To compute a minimum cost transformation between one frame to the next, a new algorithm called Iterative Closet Point Plus Plane Optimization (IC3PO) has been developed. This algorithm, detailed in Algorithm 3.2, takes as input a set of plane correspondences $\mathbf{P}_{ab}$, two point clouds $\mathcal{Q}_a$ and $\mathcal{Q}_b$, and the previously discussed constraint threshold $\hat{\tau}$.

The algorithm first computes the constraint ellipse for the set of plane correspondences, weighting each plane normal by the number of points that contributed to that plane. If the plane correspondences produce a constraint ellipse that passes the threshold $\hat{\tau}$ in all dimensions, then the lowest cost transform can be computed in closed form by decoupling the rotation and translation estimation, similar to the methods of Pathak et al. (2010). When a lack of detected planes gives insufficient constraints for a closed form solution, more features are added to the system and an iterative solution is found. The plane correspondences $\mathbf{P}_{ab}$ are fixed throughout the iterative point fitting, though the algorithm could be extended to recompute planar correspondences at each iteration depending on time constraints.

3.5.1 Plane constrained estimation

If the constraint ellipse passes threshold in all dimensions, the translation and rotation components of the transformation can be computed in closed form:

Translation estimation The decoupled translation estimation between frames a and b is solved by the following least squares estimation: $\mathbf{Mt} = \mathbf{d}$. $\mathbf{M}$ is a stacked matrix of the N plane normals from frame a, scaled by the inverse variance:

$$\mathbf{M} = \begin{bmatrix} n_{a_1}^T / \left(\sigma_{a_1} + \sigma_{b_1}\right) \\ n_{a_2}^T / \left(\sigma_{a_2} + \sigma_{b_2}\right) \\ \vdots \\ n_{a_N}^T / \left(\sigma_{a_N} + \sigma_{b_N}\right) \end{bmatrix} \tag{9}$$

The vector $\mathbf{d}$ is constructed by stacking the differences in plane offsets, and scaling by the same inverse variance:

$$\mathbf{d} = \begin{bmatrix} \left(d_{a_1} - d_{b_1}\right) / \left(\sigma_{a_1} + \sigma_{b_1}\right) \\ \left(d_{a_2} - d_{b_2}\right) / \left(\sigma_{a_2} + \sigma_{b_2}\right) \\ \vdots \\ \left(d_{a_N} - d_{b_N}\right) / \left(\sigma_{a_N} + \sigma_{b_N}\right) \end{bmatrix} \tag{10}$$

Here, σ_{a_i} and σ_{b_i} are the estimated variances of the first and second frame offsets as found by the plane detector (see also the appendix of Pathak et al. 2010).

The translation can then estimated by use the Moore-Penrose pseudo-inverse of $\mathbf{M}$:

$$\tilde{\mathbf{t}} = \mathbf{M}^{+}\mathbf{d} \tag{11}$$

Note that all translation estimation is performed using only the infinite representations of the planes, rather than considering the overlap of their hulls. Hull estimation was found to be too noisy in general to be of much use to provide good estimates for translation.

Rotation estimation Determining the optimal rotation between a set of corresponding vector observations has been well studied in aerospace engineering for the application of star trackers, and is known as Wahba's problem (Shuster 2006). The quaternion based solution derived by Davenport

(1968), known as the "q-method," is used here. For this, a matrix $\mathbf{B}$ is constructed as the sum of the product of all normal vector pairs:

$$\mathbf{B} = \sum_{i=0}^{N} \frac{n_{a_i} n_{b_i}^T}{\left(\sigma_{a_i} + \sigma_{b_i}\right)} \tag{12}$$

where σ_{a_i} and σ_{b_i} are the decoupled variances of the plane normals found by the plane detector. A matrix $\mathbf{K}$ is then constructed as follows:

$$\mathbf{K} = \begin{bmatrix} B + B^T - \mathbf{I}\,\mathrm{tr}\,(\mathbf{B}) & \mathbf{Z} \\ \mathbf{Z}^T & \mathrm{tr}\,(\mathbf{B}) \end{bmatrix} \tag{13}$$

with

$$\mathbf{Z} = \begin{bmatrix} B_{23} - B_{32} \\ B_{31} - B_{13} \\ B_{12} - B_{21} \end{bmatrix} = \sum_{i=0}^{N} \frac{n_{a_i} \times n_{b_i}}{\left(\sigma_{a_i} + \sigma_{b_i}\right)} \tag{14}$$

The best fit rotation can then be found by computing the eigenvectors of $\mathbf{K}$ through eigendecomposition. The eigenvector corresponding to the maximum eigenvalue is a quaternion representing the best possible rotation. Note that within the aerospace literature, (e.g., Ainscough et al. 2014), this decomposition is typically performed using an iterative algorithm such as QUEST (Shuster and Oh 2012). An excellent treatment of the subject of rotation estimation, with an overview of many alternative methods may be found in Markley and Mortari (2000).

3.5.2 Iterative closet point plus plane optimization

When a lack of detected planes results in insufficient constraints for a closed form solution, more features must be added to the system. These points are selected as per Sect. 3.4 from the input point cloud after first removing all points associated with detected planes as found in Sect. 3.2. Once the points are chosen, a solution inspired by the Generalized Iterative Closest Point algorithm is used, additionally incorporating the planar correspondences. The method iteratively switches between a correspondence finding phase, and a transformation solving phase. After transforming point cloud $\mathcal{Q}_a$ by the current transform estimation, correspondences between the clouds are found using a fast nearest neighbors search optimized for point clouds (Blanco and Rai 2014). These point correspondences as well as the plane correspondences are then used as inputs into a nonlinear optimization problem to be solved by the Broyden–Fletcher–Goldfarb–Shanno (BFGS) quasi-Newton method (Wright and Nocedal 1999) using a Cauchy loss function as implemented in the Google Ceres library (Agarwal et al. 2016). Although point to point alignment can be solved in closed form (e.g., Horn 1987; Umeyama 1991), using a nonlinear optimizer allows the algorithm to behave similarly to GICP in the absense of planes and follows current trends in point alignment which utilize nonlinear optimization (e.g., Segal et al. 2009; Zhang and Singh 2014b). Furthermore, these optimizers can often be more robust and accurate than direct methods (Fitzgibbon 2003).

Algorithm 3.2 The IC3PO algorithm, which combines planar features with a minimal number of point features to compute a rigid body transform between two scans

inputs
- $\mathbf{P}_{ab}$ corresponding planes from frames a and b
- $\mathcal{Q}_a$, $\mathcal{Q}_b$ point clouds from frames a and b
- $\hat{\tau}$ a constraint threshold

local variables
- $\mathcal{C}$ a constraint matrix
- γ a plane correspondence weight
- r_2 the smallest radius of a constraint ellipsoid
- $\tilde{\mathcal{Q}}_a$ points chosen from cloud a
- $\mathbf{T}$ a transformation matrix
- $\mathcal{Q}_{ab}$ a set of point correspondences

1: $\mathcal{C} \leftarrow \mathbf{0_{3\times3}}$
2: **for** $\langle p_a, p_b \rangle$ in $\mathbf{P}_{ab}$ **do**
3: $\quad\gamma \leftarrow (|p_a| + |p_b|)/2$
4: $\quad$add p_a to $\mathcal{C}$ with γ ▷ Eq. 3
5: **end for**
6: Calculate r_2 from $\mathcal{C}$ ▷ Eq. 4
7: **if** $r_2 > \hat{\tau}$ **then**
8: $\quad$**return** closed form $\mathbf{T}$
9: **end if**
10: $\tilde{\mathcal{Q}}_a \leftarrow$ `choosePoints`$(\mathcal{C}, \mathcal{Q}_a)$ ▷ Alg. 3.1
11: $\mathbf{T} \leftarrow \mathbf{I}$
12: **while** not converged **do**
13: $\quad\mathbf{Q}_{ab} \leftarrow$ `findCorrespondences`$(\mathbf{T}\tilde{\mathcal{Q}}_a, \mathcal{Q}_b)$
14: $\quad\mathbf{T} \leftarrow$ `findTransform`$(\mathbf{P}_{ab}, \mathbf{Q}_{ab})$
15: **end while**
16: **return T**

