Making a robot to operate autonomously
A set of reference papers

Included papers:

- D. Scaramuzza, F. Fraundorfer "Visual Odometry – Part I: The First 30 Years and Fundamentals"
- Giorgio Grisseti, Rainer Kummerle, Cyrril Stachniss, Wolfram Burgard, "A Tutorial on Graph-based SLAM"
- Ji Zhang, S. Singh, "Low-drift and real-time odometry and mapping"
- M. Achtelik, M. Achtelik, S. Weiss, R. Siegwart, "Onboard IMU and Monocular Vision Based Control for MAVs in Unknown In- and Outdoor Environments"
- S. Karaman, E. Frazzoli, "Incremental Sampling-based Algorithms for Optimal Motion Planning"
- S. Karaman, M. R. Walter, A. Perez, E. Frazzoli, S. Teller, "Anytime Motion Planning using the RRT*

These papers should by no means considered a "complete reference". They correspond to good reads and should enable you to identify further sources of study. They cover two topics, namely localization and collision-free motion planning. These components are fundamental into making a robot autonomous, alongside with the control, overall state estimation, path and task planning functions and more. For further references or interest to work on robotics, contact us at the Autonomous Robots Lab (kalexis@unr.edu).

Visual Odometry

Part I: The First 30 Years and Fundamentals

By Davide Scaramuzza and Friedrich Fraundorfer

Visual odometry (VO) is the process of estimating the egomotion of an agent (e.g., vehicle, human, and robot) using only the input of a single or multiple cameras attached to it. Application domains include robotics, wearable computing, augmented reality, and automotive. The term *VO* was coined in 2004 by Nister in his landmark paper [1]. The term was chosen for its similarity to wheel odometry, which incrementally estimates the motion of a vehicle by integrating the number of turns of its wheels over time. Likewise, VO operates by incrementally estimating the pose of the vehicle through examination of the changes that motion induces on the images of its onboard cameras. For VO to work effectively, there should be sufficient illumination in the environment and a static scene with enough texture to allow apparent motion to be extracted. Furthermore, consecutive frames should be captured by ensuring that they have sufficient scene overlap.

Digital Object Identifier 10.1109/MRA.2011.943233
Date of publication: 8 December 2011

The advantage of VO with respect to wheel odometry is that VO is not affected by wheel slip in uneven terrain or other adverse conditions. It has been demonstrated that compared to wheel odometry, VO provides more accurate trajectory estimates, with relative position error ranging from 0.1 to 2%. This capability makes VO an interesting supplement to wheel odometry and, additionally, to other navigation systems such as global positioning system (GPS), inertial measurement units (IMUs), and laser odometry (similar to VO, laser odometry estimates the egomotion of a vehicle by scan-matching of consecutive laser scans). In GPS-denied environments, such as underwater and aerial, VO has utmost importance.

This two-part tutorial and survey provides a broad introduction to VO and the research that has been undertaken from 1980 to 2011. Although the first two decades witnessed many offline implementations, only in the third decade did real-time working systems flourish, which has led VO to be used on another planet by two Mars-exploration rovers for the first time. Part I (this tutorial) presents a historical review of the first 30 years of research in this field and its fundamentals. After a brief discussion on camera

1070-9932/11/$26.00©2011 IEEE

modeling and calibration, it describes the main motion-estimation pipelines for both monocular and binocular scheme, outlining pros and cons of each implementation. Part II will deal with feature matching, robustness, and applications. It will review the main point-feature detectors used in VO and the different outlier-rejection schemes. Particular emphasis will be given to the random sample consensus (RANSAC), and the distinct tricks devised to speed it up will be discussed. Other topics covered will be error modeling, location recognition (or loop-closure detection), and bundle adjustment.

This tutorial provides both the experienced and non-expert user with guidelines and references to algorithms to build a complete VO system. Since an ideal and unique VO solution for every possible working environment does not exist, the optimal solution should be chosen carefully according to the specific navigation environment and the given computational resources.

History of Visual Odometry

The problem of recovering relative camera poses and three-dimensional (3-D) structure from a set of camera images (calibrated or noncalibrated) is known in the computer vision community as *structure from motion* (SFM). Its origins can be dated back to works such as [2] and [3]. VO is a particular case of SFM. SFM is more general and tackles the problem of 3-D reconstruction of both the structure and camera poses from sequentially ordered or unordered image sets. The final structure and camera poses are typically refined with an offline optimization (i.e., bundle adjustment), whose computation time grows with the number of images [4]. Conversely, VO focuses on estimating the 3-D motion of the camera sequentially—as a new frame arrives—and in real time. Bundle adjustment can be used to refine the local estimate of the trajectory.

The problem of estimating a vehicle's egomotion from visual input alone started in the early 1980s and was described by Moravec [5]. It is interesting to observe that most of the early research in VO [5]–[9] was done for planetary rovers and was motivated by the NASA Mars exploration program in the endeavor to provide all-terrain rovers with the capability to measure their 6-degree-of-freedom (DoF) motion in the presence of wheel slippage in uneven and rough terrains.

The work of Moravec stands out not only for presenting the first motion-estimation pipeline—whose main functioning blocks are still used today—but also for describing one of the earliest corner detectors (after the first one proposed in 1974 by Hannah [10]) which is known today as the *Moravec corner detector* [11], a predecessor of the one proposed by Forstner [12] and Harris and Stephens [3], [82].

Moravec tested his work on a planetary rover equipped with what he termed a *slider stereo*: a single camera sliding on a rail. The robot moved in a stop-and-go fashion, digitizing and analyzing images at every location. At each stop, the camera slid horizontally taking nine pictures at equidistant intervals. Corners were detected in an image using his operator and matched along the epipolar lines of the other eight frames using normalized cross correlation. Potential matches at the next robot locations were found again by correlation using a coarse-to-fine strategy to account for large-scale changes. Outliers were subsequently removed by checking for depth inconsistencies in the eight stereo pairs. Finally, motion was computed as the rigid body transformation to align the triangulated 3-D points seen at two consecutive robot positions. The system of equation was solved via a weighted least square, where the weights were inversely proportional to the distance from the 3-D point.

The advantage of VO with respect to wheel odometry is that VO is not affected by wheel slip in uneven terrain or other adverse conditions.

Although Moravec used a single sliding camera, his work belongs to the class of stereo VO algorithms. This terminology accounts for the fact that the relative 3-D position of the features is directly measured by triangulation at every robot location and used to derive the relative motion. Trinocular methods belong to the same class of algorithms. The alternative to stereo vision is to use a single camera. In this case, only bearing information is available. The disadvantage is that motion can only be recovered up to a scale factor. The absolute scale can then be determined from direct measurements (e.g., measuring the size of an element in the scene), motion constraints, or from the integration with other sensors, such as IMU, air-pressure, and range sensors. The interest in monocular methods is due to the observation that stereo VO can degenerate to the monocular case when the distance to the scene is much larger than the stereo baseline (i.e., the distance between the two cameras). In this case, stereo vision becomes ineffective and monocular methods must be used. Over the years, monocular and stereo VOs have almost progressed as two independent lines of research. In the remainder of this section, we have surveyed the related work in these fields.

Stereo VO

Most of the research done in VO has been produced using stereo cameras. Building upon Moravec's work, Matthies and Shafer [6], [7] used a binocular system and Moravec's procedure for detecting and tracking corners. Instead of using a scalar representation of the uncertainty as Moravec did, they took advantage of the error covariance matrix of the triangulated features and incorporated it into the motion estimation step. Compared to Moravec, they demonstrated superior results in trajectory recovery for a planetary rover, with 2% relative error on a 5.5-m path. Olson et al. [9], [13] later extended that work by

introducing an absolute orientation sensor (e.g., compass or omnidirectional camera) and using the Forstner corner detector, which is significantly faster to compute than Moravec's operator. They showed that the use of camera egomotion estimates alone results in accumulation errors with superlinear growth in the distance traveled, leading to increased orientation errors. Conversely, when an absolute orientation sensor is incorporated, the error growth can be reduced to a linear function of the distance traveled. This led them to a relative position error of 1.2% on a 20-m path.

Keyframe selection is a very important step in VO and should always be done before updating the motion.

Lacroix et al. [8] implemented a stereo VO approach for planetary rovers similar to those explained earlier. The difference lies in the selection of key points. Instead of using the Forstner detector, they used dense stereo and, then, selected the candidate key points by analyzing the correlation function around its peaks—an approach that was later exploited in [14], [15], and other works. This choice was based on the observation that there is a strong correlation between the shape of the correlation curve and the standard deviation of the feature depth. This observation was later used by Cheng et al. [16], [17] in their final VO implementation onboard the Mars rovers. They improved on the earlier implementation by Olson et al. [9], [13] in two areas. First, after using the Harris corner detector, they utilized the curvature of the correlation function around the feature—as proposed by Lacroix et al.—to define the error covariance matrix of the image point. Second, as proposed by Nister et al. [1], they used the random sample consensus (RANSAC) RANSAC [18] in the least-squares motion estimation step for outlier rejection.

A different approach to motion estimation and outlier removal for an all-terrain rover was proposed by Milella and Siegwart [14]. They used the Shi-Tomasi approach [19] for corner detection, and similar to Lacroix, they retained those points with high confidence in the stereo disparity map. Motion estimation was then solved by first using least squares, as in the methods earlier, and then the iterative closest point (ICP) algorithm [20]—an algorithm popular for 3-D registration of laser scans—for pose refinement. For robustness, an outlier removal stage was incorporated into the ICP.

The works mentioned so far have in common that the 3-D points are triangulated for every stereo pair, and the relative motion is solved as a 3-D-to-3-D point registration (alignment) problem. A completely different approach was proposed in 2004 by Nister et al. [1]. Their paper is known not only for coining the term VO but also for providing the first real-time long-run implementation with a robust outlier rejection scheme. Nister et al. improved the earlier implementations in several areas. First, contrary to all previous works, they did not track features among frames but detected features (Harris corners) independently in all frames and only allowed matches between features. This has the benefit of avoiding feature drift during cross-correlation-based tracking. Second, they did not compute the relative motion as a 3-D-to-3-D point registration problem but as a 3-D-to-two-dimensional (2-D) camera-pose estimation problem (these methods are described in the "Motion Estimation" section). Finally, they incorporated RANSAC outlier rejection into the motion estimation step.

A different motion estimation scheme was introduced by Comport et al. [21]. Instead of using 3-D-to-3-D point registration or 3-D-to-2-D camera-pose estimation techniques, they relied on the quadrifocal tensor, which allows motion to be computed from 2-D-to-2-D image matches without having to triangulate 3-D points in any of the stereo pairs. The benefit of using directly raw 2-D points in lieu of triangulated 3-D points lays in a more accurate motion computation.

Monocular VO

The difference from the stereo scheme is that in the monocular VO, both the relative motion and 3-D structure must be computed from 2-D bearing data. Since the absolute scale is unknown, the distance between the first two camera poses is usually set to one. As a new image arrives, the relative scale and camera pose with respect to the first two frames are determined using either the knowledge of 3-D structure or the trifocal tensor [22].

Successful results with a single camera over long distances (up to several kilometers) have been obtained in the last decade using both perspective and omnidirectional cameras [23]–[29]. Related works can be divided into three categories: feature-based methods, appearance-based methods, and hybrid methods. Feature-based methods are based on salient and repeatable features that are tracked over the frames; appearance-based methods use the intensity information of all the pixels in the image or subregions of it; and hybrid methods use a combination of the previous two.

In the first category are the works by the authors in [1], [24], [25], [27], and [30]–[32]. The first real-time, large-scale VO with a single camera was presented by Nister et al. [1]. They used RANSAC for outlier rejection and 3-D-to-2-D camera-pose estimation to compute the new upcoming camera pose. The novelty of their paper is the use of a five-point minimal solver [33] to calculate the motion hypotheses in RANSAC. After that paper, five-point RANSAC became very popular in VO and was used in several other works [23], [25], [27]. Corke et al. [24] provided an approach for monocular VO based on omnidirectional imagery from a catadioptric camera and optical flow. Lhuillier [25] and Mouragnon et al. [30] presented an approach based on local windowed-bundle adjustment to recover both the motion and the 3-D map (this means that bundle adjustment is performed over a window of the last m frames). Again, they used the five-point RANSAC in [33] to remove the outliers. Tardif et al. [27] presented an

approach for VO on a car over a very long run (2.5 km) without bundle adjustment. Contrary to the previous work, they decoupled the rotation and translation estimation. The rotation was estimated by using points at infinity and the translation from the recovered 3-D map. Erroneous correspondences were removed with five-point RANSAC.

Among the appearance-based or hybrid approaches are the works by the authors in [26], [28], and [29]. Goecke et al. [26] used the Fourier–Mellin transform for registering perspective images of the ground plane taken from a car. Milford and Wyeth [28] presented a method to extract approximate rotational and translational velocity information from a single perspective camera mounted on a car, which was then used in a RatSLAM scheme [34]. They used template tracking on the center of the scene. A major drawback with appearance-based approaches is that they are not robust to occlusions. For this reason, Scaramuzza and Siegwart [29] used image appearance to estimate the rotation of the car and features from the ground plane to estimate the translation and the absolute scale. The feature-based approach was also used to detect failures of the appearance-based method.

All the approaches mentioned earlier are designed for unconstrained motion in 6 DoF. However, several VO works have been specifically designed for vehicles with motion constraints. The advantage is decreased computation time and improved motion accuracy. For instance, Liang and Pears [35], Ke and Kanade [36], Wang et al. [37], and Guerrero et al. [38] took advantage of homographies for estimating the egomotion on a dominant ground plane. Scaramuzza et al. [31], [39] introduced a one-point RANSAC outlier rejection based on the vehicle nonholonomic constraints to speed up egomotion estimation to 400 Hz. In the follow-up work, they showed that nonholonomic constraints allow the absolute scale to be recovered from a single camera whenever the vehicle makes a turn [40]. Following that work, vehicle nonholonomic constraints have also been used by Pretto et al. [32] for improving feature tracking and by Fraundorfer et al. [41] for windowed bundle adjustment (see the following section).

Reducing the Drift

Since VO works by computing the camera path incrementally (pose after pose), the errors introduced by each new frame-to-frame motion accumulate over time. This generates a drift of the estimated trajectory from the real path. For some applications, it is of utmost importance to keep drift as small as possible, which can be done through local optimization over the last m camera poses. This approach—called *sliding window bundle adjustment* or *windowed bundle adjustment*—has been used in several works, such as [41]–[44]. In particular, on a 10-km VO experiment, Konolige et al. [43] demonstrated that windowed-bundle adjustment can decrease the final position error by a factor of 2–5. Obviously, the VO drift can also be reduced through combination with other sensors, such as GPS and laser, or even with only an IMU [43], [45], [46].