The optimization problem is set up to minimize the following error function:

$$\min_{\mathbf{x}} \frac{1}{2}\left(\rho_\alpha\left(\|f_\alpha(\mathbf{x})\|^2\right) + \rho_\beta\left(\|f_\beta(\mathbf{x})\|^2\right)\right) \tag{15}$$

which describes the total error over two residual blocks: point alignments (α) and plane alignments (β), with loss functions ρ_α and ρ_β. This error is evaluated given a candidate transformation ($\mathbf{x}$), which is represented as a 6D $\mathfrak{se}(3)$ vector:

$$\mathbf{x} = \begin{pmatrix} \mathbf{t} \\ \omega \end{pmatrix} \tag{16}$$

where $\mathbf{t}$ is a 3D translation vector, and ω is a 3D angle-axis rotation vector.

The point alignment block α is as described in the original GICP paper (Segal et al. 2009), and evaluates the point-wise error as:

$$f_\alpha(\mathbf{x}) = \sum_{i=0}^{N} \left(\exp(\mathbf{x})\mathbf{q}_{ai} - \mathbf{q}_{bi}\right)^T \mathbf{M}_i \left(\exp(\mathbf{x})\mathbf{q}_{ai} - \mathbf{q}_{bi}\right) \tag{17}$$

where $\mathbf{q}_{ai}$ and $\mathbf{q}_{bi}$ are the ith corresponding points from frames a and b, and $\exp(\mathbf{x})$ is the $SE(3)$ matrix representation of the $\mathfrak{se}(3)$ rigid body transformation vector $\mathbf{x}$. $\mathbf{M}$ is the inverse of the frame aligned sum of the covariance matrices of the neighborhoods around the two points, which turns Eq. (17) into a Mahalanobis-like distance metric:

$$\mathbf{M}_i = \left(\mathbf{R}\mathbf{C}_{ai}\mathbf{R}^T + \mathbf{C}_{bi}\right)^{-1} \tag{18}$$

where $\mathbf{C}_{ai}$ and $\mathbf{C}_{bi}$ are the neighborhood covariance matrices of the two points, and $\mathbf{R}$ is the rotation block of $\mathbf{T}$ from the previous iteration of Algorithm 3.2. The intuition behind this distance metric is that if the two point neighborhoods are planar, then Eq. (17) will only penalize errors normal to that plane, but will allow the points to slide relative to each other in the plane (Segal et al. 2009). In practice this minimizes the effect of incorrect correspondences as well as the fact that, due to various sources of sensor noise, two corresponding points are rarely sampled at exactly the same physical location.

The plane residual block $f_\beta(\mathbf{x})$ produces a vector describing the alignment error over each element of the Hessian normal form for every plane correspondence:

$$f_\beta(\mathbf{x}) = \begin{pmatrix} \mathbf{d}_0 \\ \mathbf{d}_1 \\ \vdots \\ \mathbf{d}_N \end{pmatrix} \tag{19}$$

where each sub-block $\mathbf{d}_i$ is then defined as follows:

$$\mathbf{d}_i = \begin{pmatrix} \mathbf{R}^T\mathbf{n}_{bi} - \mathbf{n}_{ai} \\ \mathbf{t}^T\mathbf{n}_{bi} + o_{bi} - o_{ai} \end{pmatrix} \tag{20}$$

$\mathbf{R}$ and $\mathbf{t}$ are the rotation matrix and translation vector extracted from the exponential map of $\mathbf{x}$:

$$\exp(\mathbf{x}) = \begin{bmatrix} \mathbf{R} & \mathbf{t} \\ \mathbf{0} & 1 \end{bmatrix} \tag{21}$$

The Jacobian for each of the sub blocks shown in Eq. 20 is as follows:

$$\mathcal{J}_i = \begin{bmatrix} \frac{\partial \mathbf{n}}{\partial \mathbf{t}} & \frac{\partial \mathbf{n}}{\partial \mathbf{!}} \\ \frac{\partial o}{\partial \mathbf{t}} & \frac{\partial o}{\partial \mathbf{!}} \end{bmatrix} = \begin{bmatrix} \mathbf{0}_{3\times 3} & \left[\mathbf{R}^T\mathbf{n}_{bi}\right]_\times \\ \left(\mathbf{R}^T\mathbf{n}_{bi}\right)^T & \mathbf{0}_{1\times 3} \end{bmatrix} \tag{22}$$

with the operator $[\cdot]_\times$ producing the cross-product matrix given a vector:

$$[a]_\times \overset{def}{=} \begin{bmatrix} 0 & -a_3 & a_2 \\ a_3 & 0 & -a_1 \\ -a_2 & a_1 & 0 \end{bmatrix} \tag{23}$$

3.6 SLAM

By chaining the transformation estimates found in the previous subsections together, the trajectory of the sensor through the world can be roughly estimated. Unfortunately, due to noise in the sensor readings this approach will eventually lead to unacceptable drift in the pose estimates. To combat this, a SLAM solution is implemented which uses the detected planar features as landmarks. The solution is constructed as a factor graph optimization problem using the iSAM2 framework of Kaess et al. (2011).

3.6.1 Graph formulation

The factor graph, as seen in Fig. 6, consists of two types of variable nodes, representing (1) the pose at each time step, and (2) the plane equation of each planar landmark. Poses are

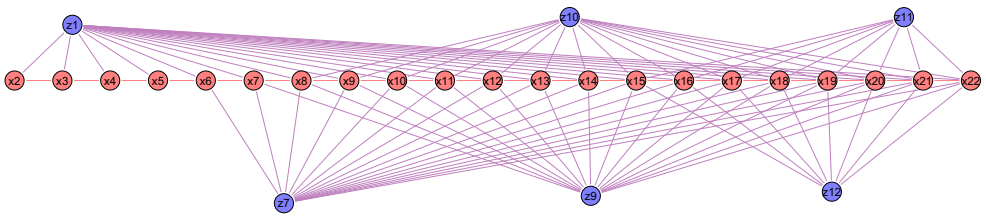


Fig. 6 Example graph from the IC3PO+SLAM formulation. Red variable nodes (x_i) represent estimated poses, while blue variable nodes (z_i) represent detected planar landmarks. Measurement factors (not shown) exist along each edge, either representing a computed odometry measurement or a plane measurement. Note that some landmark nodes have been removed for clarity of the figure (Color figure online)

represented in the system as $\mathfrak{se}(3)$ vectors, while planes are represented in their Hessian normal form as $[n_x, n_y, n_z, \rho]$. Connections between these nodes are made by two types of factors (not shown explicitly in Fig. 6): odometry factors that describe the transformation from one sensor location to the next, and 2) "observation" factors that describe the observation of a single planar feature from a pose. Although in general both types of factors arise at least in part from plane measurements and hence do not always have independent noise (which is an assumption of iSAM2), both types are needed to allow the developed framework to handle environments in which available planes (if any) do not provide sufficient constraints; the results in Sect. 4 demonstrate excellent performance in practice.

Odometry factors provide the following error function, which takes as input two poses (x_a and x_b), and computes the error given an odometry measurement (u):

$$e(x_a, x_b|u) = u - x_a^{-1} x_b \tag{24}$$

Note that all coordinates are represented as 6D $\mathfrak{se}(3)$ vectors, and thus the subtraction operator represents the distance between u and $x_a^{-1} x_b$ on the Lie manifold positioned around u.

Observation factors provide an error function which takes as input a pose value (x) and a plane value (p), and describes the difference between the measurement that would have been taken at x of p to the actual measurement z:

$$e(x, p|z) = z - x^{-1} p \tag{25}$$

Note here that both x and p are in a global coordinate system, thus the operation $x^{-1} p$ generates a hypothetical plane measurement in the local coordinate system to be compared with z. Planes are optimized in their Hessian normal form, though recent work has shown that a more minimal representation can be used to improve run-time performance and allow a wider range of optimizers to be used (Kaess 2015).

3.6.2 Graph construction

Once the variables and factors have been defined for the SLAM problem, it is a matter of bookkeeping to implement a full SLAM algorithm. Taking as input a sequence of transformation estimates and plane correspondences, the IC3PO SLAM (IC3PO + SLAM) algorithm iterates through the sequence to generate a globally consistent factor graph representing both the scene and the sensor's path through it. The design of iSAM2 allows each update iteration to take place online, resulting in an algorithm which is suitable for use on a robot.

At each time step, the algorithm inserts a new pose node into the graph and links it to the previous one by a factor created using the estimated transform of Sect. 3.5. Then, a new observation factor is inserted between this pose node and a landmark node for each plane correspondence. If a correspondence is between two planes that are not known by the map, a new landmark node is created for the plane before the factor is inserted, conditioned on the correspondence being observed for a sufficient number of consecutive iterations. Otherwise, if the correspondence is known to the map, the factor is made between the new pose and the known landmark. Correspondence calculation is detailed in the next section.

3.6.3 Plane correspondence

Correspondences to planes in the map is stricter than the algorithm used for correspondence during frame to frame registration (Sect. 3.2).

For a pair of planes $\boldsymbol{p_a}$ and $\boldsymbol{p_b}$, first check if they satisfy both constraints required of planes for frame to frame registration (Eqs. 1 and 2).

If these basic constraints are met, calculate their mean plane $\bar{\boldsymbol{p}}_{\boldsymbol{ab}}$ by averaging the plane normals and offsets. Let $H_{\boldsymbol{p_a}}$ and $H_{\boldsymbol{p_b}}$ denote the convex hulls created by projecting the points belonging to $\boldsymbol{p_a}$ and $\boldsymbol{p_b}$, respectively, onto their mean plane $\bar{\boldsymbol{p}}_{\boldsymbol{ab}}$. The intersection and union of these hulls can then be calculated: $I_{ab} = H_{\boldsymbol{p_a}} \cap H_{\boldsymbol{p_b}}$ and $U_{ab} = H_{\boldsymbol{p_a}} \cup H_{\boldsymbol{p_b}}$.

The two planes are considered to be in correspondence if either the area of their intersection over union exceeds a threshold:

$$\text{area}(I_{ab})/\text{area}(U_{ab}) > t_{i\_u} \tag{26}$$

or if the ratio of their intersection over their minimum area exceeds a threshold, along with a stricter test for Eqs. (1 and 2):

$$\text{area}(I_{ab})/\min(H_{\boldsymbol{p_a}}, H_{\boldsymbol{p_b}}) > t_{i\_m} \tag{27}$$

where t_{i_u} and t_{i_m} are thresholds in square meters.

3.6.4 Loop closure and plane merging

Once all factors and nodes are inserted into the graph, the iSAM2 optimizer finds the least cost solution for all nodes given the factors. Following this, a loop closure and merging step is applied to the map, where any similar planes are merged together. To accomplish this, a graph is created where nodes represent landmark planes and edges exist if the planes are considered similar enough. To determine similarity, the correspondence test of Sect. 3.6.3 is used. The creation of this graph has quadratic time complexity in the number of planes. In practice the number of planes, even in larger environments, is generally small enough that this step does not

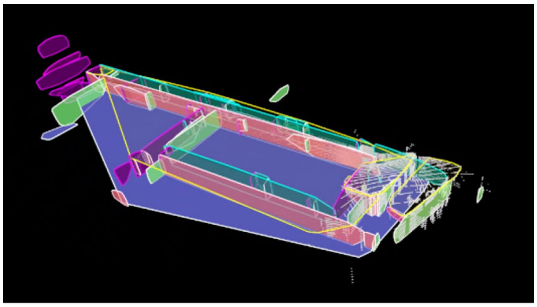
add a detrimental amount of time to the SLAM algorithm (see Sect. 4, Fig. 16 for a timing example).

The graph is then collapsed using its connected components, with each component being merged into a single planar landmark. The hulls of each plane are combined, and the factors associated assigned to a single landmark node, with the merged nodes being removed. This loop closure operation both detects actual loop closures as well as merges similar plane insertions.

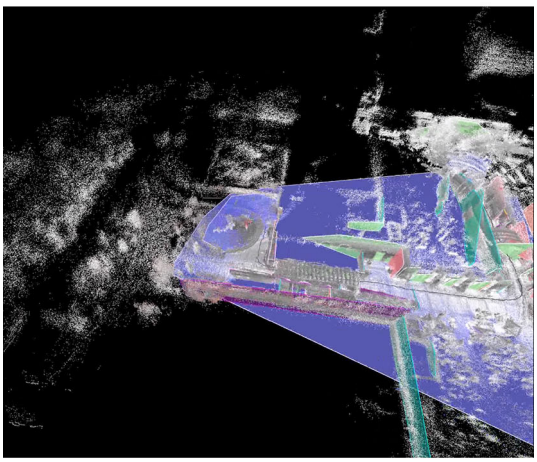
4 Experimental evaluation

To test the effectiveness of the system, a Velodyne HDL-32E sensor with 32 rotating laser/receiver pairs was mounted on a backpack and walked through four different environments. Due to the plane finding algorithm used, it is essential that the data in the resultant pointcloud can be traced back to the individual laser scanline it originated from. The algorithm was implemented as a single threaded C++ application using the iLab Neuromorphic Robotics Toolkit (Itti et al. 2016). The four datasets were as follows, with raw data, source code (and parameters), video, and reconstructions available on the authors' website (Grant et al. 2016):

1. Parkside: an 80 m × 45 m "L" shaped path in an outdoor courtyard of a residential building.
2. Tutor: a 65 m × 40 m loop around the exterior of a building in a busy campus center.
3. Alley: a 40 m forward and back path through an outdoor alleyway.
4. RTH: a 10 m × 40 m loop through an indoor office hallway.



(a)



(b)

Fig. 7 Examples of maps created using the IC3PO + SLAM algorithm. **a** Final planar map for the RTH dataset created with the IC3PO + SLAM algorithm. The convex hulls of each plane are shown, colored according to their plane normal. Planes floating away from the main corridors are adjacent buildings visible through windows. **b** A snapshot of the Tutor dataset in progress, showing the transition into the less structured part of the environment, which consists of an open pathway bounded by trees. Both planes as well as the integrated point cloud are shown

These datasets capture environments in which a variable number of planes are present. The RTH dataset, which is indoors, has many consistent planar features in all dimensions. The Alley dataset, which is outdoors, lacks many planes perpendicular to the direction of motion. The Parkside dataset consists of a building next to a large open courtyard, which offers no planar constraints. The Tutor dataset is the most mixed environment, consisting of passageways between buildings, large open courtyards with many pedestrians present, and open areas with trees and grass. Figure 7 shows example reconstructions using the IC3PO+SLAM algorithm for two of these datasets.