V-SLAM

Although this tutorial focuses on VO, it is worth mentioning the parallel line of research undertaken by visual simultaneous localization and mapping (V-SLAM). For an in-depth study of the SLAM problem, the reader is referred to two tutorials on this topic by Durrant-Whyte and Bailey [47], [48]. Two methodologies have become predominant in V-SLAM: 1) filtering methods fuse the information from all the images with a probability distribution [49] and 2) nonfiltering methods (also called *keyframe methods*) retain the optimization of global bundle adjustment to selected keyframes [50]. The main advantages of either approach have been evaluated and summarized in [51].

VO is only concerned with the local consistency of the trajectory, whereas SLAM with the global consistency.

In the last few years, successful results have been obtained using both single and stereo cameras [49], [52]–[62]. Most of these works have been limited to small, indoor workspaces and only a few of them have recently been designed for large-scale areas [54], [60], [62]. Some of the early works in real-time V-SLAM were presented by Chiuso et al. [52], Deans [53], and Davison [49] using a full-covariance Kalman approach. The advantage of Davison's work was to account for repeatable localization after an arbitrary amount of time. Later, Handa et al. [59] improved on that work using an active matching technique based on a probabilistic framework. Civera et al. [60] built upon that work by proposing a combination of one-point RANSAC within the Kalman filter that uses the available prior probabilistic information from the filter in the RANSAC model-hypothesis stage. Finally, Strasdat et al. [61] presented a new framework for large-scale V-SLAM that takes advantage of the keyframe optimization approach [50] while taking into account the special character of SLAM.

VO Versus V-SLAM

In this section, the relationship of VO with V-SLAM is analyzed. The goal of SLAM in general (and V-SLAM in particular) is to obtain a global, consistent estimate of the robot path. This implies keeping a track of a map of the environment (even in the case where the map is not needed per se) because it is needed to realize when the robot returns to a previously visited area. (This is called *loop closure*. When a loop closure is detected, this information is used to reduce the drift in both the map and camera path. Understanding when a loop closure occurs and efficiently integrating this new constraint into the current map are two of the main issues in SLAM.) Conversely, VO aims at recovering the path incrementally, pose after pose, and potentially optimizing only over the last n poses of the

path (this is also called *windowed bundle adjustment*). This sliding window optimization can be considered equivalent to building a local map in SLAM; however, the philosophy is different: in VO, we only care about local consistency of the trajectory and the local map is used to obtain a more accurate estimate of the local trajectory (for example, in bundle adjustment), whereas SLAM is concerned with the global map consistency.

VO can be used as a building block for a complete SLAM algorithm to recover the incremental motion of the camera; however, to make a complete SLAM method, one must also add some way to detect loop closing and possibly a global optimization step to obtain a metrically consistent map (without this step, the map is still topologically consistent).

In the motion estimation step, the camera motion between the current and the previous image is computed.

If the user is only interested in the camera path and not in the environment map, there is still the possibility of using a complete V-SLAM method instead of one of the VO techniques described in this tutorial. A V-SLAM method is potentially much more precise, because it enforces many more constraints on the path, but not necessarily more robust (e.g., outliers in loop closing can severely affect the map consistency). In addition, it is more complex and computationally expensive.

In the end, the choice between VO and V-SLAM depends on the tradeoff between performance and consistency, and simplicity in implementation. Although the global consistency of the camera path is sometimes desirable, VO trades off consistency for real-time performance, without the need to keep track of all the previous history of the camera.

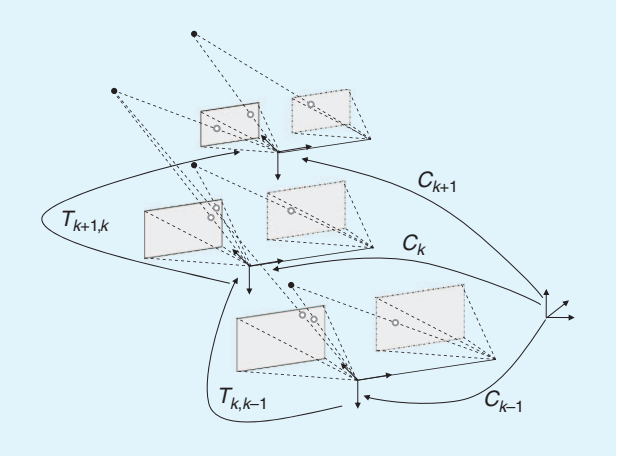


Figure 1. An illustration of the visual odometry problem. The relative poses $T_{k,k-1}$ of adjacent camera positions (or positions of a camera system) are computed from visual features and concatenated to get the absolute poses C_k with respect to the initial coordinate frame at $k = 0$.

Formulation of the VO Problem

An agent is moving through an environment and taking images with a rigidly attached camera system at discrete time instants k. In case of a monocular system, the set of images taken at times k is denoted by $I_{0:n} = \{I_0, \ldots, I_n\}$. In case of a stereo system, there are a left and a right image at every time instant, denoted by $I_{l,0:n} = \{I_{l,0}, \ldots, I_{l,n}\}$ and $I_{r,0:n} = \{I_{r,0}, \ldots, I_{r,n}\}$. Figure 1 shows an illustration of this setting.

For simplicity, the camera coordinate frame is assumed to be also the agent's coordinate frame. In case of a stereo system, without loss of generality, the coordinate system of the left camera can be used as the origin.

Two camera positions at adjacent time instants $k-1$ and k are related by the rigid body transformation $T_{k,k-1} \in \mathbb{R}^{4\times4}$ of the following form:

$$T_{k,k-1} = \begin{bmatrix} R_{k,k-1} & t_{k,k-1} \\ 0 & 1 \end{bmatrix}, \tag{1}$$

where $R_{k,k-1} \in SO(3)$ is the rotation matrix, and $t_{k,k-1} \in \mathbb{R}^{3\times1}$ the translation vector. The set $T_{1:n} = \{T_{1,0}, \ldots, T_{n,n-1}\}$ contains all subsequent motions. To simplify the notation, from now on, T_k will be used instead of $T_{k,k-1}$. Finally, the set of camera poses $C_{0:n} = \{C_0, \ldots, C_n\}$ contains the transformations of the camera with respect to the initial coordinate frame at $k = 0$. The current pose C_n can be computed by concatenating all the transformations T_k $(k = 1 \ldots n)$, and, therefore, $C_n = C_{n-1}T_n$, with C_0 being the camera pose at the instant $k = 0$, which can be set arbitrarily by the user.

The main task in VO is to compute the relative transformations T_k from the images I_k and I_{k-1} and then to concatenate the transformations to recover the full trajectory $C_{0:n}$ of the camera. This means that VO recovers the path incrementally, pose after pose. An iterative refinement over the last m poses can be performed after this step to obtain a more accurate estimate of the local trajectory. This iterative refinement works by minimizing the sum of the squared reprojection errors of the reconstructed 3-D points (i.e., the 3-D map) over the last m images (this is called *windowed-bundle adjustment*, because it is performed on a window of m frames. Bundle adjustment will be described in Part II of this tutorial). The 3-D points are obtained by triangulation of the image points (see the "Triangulation and Keyframe Selection" section).