For each dataset, the sensor was walked in a closed loop such that the final position was equal to the starting position. The IC3PO algorithm was tested on both its frame to frame performance as well as its performance in a full SLAM configuration (IC3PO + SLAM). For the frame to frame testing, it was compared to using only planar correspondences (Plane) and to using only point features (GICP). The Plane and GICP algorithms were implemented by turning off the appropriate features of IC3PO (e.g., for GICP, no planes were used and the constraint matrix was constructed such that all points were used). For the SLAM configuration, it was compared against the Omnimapper implementation of Generalized ICP SLAM (Trevor et al. 2014), which also uses afactor graph formulation, as well as against the ROS implementation of the LOAM algorithm (Zhang and Singh 2014b). It should be noted that although LOAM does maintain a global map of the environment, it does not perform global optimization over that map should a loop closure be detected; rather it uses the map to perform continuous local optimization over several pose estimates. LOAM therefore lies somwhere in between the pure frame to frame algorithms and the SLAM algorithms with global optimization.

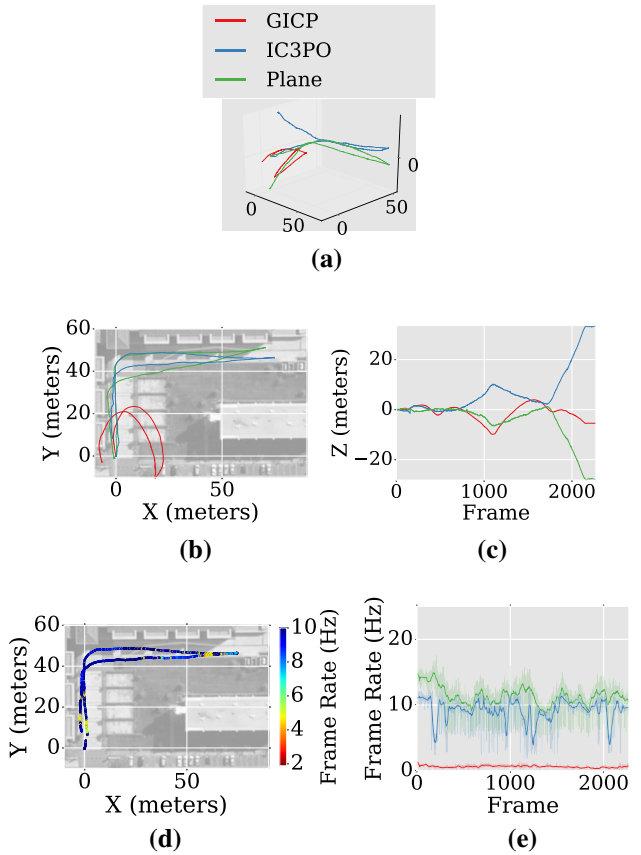


Fig. 8 Frame to frame results for the Parkside Dataset: an 80 m × 45 m "L" shaped path in an outdoor residential courtyard. Legend applies to all subfigures. **a** 3D view (comparison), all units meters. **b** Top view (comparison). **c** Z position (comparison). **d** IC3PO timing. **e** Frame rate (comparison)

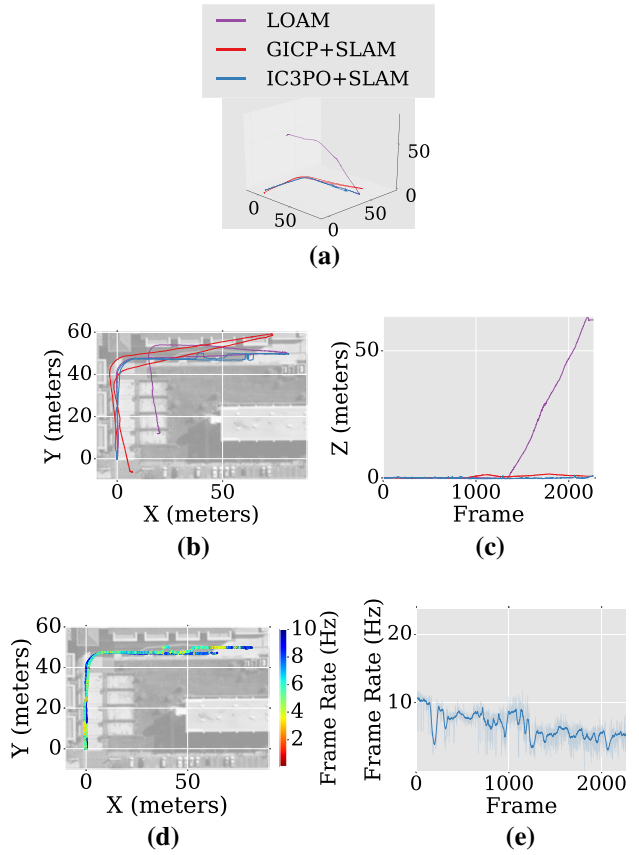


Fig. 9 SLAM and LOAM results for the Parkside Dataset: an 80 m × 45 m "L" shaped path in an outdoor residential courtyard. Legend applies to all subfigures. **a** 3D view (comparison), all units meters. **b** Top view (comparison). **c** Z position (comparison). **d** IC3PO+SLAM timing. **e** IC3PO + SLAM frame rate

Figures 8, 9, 10, 11, 12, 13, 14 and 15 show the results of the testing. Each figure contains five subfigures, showing (a) a 3D isometric view of the paths of the tested algorithms, (b) a top-down view of the paths overlaid onto a grayscale satellite image for outdoor datasets, (c) the Z position of the estimated pose over time (because all paths were approximately planar, large deviations from $Z = 0$ signal alignment errors), (d) the path of either IC3PO or IC3PO + SLAM, colored by frame rate, and (e) a plot of each algorithm's frame rate over time. Solid lines depict smoothed frame rate, with transparent lines used for actual frame rates. For (e), only algorithms with variable frame rates are displayed. The LOAM and GICP + SLAM algorithms are implemented in such a way that they operate at a fixed frame rate, dropping any frames they cannot process. The algorithms were run at the fastest frame rate for which there was no appreciable benefit to running slower, which was 1 Hz for GICP + SLAM and 5 Hz for LOAM. The original LOAM paper runs the algorithm at 10 Hz, but this caused the non-blocking algorithm to drop too many frames resulting in poor performance. GICP + SLAM was attempted at 5 Hz, but suffered from the same issues, with 1 Hz being optimal.

Table 1 shows the distance between the start and end positions reported by all algorithms for each dataset. Because each dataset was a loop, numbers closer to zero indicate better performance. Table 2 provides detailed timing information for each algorithm. Looking at the timing for IC3PO and IC3PO+SLAM versus Planes, it is evident that finding planes provides a rough lower bound on the runtime of either algorithm. Optimizing the plane finding and registration is thus an obvious target for future optimization work. The results of these tables are best viewed in conjunction with the specific dataset result from Figs. 8, 9, 10, 11, 12, 13, 14 and 15 to provide context.

In all four datasets, the IC3PO algorithm was able to provide an accurate path reconstruction and to close the loop between the starting and ending poses when using SLAM. Competing algorithms were able to provide good reconstructions for portions of the paths, but often made catastrophic errors from which they were unable to recover. Some of these

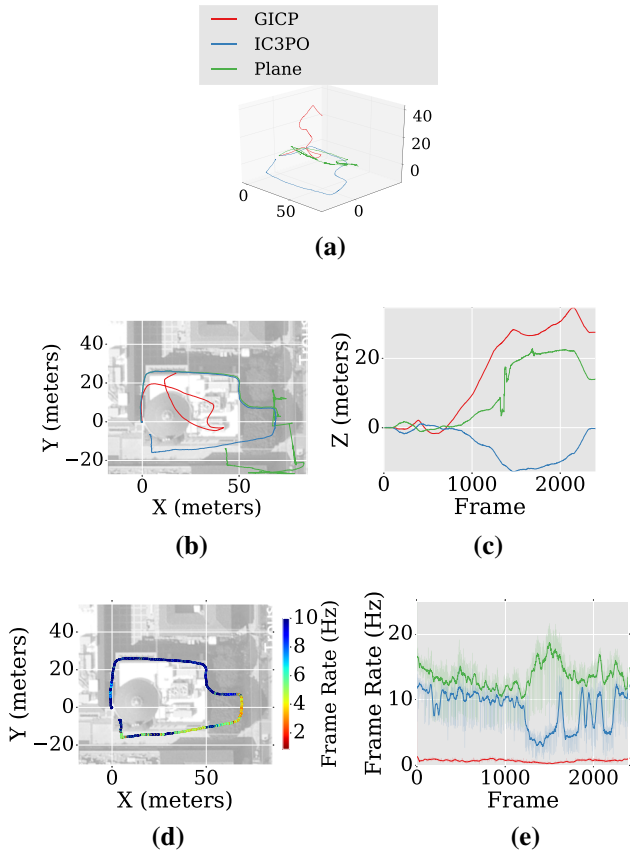


Fig. 10 Frame to frame results for the Tutor Dataset: a 65 m × 40 m loop around the exterior of a campus center building. Legend applies to all subfigures. **a** 3D view (comparison), all units meters. **b** Top view (comparison). **c** Z position (comparison). **d** IC3PO timing. **e** Frame rate (comparison)

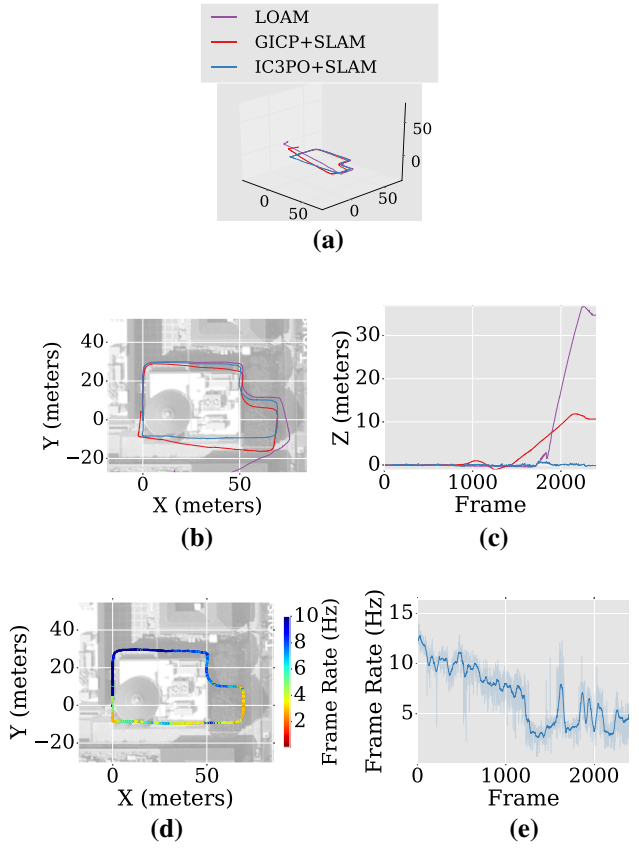


Fig. 11 SLAM and LOAM results for the Tutor Dataset: a 65 m × 40 m loop around the exterior of a campus center building. Legend applies to all subfigures. **a** 3D view (comparison), all units meters. **b** Top view (comparison). **c** Z position (comparison). **d** IC3PO + SLAM timing. **e** IC3PO + SLAM frame rate

errors may be due to the large swaying motion of the sensor induced by the walking gait of the human carrying it combined with the uneven sparsity of the point clouds. The algorithm's ability to handle such situations highlight the robustness of using planes for odometry and SLAM. However, the tested algorithms, including IC3PO+SLAM, cannot meaningfully recover from a drastic mistake in estimated trajectory, as each algorithm relies on matching its current location to a global map to perform loop closure or otherwise optimize the pose. While GICP+SLAM and LOAM map localized features in their current pose to localized features in a subset of their global map, IC3PO+SLAM uses an infinite plane based formulation that allows the algorithm to always leverage the full structure of the global plane map during loop closure. A practical result of this is that IC3PO+SLAM tends to perform constant refinments to the estimated pose instead of the large instant rectification typically associated with loop closure.

When operating without the SLAM component, the IC3PO algorithm clearly shows benefits over either a planes only or purely GICP solution, being the most accurate of the three. Although using only planes can be very fast, environments lacking sufficient constraints (see Figs. 8, 10) cause severe errors to accumulate. Integrating point features prevents this from being a problem, providing results that generally show little drift over time.

It is useful to compare points of failure in GICP+SLAM and LOAM to not only IC3PO+SLAM, but against its operation without the SLAM component. For example, in situations where LOAM veered dramatically off the actual trajectory, the IC3PO algorithm rarely made comparible mistakes. The LOAM algorithm relies on the extraction of structured edge and planar features so it is perhaps not unsurprising that the datasets where it performs most poorly, Parkside and Tutor (Figs. 9, 11), are those with large open areas that can lack strong planar constraints. Indeed similar mistakes, often more severe, can be seen in the Planes algorithm which is completely dependent on structure found in the point cloud. Looking at the points of failure for LOAM more closely (video walktrhoughs provided on the authors' website, Grant et al. 2016), the Tutor dataset drifts off course during the transition to the open pathway flanked by trees

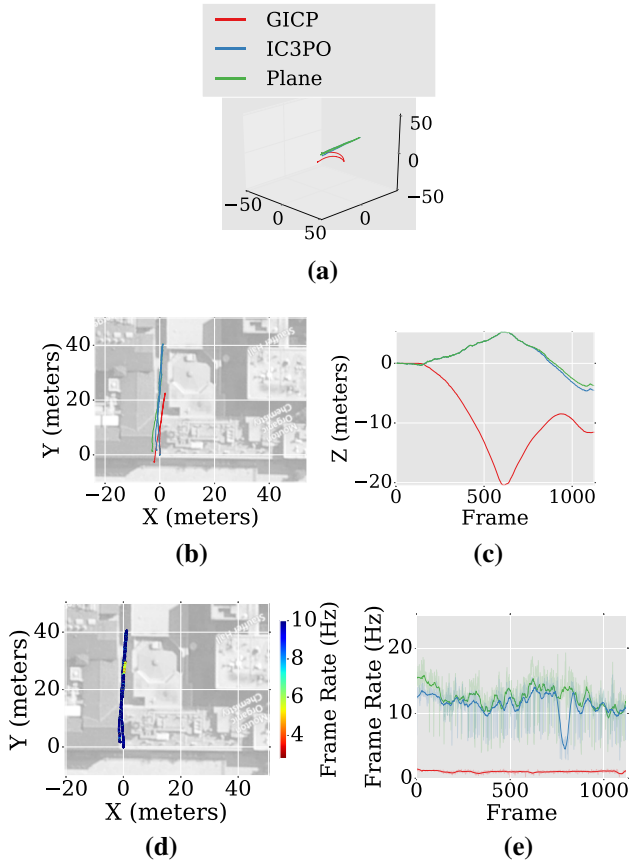


Fig. 12 Frame to frame results for the Alley Dataset: a 40 m forward and back path through an outdoor alleyway. Legend applies to all subfigures. **a** 3D view (comparison), all units meters. **b** Top view (comparison). **c** Z position (comparison). **d** IC3PO timing. **e** Frame rate (comparison)