As mentioned in the "Monocular VO" section, there are two main approaches to compute the relative motion T_k: appearance-based (or global) methods, which use the intensity information of all the pixels in the two input images, and feature-based methods, which only use salient and repeatable features extracted (or tracked) across the images. Global methods are less accurate than feature-based methods and are computationally more expensive. (As observed in the "History of VO" section, most appearance-based methods have been applied to monocular VO. This is due to ease of implementation compared with the

stereo camera case.) Feature-based methods require the ability to robustly match (or track) features across frames but are faster and more accurate than global methods. Therefore, most VO implementations are feature based.

The VO pipeline is summarized in Figure 2. For every new image I_k (or image pair in the case of a stereo camera), the first two steps consist of detecting and matching 2-D features with those from the previous frames. Two-dimensional features that are the reprojection of the same 3-D feature across different frames are called *image correspondences*. (As will be explained in Part II of this tutorial, we distinguish between feature matching and feature tracking. The first one consists of detecting features independently in all the images and then matching them based on some similarity metrics; the second one consists of finding features in one image and then tracking them in the next images using a local search technique, such as correlation.) The third step consists of computing the relative motion T_k between the time instants $k-1$ and k. Depending on whether the correspondences are specified in three or two dimensions, there are three distinct approaches to tackle this problem (see the "Motion Estimation" section). The camera pose C_k is then computed by concatenation of T_k with the previous pose. Finally, an iterative refinement (bundle adjustment) can be done over the last m frames to obtain a more accurate estimate of the local trajectory.

Motion estimation is explained in this tutorial (see "Motion Estimation" section). Feature detection and matching and bundle adjustment will be described in Part II. Also, notice that for an accurate motion computation, feature correspondences should not contain outliers (i.e., wrong data associations). Ensuring accurate motion estimation in the presence of outliers is the task of robust estimation, which will be described in Part II of this tutorial. Most VO implementations assume that the camera is calibrated. To this end, the next section reviews the standard models and calibration procedures for perspective and omnidirectional cameras.

Camera Modeling and Calibration

VO can be done using both perspective and omnidirectional cameras. In this section, we review the main models.

Perspective Camera Model

The most used model for perspective camera assumes a pinhole projection system: the image is formed by the intersection of the light rays from the objects through the center of the lens (projection center), with the focal plane [Figure 3(a)]. Let $X = [x, y, z]^\top$ be a scene point in the camera reference frame and $p = [u, v]^\top$ its projection on the image plane measured in pixels. The mapping from the 3-D world to the 2-D image is given by the perspective projection equation:

$$\lambda \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = KX = \begin{bmatrix} \alpha_u & 0 & u_0 \\ 0 & \alpha_v & v_0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \end{bmatrix}, \qquad (2)$$

where λ is the depth factor, α_u and α_v the focal lengths, and u_0, v_0 the image coordinates of the projection center. These parameters are called *intrinsic parameters*. When the field of view of the camera is larger than 45°, the effects of the radial distortion may become visible and can be modeled using a second- (or higher)-order polynomial. The derivation of the complete model can be found in computer vision textbooks, such as [22] and [63]. Let $\tilde{p} = [\tilde{u}, \tilde{v}, 1]^\top = K^{-1}[u, v, 1]^\top$ be the normalized image coordinates. Normalized coordinates will be used throughout in the following sections.

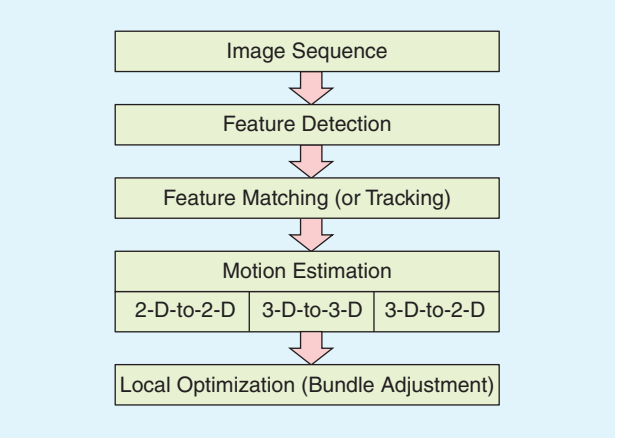


Figure 2. A block diagram showing the main components of a VO system.

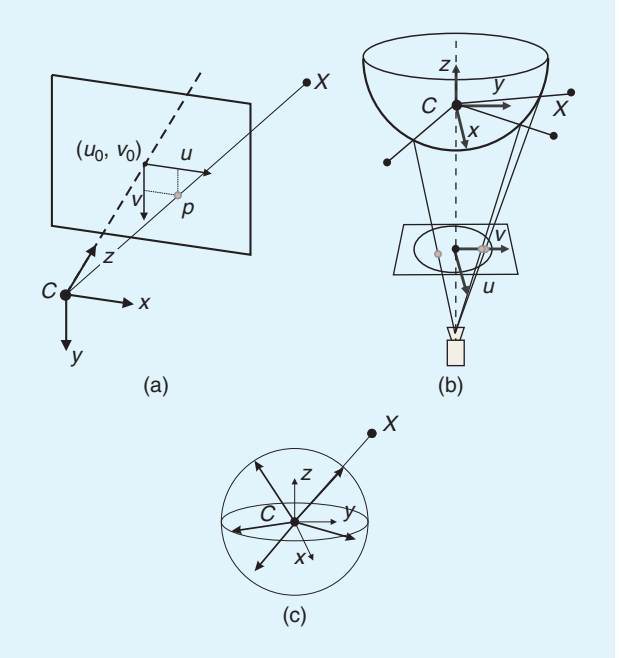


Figure 3. (a) Perspective projection, (b) catadioptric projection, and (c) a spherical model for perspective and omnidirectional cameras. Image points are represented as directions to the viewed points normalized on the unit sphere.

Omnidirectional Camera Model

Omnidirectional cameras are cameras with wide field of view (even more than 180°) and can be built using fish-eye lenses or by combining standard cameras with mirrors [the latter are called *catadioptric cameras*, Figure 3(b)]. Typical mirror shapes in catadioptric cameras are quadratic surfaces of revolution (e.g., paraboloid or hyperboloid), because they guarantee a single projection center, which makes it possible to use the motion estimation theory presented in the "Motion Estimation" section.

In GPS-denied environments, VO becomes of utmost importance.

Currently, there are two accepted models for omnidirectional cameras. The first one proposed by Geyer and Daniilidis [64] is for general catadioptric cameras (parabolic or hyperbolic), while the second one proposed by Scaramuzza et al. [65] is a unified model for both fish-eye and catadioptric cameras. A survey of these two models can be found in [66] and [67]. The projection equation of the unified model is as follows:

$$\lambda \begin{bmatrix} u \\ v \\ a_0 + a_1\rho + \cdots + a_{n-1}\rho^{n-1} \end{bmatrix} = \begin{bmatrix} x \\ y \\ z \end{bmatrix}, \qquad (3)$$

where $\rho = \sqrt{u^2 + v^2}$ and $a_0, a_1, \ldots, a_n$ are intrinsic parameters that depend on the type of mirror or fish-eye lens. As shown in [65], $n = 4$ is a reasonable choice for a large variety of mirrors and fish-eye lenses. Finally, this model assumes that the image plane satisfies the ideal property that the axes of symmetry of the camera and mirror are aligned. Although this assumption holds for most catadioptric and fish-eye cameras, misalignments can be modeled by introducing a perspective projection between the ideal and real-image plane [66].