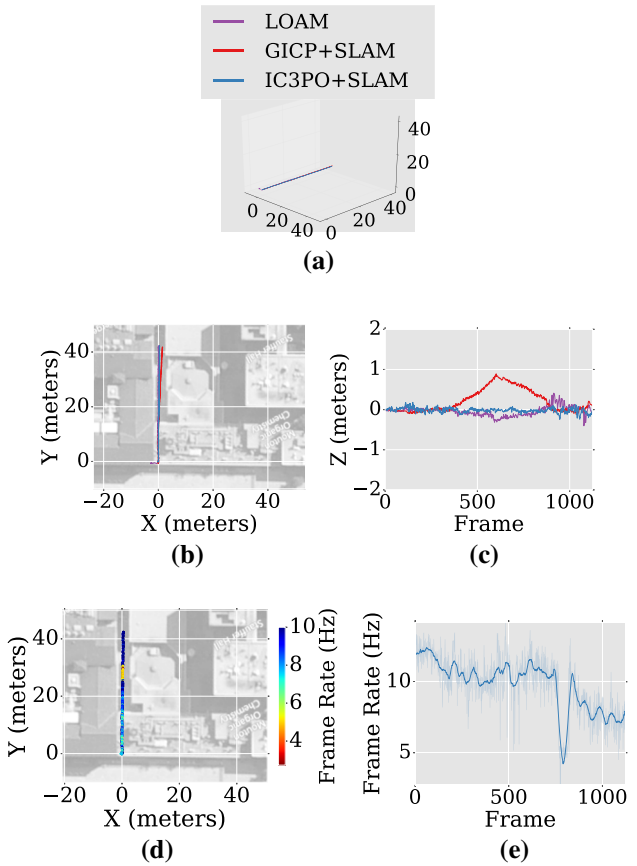


Fig. 13 SLAM and LOAM results for the Alley Dataset: a 40 m forward and back path through an outdoor alleyway. Legend applies to all subfigures. **a** 3D view (comparison), all units meters. **b** Top view (comparison). **c** Z position (comparison). **d** IC3PO + SLAM timing. **e** IC3PO + SLAM frame rate

and grass. Even in the RTH dataset, which is an indoor office environment, mistakes in trajectory come at a point where the sensor passes by windows which causes much of the point data to come from distant, less structured objects. These results, in conjunction with the Planes results, show that relying purely on structure found in the point cloud is not robust to all environments. When structured features are readily available, they perform remarkably well, but their absence can cause catostrophic trajectory errors. The results also demonstrate that relying purely on points misses out on valuable constraints that highly structured components such as large planar surfaces and edges provide. This is most evident looking at the Z trajectories of GICP + SLAM, which occasionally makes large errors that a ground plane can prevent.

One interesting feature of the IC3PO algorithm is its variable runtime, which depends upon the complexity of the surrounding environment. When the environment is well ordered and planes are numerous, the algorithm is able to run at a very high frame rate, as much as 14x faster than GICP. However, when insufficient planes are present, the algorithm smoothly lowers the frame rate without sacrificing accuracy as it begins to integrate points, while still maintaining a frame rate that exceeds GICP. The lower right corner of the Tutor dataset shown in Fig. 10d demonstrates this feature very well. In this part of the path, there were very few visible planar surfaces in the north and south directions causing a drop in frame rate as more point features needed to be processed. Figure 16 shows a detailed timing plot for the Tutor dataset which breaks down the timing of each frame by the stages of the algorithm. As can be seen in the figure, the nearest neighbors selection from IC3PO causes a large spike in computation time near frame 1500 where the environment becomes under constrained. Should strict time budgets be a concern, the implementation could easily be adjusted to drop frames in a similar fashion to the GICP+SLAM and LOAM competitors. The current implementation could also be extended for parallelism, as it is currently single threaded.

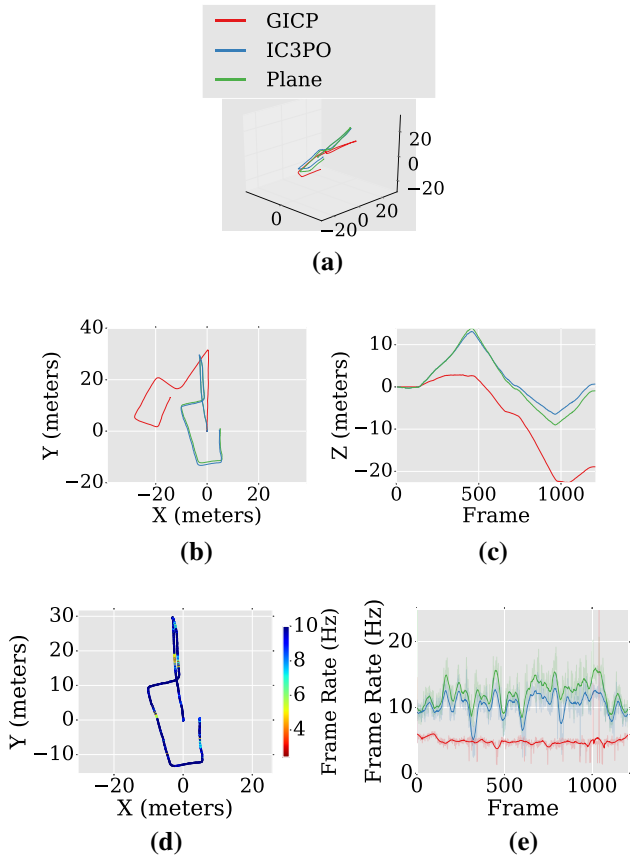


Fig. 14 Frame to frame results for the RTH Dataset: a 10 m × 40 m loop through an indoor office hallway. Legend applies to all subfigures. **a** 3D view (comparison), all units meters. **b** Top view (comparison). **c** Z position (comparison). **d** IC3PO timing. **e** Frame rate (comparison)

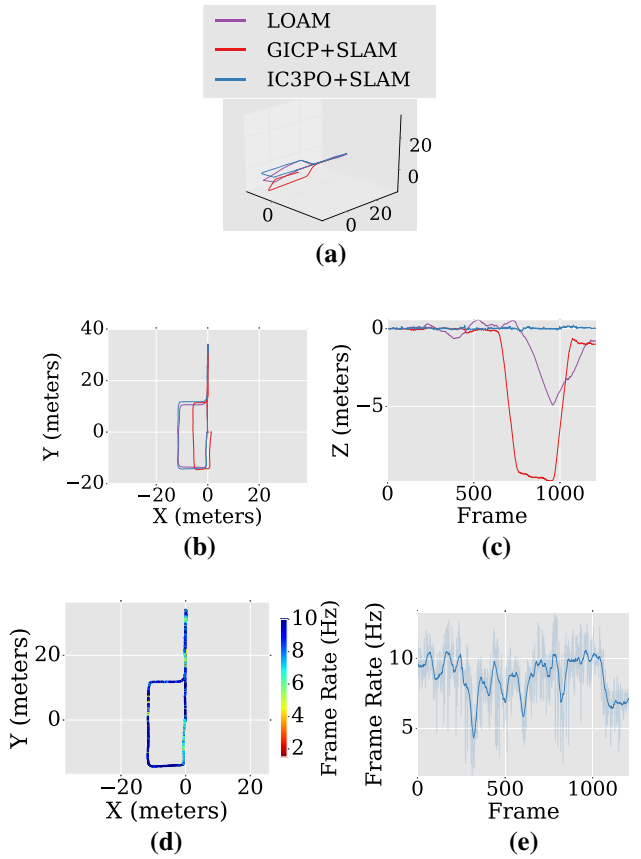


Fig. 15 SLAM and LOAM results for the RTH Dataset: a 10 m × 40 m loop through an indoor office hallway. Legend applies to all subfigures. **a** 3D view (comparison), all units meters. **b** Top view (comparison). **c** Z position (comparison). **d** IC3PO + SLAM timing. **e** IC3PO + SLAM frame rate

5 Conclusion

In this work, a new solution to the SLAM problem for scanning LiDAR sensors was developed. The solution has a variable runtime, allowing it to exploit the structure of man made planar environments when available. When such structure is unavailable, the algorithm smoothly transitions to a more computationally intensive mode without sacrificing accuracy. The system has been shown to perform well on datasets taken in a variety of indoor and outdoor environments, beating state of the art competitors. As scanning LiDAR sensors become more inexpensive and readily available, algorithms such as this will allow mobile robots to traverse human environments with the greatest possible speed. A benefit of the algorithm is that it produces plane detections as a by-product. These planes can be used by higher-level processes for point cloud compression and semantic reasoning. While the algorithm was implemented and tested as a single threaded application, many of its components are easily parallelizeable.