Spherical Model

As mentioned earlier, it is desirable that the camera possesses a single projection center (also called *single effective viewpoint*). In a catadioptric camera, this happens when the rays reflected by the mirror intersect all in a single point (namely C). The existence of this point allows us to model any omnidirectional projection as a mapping from the single viewpoint to a sphere. For convenience, a unit sphere is usually adopted.

It is important to notice that the spherical model applies not only to omnidirectional cameras but also to perspective cameras. If the camera is calibrated, any point in the perspective or omnidirectional image can be mapped into a vector on the unit sphere. As can be observed in Figure 3(c), these unit vectors represent the directions to the viewed scene points. These vectors are called *normalized image points on the unit sphere*.

Camera Calibration

The goal of calibration is to accurately measure the intrinsic and extrinsic parameters of the camera system. In a multicamera system (e.g., stereo and trinocular), the extrinsic parameters describe the mutual position and orientation between each camera pair. The most popular method uses a planar checkerboard-like pattern. The position of the squares on the board is known. To compute the calibration parameters accurately, the user must take several pictures of the board shown at different positions and orientations by ensuring that the field of view of the camera is filled as much as possible. The intrinsic and extrinsic parameters are then found through a least-square minimization method. The input data are the 2-D positions of the corners of the squares of the board and their corresponding pixel coordinates in each image.

Many camera calibration toolboxes have been devised for MATLAB and C. An up-to-date list can be found in [68]. Among these, the most popular ones for MATLAB are given in [69] and [70]–[72]—for perspective and omnidirectional cameras, respectively. A C implementation of camera calibration for perspective cameras can be found in OpenCV [73], the open-source computer vision library.

Motion Estimation

Motion estimation is the core computation step performed for every image in a VO system. More precisely, in the motion estimation step, the camera motion between the current image and the previous image is computed. By concatenation of all these single movements, the full trajectory of the camera and the agent (assuming that the camera is rigidly mounted) can be recovered. This section explains how the transformation T_k between two images I_{k-1} and I_k can be computed from two sets of corresponding features f_{k-1}, f_k at time instants $k-1$ and k, respectively. Depending on whether the feature correspondences are specified in two or three dimensions, there are three different methods.

- *2-D-to-2-D:* In this case, both f_{k-1} and f_k are specified in 2-D image coordinates.
- *3-D-to-3-D:* In this case, both f_{k-1} and f_k are specified in 3-D. To do this, it is necessary to triangulate 3-D points at each time instant; for instance, by using a stereo camera system.
- *3-D-to-2-D:* In this case, f_{k-1} are specified in 3-D and f_k are their corresponding 2-D reprojections on the image I_k. In the monocular case, the 3-D structure needs to be triangulated from two adjacent camera views (e.g., I_{k-2} and I_{k-1}) and then matched to 2-D image features in a third view (e.g., I_k). In the monocular scheme, matches over at least three views are necessary.

Notice that features can be points or lines. In general, due to the lack of lines in unstructured scenes, point features are used in VO. An in-depth review of these three approaches for both point and line features can be found in [74]. The formulation given in this tutorial is for point features only.

2-D to 2-D: Motion from Image Feature Correspondences

Estimating the Essential Matrix

The geometric relations between two images I_k and I_{k-1} of a calibrated camera are described by the so-called essential matrix E. E contains the camera motion parameters up to an unknown scale factor for the translation in the following form:

$$E_k \simeq \hat{t}_k R_k, \tag{4}$$

where $t_k = [t_x, t_y, t_z]^\top$ and

$$\hat{t}_k = \begin{bmatrix} 0 & -t_z & t_y \\ t_z & 0 & -t_x \\ -t_y & t_x & 0 \end{bmatrix}. \tag{5}$$

The symbol $\simeq$ is used to denote that the equivalence is valid up to a multiplicative scalar.

The essential matrix can be computed from 2-D-to-2-D feature correspondences, and rotation and translation can directly be extracted from E. The main property of 2-D-to-2-D-based motion estimation is the epipolar constraint, which determines the line on which the corresponding feature point $\tilde{p}'$ of $\tilde{p}$ lies in the other image (Figure 4). This constraint can be formulated by $\tilde{p}'^\top E\tilde{p} = 0$, where $\tilde{p}'$ is a feature location in one image (e.g., I_k) and $\tilde{p}$ is the location of its corresponding feature in another image (e.g., I_{k-1}). $\tilde{p}$ and $\tilde{p}'$ are normalized image coordinates. For the sake of simplicity, throughout the following sections, normalized coordinates in the form $\tilde{p} = [\tilde{u}, \tilde{v}, 1]^\top$ will be used (see the Perspective Camera Model" section). However, very similar equations can also be derived for normalized coordinates on the unit sphere (see the "Spherical Model" section).

The essential matrix can be computed from 2-D-to-2-D feature correspondences using the epipolar constraint. The minimal case solution involves five 2-D-to-2-D correspondences [75] and an efficient implementation proposed by Nister in [76]. Nister's five-point algorithm has become the standard for 2-D-to-2-D motion estimation in the presence of outliers (the problem of robust estimation will be tackled in Part II of this tutorial). A simple and straightforward solution for $n \geq 8$ noncoplanar points is the Longuet-Higgins' eight-point algorithm [2], which is summarized here. Each feature match gives a constraint of the following form:

$$[\,\tilde{u}\tilde{u}' \quad \tilde{u}'\tilde{v} \quad \tilde{u}' \quad \tilde{u}\tilde{v}' \quad \tilde{v}\tilde{v}' \quad \tilde{v}' \quad \tilde{u} \quad \tilde{v} \quad 1\,]E = 0, \tag{6}$$

where $E = [e_1\, e_2\, e_3\, e_4\, e_5\, e_6\, e_7\, e_8\, e_9]^\top$.

Stacking the constraints from eight points gives the linear equation system $AE = 0$, and by solving the system, the parameters of E can be computed. This homogeneous equation system can easily be solved using singular value decomposition (SVD) [2]. Having more than eight points leads to an overdetermined system to solve in the least-squares sense and provides a degree of robustness to noise. The SVD of A has the form $A = USV^\top$, and the least-squares estimate of E with $||E|| = 1$ can be found as the last column of V. However, this linear estimation of E does not fulfill the inner constraints of an essential matrix, which come from the multiplication of the rotation matrix R and the skew-symmetric translation matrix $\hat{t}$. These constraints are visible in the singular values of the essential matrix. A valid essential matrix after SVD is $E = USV^\top$ and has $\mathrm{diag}(S) = \{s, s, 0\}$, which means that the first and second singular values are equal and the third one is zero. To get a valid E that fulfills the constraints, the solution needs to be projected onto the space of valid essential matrices. The projected essential matrix is $\bar{E} = U\mathrm{diag}\{1, 1, 0\}V^\top$.

Observe that the solution of the eight-point algorithm is degenerate when the 3-D points are coplanar. Conversely, the five-point algorithm works also for coplanar points. Finally, observe that the eight-point algorithm works for both calibrated (perspective or omnidirectional) and uncalibrated (only perspective) cameras, whereas the five-point algorithm assumes that the camera (perspective or omnidirectional) is calibrated.