Future work will focus on improving the speed of the algorithm through multithreading and refining the SLAM formulation. For example, taking inspiration from LOAM and incorporating vertical landmarks into both the frame to frame registration as well as the SLAM map could greatly help in environments that lack sufficient planar constraint, but still contain stable vertical features such as traffic signs or street lamps. Such features would likely reduce the reliance on point features and provide both a computational speed up as well as stable landmarks. Additionally, the optimization of transformations between point and plane correspondences should be explored further. A closed form solution may provide superior performance, and the effect of different loss functions should be investigated.

Table 1 Distance between start and end positions using all tested methods for each dataset in meters

	Planes	IC3PO	IC3PO + SLAM	GICP	GICP + SLAM	LOAM	# of scans
Tutor	47.17	7.78	0.14	41.06	11.43	46.66	2372
Parkside	28.05	34.63	0.95	8.99	9.43	66.11	2273
Alley	4.81	4.99	0.21	12.00	0.44	2.69	1124
RTH	5.24	4.82	0.23	27.11	1.68	0.85	1212

Because each dataset consists of a loop, numbers closer to zero generally indicate better performance. An exception to this is the GICP result for the Parkside dataset, which does not accurately reflect the failure of the registration that can be seen in Fig. 8b. These results are best considered in conjunction with the individual results of Figs. 8, 9, 10, 11, 12, 13, 14 and 15

Table 2 Total runtime, in seconds, followed by average framerate, in hertz, for all tested methods for each dataset

	Planes	IC3PO	IC3PO + SLAM	GICP	GICP + SLAM	LOAM	Real time
Tutor	187.23 (12.84)	359.26 (6.69)	479.25 (5.02)	3947.54 (0.61)	2372 (1.00)	474.40 (5.00)	237.2 (10.0)
Parkside	209.20 (10.86)	274.49 (8.28)	410.27 (5.54)	4191.03 (0.54)	2273 (1.00)	454.60 (5.00)	227.3 (10.0)
Alley	96.70 (11.62)	106.36 (10.57)	121.66 (9.24)	1154.58 (0.97)	1124 (1.00)	224.80 (5.00)	112.4 (10.0)
RTH	103.72 (11.69)	123.86 (9.79)	150.91 (8.03)	259.70 (4.67)	1212 (1.00)	242.40 (5.00)	121.2 (10.0)

Note that the runtimes for GICP + SLAM and LOAM were fixed to framerates for which there was no appreciable benefit for running any slower, as discussed in Sect. 4. No restrictions were placed on the other algorithms, which took as long as necessary to analyze every frame. These results are best considered in conjunction with the individual results of Figs. 8, 9, 10, 11, 12, 13, 14 and 15

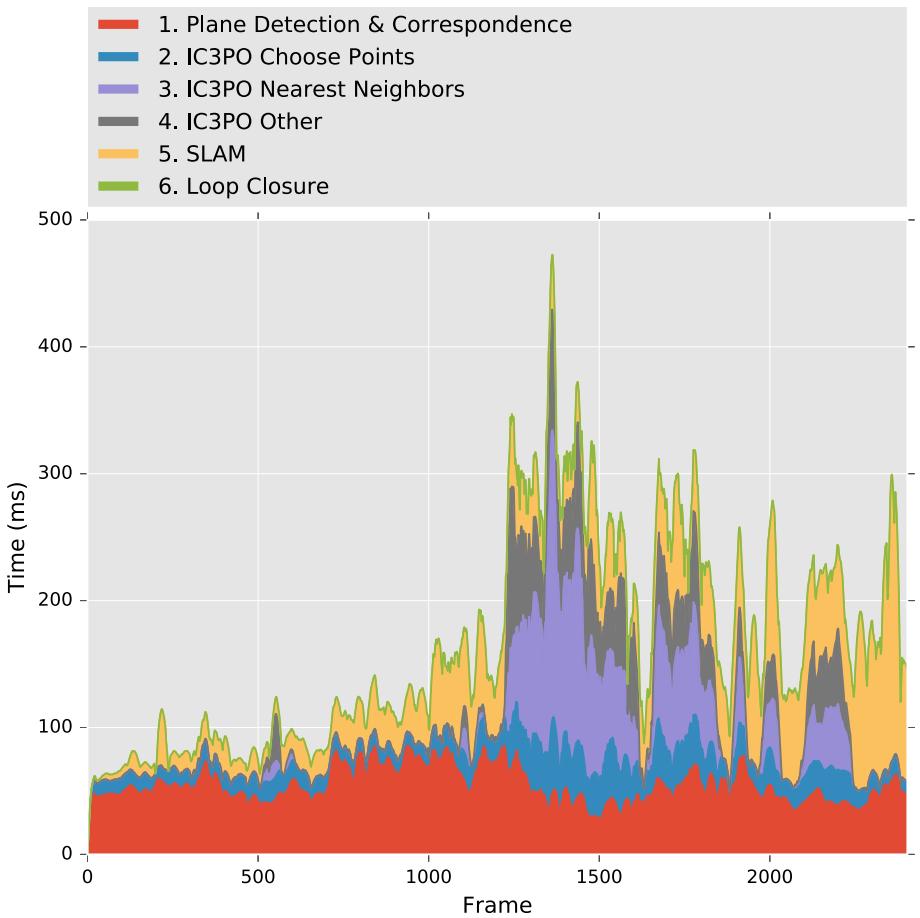


Fig. 16 Detailed timing of Tutor dataset showing the time required for each state in the IC3PO+SLAM system. The y axis shows the total time for each frame, broken down into constituent steps of the algorithm. When planes provide sufficient constraint (e.g., frames 0 to about 1250), the runtime for the algorithm can exceed the data rate of the sensor (10 Hz). When points must be integrated into the frame to frame registration (e.g., around frame 1400), the algorithm adapts but still maintains a relatively high frame rate

References

Agarwal, S., Mierle, K., & et al. (2016). *Ceres solver*. http://ceres-solver.org.

Ainscough, T., Zanetti, R., Christian, J., & Spanos, P. D. (2014). Q-method extended kalman filter. *Journal of Guidance, Control, and Dynamics*, *38*(4), 752–760.

Badino, H., Huber, D., Park, Y., & Kanade, T. (2011). Fast and accurate computation of surface normals from range images. In: *2011 IEEE international conference on robotics and automation (ICRA)* (pp. 3084–3091). IEEE.

Behringer, R., Travis, W., Daily, R., Bevly, D., Kubinger, W., Herzner, W., et al. (2005). Rascal-an autonomous ground vehicle for desert driving in the darpa grand challenge 2005. In: *Intelligent Transportation Systems, 2005. Proceedings* (pp. 644–649). IEEE.

Blanco, J. L., & Rai, P. K. (2014). *nanoflann: A C++ header-only fork of FLANN, a library for nearest neighbor (NN) wih kd-trees*. https://github.com/jlblancoc/nanoflann.

Borrmann, D., Elseberg, J., Lingemann, K., & Nüchter, A. (2011). The 3d hough transform for plane detection in point clouds: A review and a new accumulator design. *3D Research*, *2*(2), 1–13.

Ceriani, S., Sanchez, C., Taddei, P., Wolfart, E., & Sequeira, V. (2015). Pose interpolation slam for large maps using moving 3d sensors. In: *2015 IEEE/RSJ international conference on intelligent robots and systems (IROS)* (pp. 750–757). https://doi.org/10.1109/IROS.2015.7353456

Choi, J., Lee, J., Kim, D., Soprani, G., Cerri, P., Broggi, A., et al. (2012). Environment-detection-and-mapping algorithm for autonomous driving in rural or off-road environment. *IEEE Transactions on Intelligent Transportation Systems*, *13*(2), 974–982.