Extracting R and t from E

From the estimate of $\bar{E}$, the rotation and translation parts can be extracted. In general, there are four different solutions for R, t for one essential matrix; however, by triangulation of a single point, the correct R, t pair can be identified. The four solutions are

$$R = U(\pm W^\top)V^\top,$$
$$\hat{t} = U(\pm W)SU^\top,$$

where

$$W^\top = \begin{bmatrix} 0 & \pm 1 & 0 \\ \mp 1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix}. \tag{7}$$

An efficient decomposition of E into R and t is described in [76].

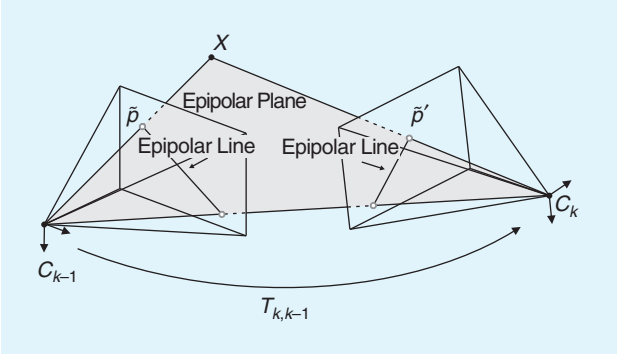


Figure 4. An illustration of the epipolar constraint.

After selecting the correct solution by triangulation of a point and choosing the solution where the point is in front of both cameras, a nonlinear optimization of the rotation and translation parameters should be performed using the estimate R, t as initial values. The function to minimize is the reprojection error defined in (10).

Computing the Relative Scale

To recover the trajectory of an image sequence, the different transformations $T_{0:n}$ have to be concatenated. To do this, the proper relative scales need to be computed as the absolute scale of the translation cannot be computed from two images. However, it is possible to compute relative scales for the subsequent transformations. One way of doing this is to triangulate 3-D points X_{k-1} and X_k from two subsequent image pairs. From the corresponding 3-D points, the relative distances between any combination of two 3-D points can be computed. The proper scale can then be determined from the distance ratio r between a point pair in X_{k-1} and a pair in X_k.

Bundle adjustment can be used to refine the local estimate of the trajectory.

$$r = \frac{||X_{k-1,i} - X_{k-1,j}||}{||X_{k,i} - X_{k,j}||}. \tag{8}$$

For robustness, the scale ratios for many point pairs are computed and the mean (or in presence of outliers, the median) is used. The translation vector t is then scaled with this distance ratio. Observe that the relative-scale computation requires features to be matched (or tracked) over multiple frames (at least three). Instead of performing explicit triangulation of the 3-D points, the scale can also be recovered by exploiting the trifocal constraint between three-view matches of 2-D features [22].

The VO algorithm with the 2-D-to-2-D correspondences is summarized in Algorithm 1.

Algorithm 1. VO from 2-D-to-2-D correspondences.

1) Capture new frame I_k
2) Extract and match features between I_{k-1} and I_k
3) Compute essential matrix for image pair I_{k-1}, I_k
4) Decompose essential matrix into R_k and t_k, and form T_k
5) Compute relative scale and rescale t_k accordingly
6) Concatenate transformation by computing $C_k = C_{k-1}T_k$
7) Repeat from 1).

3D-to-3D: Motion from 3-D Structure Correspondences

For the case of corresponding 3-D-to-3-D features, the camera motion T_k can be computed by determining the aligning transformation of the two 3-D feature sets. Corresponding 3-D-to-3-D features are available in the stereo vision case.

The general solution consists of finding the T_k that minimizes the L_2 distance between the two 3-D feature sets

$$\arg\min_{T_k} \sum_i ||\tilde{X}_k^i - T_k \tilde{X}_{k-1}^i||, \tag{9}$$

where the superscript i denotes the i th feature, and $\tilde{X}_k$, $\tilde{X}_{k-1}$ are the homogeneous coordinates of the 3-D points, i.e., $\tilde{X} = [x, y, z, 1]^\top$.

As shown in [77], the minimal case solution involves three 3-D-to-3-D noncollinear correspondences, which can be used for robust estimation in the presence of outliers (Part II of this tutorial). For the case of $n \geq 3$ correspondences, one possible solution (according to Arun et al. [78]) is to compute the translation part as the difference of the centroids of the 3-D feature sets and the rotation part using SVD. The translation is given by

$$t_k = \overline{X_k} - R\overline{X}_{k-1},$$

where $\bar{\cdot}$ stands for the arithmetic mean value.

The rotation can be efficiently computed using SVD as

$$R_k = VU^\top,$$

where $\mathrm{USV}^\top = \mathrm{svd}((X_{k-1} - \overline{X}_{k-1})(X_k - \overline{X}_k)^\top)$ and X_{k-1} and X_k are sets of corresponding 3-D points.

If the measurement uncertainties of the 3-D points are known, they can be added as weights into the estimation as described by Maimone et al. [17]. The computed transformations have absolute scale, and thus, the trajectory of a sequence can be computed by directly concatenating the transformations.

The VO algorithm with the 3-D-to-3-D correspondences is summarized in Algorithm 2.

Algorithm 2. VO from 3-D-to-3-D correspondences.

1) Capture two stereo image pairs $I_{l,k-1}$, $I_{r,k-1}$ and $I_{l,k}$, $I_{r,k}$
2) Extract and match features between $I_{l,k-1}$ and $I_{l,k}$
3) Triangulate matched features for each stereo pair
4) Compute T_k from 3-D features X_{k-1} and X_k
5) Concatenate transformation by computing $C_k = C_{k-1}T_k$
6) Repeat from 1).

To compute the transformation, it is also possible to avoid the triangulation of the 3-D points in the stereo camera and use quadrifocal constraints instead. This method was pointed out by Comport et al. [21]. The quadrifocal tensor allows computing the transformation directly from 2-D-to-2-D stereo correspondences.

3-D-to-2-D: Motion from 3-D Structure and Image Feature Correspondences

As pointed out by Nister et al. [1], motion estimation from 3-D-to-2-D correspondences is more accurate than from 3-D-to-3-D correspondences because it minimizes the image reprojection error (10) instead of the 3-D-to-3-D feature position error (9). The transformation T_k is computed from the 3-D-to-2-D correspondences X_{k-1} and p_k. X_{k-1} can be estimated from stereo data or, in the monocular case, from triangulation of the image measurements p_{k-1} and p_{k-2}. The latter, however, requires image correspondences across three views.

The general formulation in this case is to find T_k that minimizes the image reprojection error

$$\arg\min_{T_k} \sum_i \| p_k^i - \hat{p}_{k-1}^i \|^2, \tag{10}$$

where $\hat{p}_{k-1}^i$ is the reprojection of the 3-D point X_{k-1}^i into image I_k according to the transformation T_k. This problem is known as *perspective from n points* (PnP) (or resection), and there are many different solutions to it in the literature [79]. As shown in [18], the minimal case involves three 3-D-to-2-D correspondences. This is called *perspective from three points* (P3P) and returns four solutions that can be disambiguated using one or more additional points. (A fast implementation of P3P is described in [80], and C code can be freely downloaded from the authors' Web page.) In the 3-D-to-2-D case, P3P is the standard method for robust motion estimation in the presence of outliers [18]. Robust estimation will be described in Part II of this tutorial.

A simple and straightforward solution to the P*n*P problem for $n \geq 6$ points is the direct linear transformation algorithm [22]. One 3-D-to-2-D point correspondence provides two constraints of the following form for the entries of $P_k = [R|t]$.

$$\begin{bmatrix} 0 & 0 & 0 & 0 & -x & -y & -z & -1 & x\tilde{v} & y\tilde{v} & z\tilde{v} & \tilde{v} \\ x & y & z & 1 & 0 & 0 & 0 & 0 & -x\tilde{u} & -y\tilde{u} & -z\tilde{u} & -\tilde{u} \end{bmatrix} \begin{bmatrix} P^1 \\ P^2 \\ P^3 \end{bmatrix} = 0, \tag{11}$$

where each $P^{j\top}$ is a four vector (the jth row of P_k) and x, y, z are the coordinates of the 3-D points X_{k-1}.

Stacking the constraints of six-point correspondences gives a linear system of equations of the form $AP = 0$. The entries of P can be computed from the nullvector of A, e.g., by using SVD. The rotation and translation parts can easily be extracted from $P_k = [R|t]$. The resulting rotation R is not necessarily orthonormal. However, this is not a problem since both R and t can be refined by nonlinear optimization of the reprojection error as defined in (10).

The 3-D-to-2-D motion estimation assumes that the 2-D image points only come from one camera. This means that for the case of a stereo camera, the 2-D image points are those of either the left or the right camera. Obviously, it is desirable to make use of the image points of both cameras at the same time. A generalized version of the 3-D-to-2-D motion estimation algorithm for nonconcurrent rays (i.e., 2-D image points from multiple cameras) was proposed by Nister in [81] for extrinsically calibrated cameras (i.e., the mutual position and orientation between the cameras is known).

For the monocular case, it is necessary to triangulate 3-D points and estimate the pose from 3-D-to-2-D matches in an alternating fashion. This alternating scheme is often referred to as SFM. Starting from two views, the initial set of 3-D points and the first transformation are computed from 2-D-to-2-D feature matches. Subsequent transformations are then computed from 3-D-to-2-D feature matches. To do this, features need to be matched (or tracked) over multiple frames (at least three). New 3-D features are again triangulated when a new transformation is computed and added to the set of 3-D features. The main challenge of this method is to maintain a consistent and accurate set of triangulated 3-D features and to create 3-D-to-2-D feature matches for at least three adjacent frames.

2-D-to-2-D and 3-D-to-2-D methods are more accurate than 3-D-to-3-D methods.

The VO algorithm with 3-D-to-2-D correspondences is summarized in Algorithm 3.

Algorithm 3. VO from 3-D-to-2-D Correspondences.

1) Do only once:
1.1) Capture two frames I_{k-2}, I_{k-1}
1.2) Extract and match features between them
1.3) Triangulate features from I_{k-2}, I_{k-1}
2) Do at each iteration:
2.1) Capture new frame I_k
2.2) Extract features and match with previous frame I_{k-1}
2.3) Compute camera pose (PnP) from 3-D-to-2-D matches
2.4) Triangulate all new feature matches between I_k and I_{k-1}
2.5) Iterate from 2.1).

Triangulation and Keyframe Selection

Some of the previous motion estimation methods require triangulation of 3-D points (structure) from 2-D image correspondences. Structure computation is also needed by bundle adjustment (Part II of this tutorial) to compute a more accurate estimate of the local trajectory.

Triangulated 3-D points are determined by intersecting back-projected rays from 2-D image correspondences of at least two image frames. In perfect conditions, these rays would intersect in a single 3-D point. However, because of image noise, camera model and calibration errors, and

feature matching uncertainty, they never intersect. Therefore, the point at a minimal distance, in the least-squares sense, from all intersecting rays can be taken as an estimate of the 3-D point position. Notice that the standard deviation of the distances of the triangulated 3-D point from all rays gives an idea of the quality of the 3-D point. Three-dimensional points with large uncertainty will be thrown out. This happens especially when frames are taken at very nearby intervals compared with the distance to the scene points. When this occurs, 3-D points exhibit very large uncertainty. One way to avoid this consists of skipping frames until the average uncertainty of the 3-D points decreases below a certain threshold. The selected frames are called *keyframes*. Keyframe selection is a very important step in VO and should always be done before updating the motion.

VO trades off consistency for real-time performance.

Discussion

According to Nister et al. [1], there is an advantage in using the 2-D-to-2-D and 3-D-to-2-D methods compared to the 3-D-to-3-D method for motion computation. Nister compared the VO performance of the 3-D-to-3-D case to that of the 3-D-to-2-D case for a stereo camera system and found the latter being greatly superior to the former. The reason is due to the triangulated 3-D points being much more uncertain in the depth direction. When 3-D-to-3-D feature correspondences are used in motion computation, their uncertainty may have a devastating effect on the motion estimate. In fact, in the 3-D-to-3-D case, the 3-D position error, (9), is minimized whereas in the 3-D-to-2-D case the image reprojection error, (10).

In the monocular scheme, the 2-D-to-2-D method is preferable compared to the 3-D-to-2-D case since it avoids point triangulation. However, in practice, the 3-D-to-2-D method is used more often than the 2-D-to-2-D method. The reason lies in its faster data association. As will be described in Part II of this tutorial, for accurate motion computation, it is of utmost importance that the input data do not contain outliers. Outlier rejection is a very delicate step, and the computation time of this operation is strictly linked to the minimum number of points necessary to estimate the motion. As mentioned previously, the 2-D-to-2-D case requires a minimum of five-point correspondences (see the five-point algorithm); however, only three correspondences are necessary in the 3-D-to-2-D motion case (see P3P). As will be shown in Part II of this tutorial, this lower number of points results in a much faster motion estimation.

An advantage of the stereo camera scheme compared to the monocular one, besides the property that 3-D features are computed directly in the absolute scale, is that matches need to be computed only between two views instead of three views as in the monocular scheme. Additionally, since the 3-D structure is computed directly from a single stereo pair rather than from adjacent frames as in the monocular case, the stereo scheme exhibits less drift than the monocular one in case of small motions. Monocular methods are interesting because stereo VO degenerates into the monocular case when the distance to the scene is much larger than the stereo baseline (i.e., the distance between the two cameras). In this case, stereo vision becomes ineffective and monocular methods must be used.

Regardless of the chosen motion computation method, local bundle adjustment (over the last m frames) should always be performed to compute a more accurate estimate of the trajectory. After bundle adjustment, the effects of the motion estimation method are much more alleviated.

Conclusions

This tutorial has described the history of VO, the problem formulation, and the distinct approaches to motion computation. VO is a well-understood and established part of robotics. Part II of this tutorial will summarize the remaining building blocks of the VO pipeline: how to detect and match salient and repeatable features across frames, robust estimation in the presence of outliers, and bundle adjustment. In addition, error propagation, applications, and links to free-to-download code will be included.

Acknowledgments

The authors thank Konstantinos Derpanis, Oleg Naroditsky, Carolin Baez, and Andrea Censi for their fruitful comments and suggestions.

References

[1] D. Nister, O. Naroditsky, and J. Bergen, "Visual odometry," in *Proc. Int. Conf. Computer Vision and Pattern Recognition*, 2004, pp. 652–659.