Davenport, P. B. (1968). *A vector approach to the algebra of rotations with applications*. National Aeronautics and Space Administration

Endres, F., Hess, J., Engelhard, N., Sturm, J., Cremers, D., & Burgard, W. (2012). An evaluation of the rgb-d slam system. In: *2012 IEEE International Conference on Robotics and Automation (ICRA)* (pp. 1691–1696). IEEE.

Fitzgibbon, A. W. (2003). Robust registration of 2d and 3d point sets. *Image and Vision Computing*, *21*(13), 1145–1153.

Grant, W. S., Voorhies, R. C., & Itti, L. (2013). Finding planes in lidar point clouds for real-time registration. In: *2013 IEEE/RSJ international conference on intelligent robots and systems (IROS)* (pp. 4347–4354). IEEE.

Grant, W. S., Voorhies, R. C., & Itti, L. (2016). *Ic3po data and source code*. ilab.usc.edu/ic3po/. Accessed 3 February 2016.

Grisetti, G., Stachniss, C., & Burgard, W. (2005). Improving grid-based slam with rao-blackwellized particle filters by adaptive proposals and selective resampling. In: *Proceedings of the 2005 IEEE international conference on robotics and automation, 2005. ICRA 2005* (pp. 2432–2437). IEEE.

Grisetti, G., Stachniss, C., & Burgard, W. (2007). Improved techniques for grid mapping with rao-blackwellized particle filters. *IEEE Transactions on Robotics*, *23*(1), 34–46.

Hahnel, D., Burgard, W., Fox, D., & Thrun, S. (2003) An efficient fastslam algorithm for generating maps of large-scale cyclic environments from raw laser range measurements. In: *2003 IEEE/RSJ International Conference on Intelligent Robots and Systems, 2003. (IROS 2003). Proceedings* (Vol. 1, pp. 206–211). IEEE.

Henry, P., Krainin, M., Herbst, E., Ren, X., & Fox, D. (2012). Rgb-d mapping: Using kinect-style depth cameras for dense 3d modeling of indoor environments. *The International Journal of Robotics Research*, *31*(5), 647–663.

Horn, B. K. (1987). Closed-form solution of absolute orientation using unit quaternions. *JOSA A*, *4*(4), 629–642.

Itti, L., Voorhies, R. C., Grant, W. S., Parks, D., & Berg, D. (2016). *Neuromorphic robotics toolkit*. nrtkit.org . Accessed 3 February 2016

Kaess, M. (2015). Simultaneous localization and mapping with infinite planes. In: *2015 IEEE International Conference on Robotics and Automation (ICRA)* (pp. 4605–4611). IEEE.

Kaess, M., Johannsson, H., Roberts, R., Ila, V., Leonard, J.J., & Dellaert, F. (2011). isam2: Incremental smoothing and mapping using the bayes tree. *The International Journal of Robotics Research*, 0278364911430419.

Markley, F. L., & Mortari, D. (2000). Quaternion attitude estimation using vector observations. *Journal of the Astronautical Sciences*, *48*(2), 359–380.

Moosmann, F., & Stiller, C. (2011). Velodyne slam. In: *2011 IEEE Intelligent Vehicles Symposium (IV)* (pp. 393–398). IEEE.

Newcombe, R. A., Davison, A. J., Izadi, S., Kohli, P., Hilliges, O., Shotton, J., et al. (2011). Kinectfusion: Real-time dense surface mapping and tracking. In: *2011 10th IEEE international symposium on Mixed and augmented reality (ISMAR)* (pp. 127–136). IEEE.

Pathak, K., Birk, A., Vaskevicius, N., Pfingsthorn, M., Schwertfeger, S., & Poppinga, J. (2010a). Online three-dimensional slam by registration of large planar surface segments and closed-form pose-graph relaxation. *Journal of Field Robotics*, *27*(1), 52–84.

Pathak, K., Birk, A., Vaskevicius, N., & Poppinga, J. (2010b). Fast registration based on noisy planes with unknown correspondences for 3-d mapping. *IEEE Transactions on Robotics*, *26*(3), 424–441.

Salas-Moreno, R. F., Glocken, B., Kelly, P. H., & Davison, A. J. (2014). Dense planar slam. In: *2014 IEEE international symposium on mixed and augmented reality (ISMAR)* (pp. 157–164). IEEE.

Segal, A., Haehnel, D., & Thrun, S. (2009). Generalized-icp. In: *Robotics: Science and Systems*, Vol. 2.

Shuster, M. D. (2006). The generalized Wahba problem. *The Journal of the Astronautical Sciences*, *54*(2), 245–259.

Shuster, M. D., & Oh, S. (2012). Three-axis attitude determination from vector observations. *Journal of Guidance, Control, and Dynamics*.

Trevor, A. J., Rogers, J. G., & Christensen, H. I. (2014). Omnimapper: A modular multimodal mapping framework. In: *2014 IEEE International Conference on Robotics and Automation (ICRA)* (pp. 1983–1990). IEEE.

Umeyama, S. (1991). Least-squares estimation of transformation parameters between two point patterns. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, *13*(4), 376–380.

Urmson, C., Bagnell, J. A., Baker, C. R., Hebert, M., Kelly, A., Rajkumar, R., et al. (2007). Tartan racing: A multi-modal approach to the darpa urban challenge.

Weingarten, J., & Siegwart, R. (2005). Ekf-based 3d slam for structured environment reconstruction. In: *2005 IEEE/RSJ international conference on intelligent robots and systems, 2005 (IROS 2005)* (pp. 3834–3839). IEEE.

Weingarten, J., & Siegwart, R. (2016) 3d slam using planar segments. In: *2006 IEEE/RSJ International Conference on Intelligent Robots and Systems* (pp. 3062–3067). IEEE.

Whelan, T., Kaess, M., Fallon, M., Johannsson, H., Leonard, J., & McDonald, J. (2012). *Kintinuous: Spatially extended kinectfusion*. CSAIL Technical Reports

Wright, S., & Nocedal, J. (1999). *Numerical optimization* (Vol. 2). New York: Springer.

Zhang, J., & Singh, S. (2014a). LOAM: Lidar odometry and mapping in real-time. In: *Robotics: Science and Systems Conference (RSS)*. Berkeley, CA.

Zhang, J., & Singh, S. (2014b). *Loam velodyne: A realtime method for state estimation and mapping using a 3d lidar*. https://github.com/laboshinl/loam_velodyne.

W. Shane Grant received his B.S. degree in Computer Engineering from the University of California, San Diego in 2010, and his Ph.D. in Computer Science from the University of Southern California in 2018. His research interests include biologically-inspired computational vision focusing on learning, as well as perception and coordination for robotics. Dr. Grant currently works as a research scientist for inVia Robotics.

Randolph C. Voorhies received his B.S, M.S, and Ph.D. degrees all from the University of Southern California in 2006, 2009, and 2015 respectively. He is currently the CTO and co-founder of inVia Robotics, a Los Angeles based warehouse automation company. His research interests include real-time localisation and mapping, high speed robot control, and distributed multi-agent systems.

Laurent Itti received his M.S. degree in Image Processing from the Ecole Nationale Superieure des Telecommunications (Paris, France) in 1994, and his Ph.D. in Computation and Neural Systems from Caltech (Pasadena, California) in 2000. He has since then been an Assistant, Associate, and now Full Professor of Computer Science, Psychology, and Neuroscience at the University of Southern California. Dr. Itti's research interests are in biologically-inspired computational vision, in particular in the domains of visual attention, scene understanding, control of eye movements, and surprise. This basic research has technological applications to, among others, video compression, target detection, and robotics. Dr. Itti has co-authored over 175 publications in peer-reviewed journals, books and conferences, three patents, and several open-source neuromorphic vision software toolkits.