[2] H. Longuet-Higgins, "A computer algorithm for reconstructing a scene from two projections," *Nature*, vol. 293, no. 10, pp. 133–135, 1981.

[3] C. Harris and J. Pike, "3d positional integration from image sequences," in *Proc. Alvey Vision Conf.*, 1988, pp. 87–90.

[4] J.-M. Frahm, P. Georgel, D. Gallup, T. Johnson, R. Raguram, C. Wu, Y.-H. Jen, E. Dunn, B. Clipp, S. Lazebnik, and M. Pollefeys, "Building rome on a cloudless day," in *Proc. European Conf. Computer Vision*, 2010, pp. 368–381.

[5] H. Moravec, "Obstacle avoidance and navigation in the real world by a seeing robot rover," Ph.D. dissertation, Stanford Univ., Stanford, CA, 1980.

[6] L. Matthies and S. Shafer, "Error modeling in stereo navigation," *IEEE J. Robot. Automat.*, vol. 3, no. 3, pp. 239–248, 1987.

[7] L. Matthies, "Dynamic stereo vision," Ph.D. dissertation, Carnegie Mellon Univ., Pittsburgh, PA, 1989.

[8] S. Lacroix, A. Mallet, R. Chatila, and L. Gallo, "Rover self localization in planetary-like environments," in *Proc. Int. Symp. Articial Intelligence, Robotics, and Automation for Space (i-SAIRAS)*, 1999, pp. 433–440.

[9] C. Olson, L. Matthies, M. Schoppers, and M. W. Maimone, "Robust stereo ego-motion for long distance navigation," in *Proc. IEEE Conf. Computer Vision and Pattern Recognition*, 2000, pp. 453–458.

[10] M. Hannah, "Computer matching of areas in stereo images," Ph.D. dissertation, Stanford Univ., Stanford, CA, 1974.
[11] H. Moravec, "Towards automatic visual obstacle avoidance," in *Proc. 5th Int. Joint Conf. Artificial Intelligence*, Aug. 1977, p. 584.
[12] W. Forstner, "A feature based correspondence algorithm for image matching," *Int. Arch. Photogrammetry*, vol. 26, no. 3, pp. 150–166, 1986.
[13] C. Olson, L. Matthies, M. Schoppers, and M. Maimone, "Rover navigation using stereo ego-motion," *Robot. Autonom. Syst.*, vol. 43, no. 4, pp. 215–229, 2003.
[14] A. Milella and R. Siegwart, "Stereo-based ego-motion estimation using pixel tracking and iterative closest point," in *Proc. IEEE Int. Conf. Vision Systems*, pp. 21–24, 2006.
[15] A. Howard, "Real-time stereo visual odometry for autonomous ground vehicles," in *Proc. IEEE/RSJ Int. Conf. Intelligent Robots and Systems*, 2008, pp. 3946–3952.
[16] Y. Cheng, M. W. Maimone, and L. Matthies, "Visual odometry on the mars exploration rovers," *IEEE Robot. Automat. Mag.*, vol. 13, no. 2, pp. 54–62, 2006.
[17] M. Maimone, Y. Cheng, and L. Matthies, "Two years of visual odometry on the mars exploration rovers: Field reports," *J. Field Robot.*, vol. 24, no. 3, pp. 169–186, 2007.
[18] M. A. Fischler and R. C. Bolles, "Random sample consensus: A paradigm for model fitting with applications to image analysis and automated cartography," *Commun. ACM*, vol. 24, no. 6, pp. 381–395, 1981.
[19] C. Tomasi and J. Shi, "Good features to track," in *Proc. Computer Vision and Pattern Recognition (CVPR '94)*, 1994, pp. 593–600.
[20] P. Besl and N. McKay, "A method for registration of 3-d shapes," *IEEE Trans. Pattern Anal. Machine Intell.*, vol. 14, no. 2, pp. 239–256, 1992.
[21] A. Comport, E. Malis, and P. Rives, "Accurate quadrifocal tracking for robust 3d visual odometry," in *Proc. IEEE Int. Conf. Robotics and Automation*, 2007, pp. 40–45.
[22] R. Hartley and A. Zisserman, *Multiple View Geometry in Computer Vision*, 2nd ed. Cambridge U.K.: Cambridge Univ. Press, 2004.
[23] D. Nister, O. Naroditsky, and J. Bergen, "Visual odometry for ground vehicle applications," *J. Field Robot.*, vol. 23, no. 1, pp. 3–20, 2006.
[24] P. I. Corke, D. Strelow, and S. Singh, "Omnidirectional visual odometry for a planetary rover," in *Proc. IEEE/RSJ Int. Conf. Intelligent Robots and Systems*, 2005, pp. 4007–4012.
[25] M. Lhuillier, "Automatic structure and motion using a catadioptric camera," in *Proc. IEEE Workshop Omnidirectional Vision*, 2005, pp. 1–8.
[26] R. Goecke, A. Asthana, N. Pettersson, and L. Petersson, "Visual vehicle egomotion estimation using the Fourier-Mellin transform," in *Proc. IEEE Intelligent Vehicles Symp.*, 2007, pp. 450–455.
[27] J. Tardif, Y. Pavlidis, and K. Daniilidis, "Monocular visual odometry in urban environments using an omnidirectional camera," in *Proc. IEEE/RSJ Int. Conf. Intelligent Robots and Systems*, 2008, pp. 2531–2538.
[28] M. J. Milford and G. Wyeth, "Single camera vision-only SLAM on a suburban road network," in *Proc. IEEE Int. Conf. Robotics and Automation (ICRA '08)*, 2008, pp. 3684–3689.
[29] D. Scaramuzza and R. Siegwart, "Appearance-guided monocular omnidirectional visual odometry for outdoor ground vehicles," *IEEE Trans. Robot. (Special Issue on Visual SLAM)*, vol. 24, no. 5, pp. 1015–1026, Oct. 2008.
[30] E. Mouragnon, M. Lhuillier, M. Dhome, F. Dekeyser, and P. Sayd, "Real time localization and 3d reconstruction," in *Proc. Int. Conf. Computer Vision and Pattern Recognition*, 2006, pp. 363–370.
[31] D. Scaramuzza, F. Fraundorfer, and R. Siegwart, "Real-time monocular visual odometry for on-road vehicles with 1-point RANSAC," in *Proc. IEEE Int. Conf. Robotics and Automation (ICRA '09)*, 2009, pp. 4293–4299.
[32] A. Pretto, E. Menegatti, and E. Pagello, "Omnidirectional dense large-scale mapping and navigation based on meaningful triangulation," in *Proc. IEEE Int. Conf. Robotics and Automation*, 2011, pp. 3289–3296.
[33] D. Nister, "An efficient solution to the five-point relative pose problem," in *Proc. Int. Conf. Computer Vision and Pattern Recognition*, 2003, pp. 195–202.
[34] M. Milford, G. Wyeth, and D. Prasser, "RatSLAM: A hippocampal model for simultaneous localization and mapping," in *Proc. IEEE Int. Conf. Robotics and Automation (ICRA '04)*, 2004, pp. 403–408.
[35] B. Liang and N. Pears, "Visual navigation using planar homographies," in *Proc. IEEE Int. Conf. Robotics and Automation (ICRA '02)*, 2002, pp. 205–210.
[36] Q. Ke and T. Kanade, "Transforming camera geometry to a virtual downward-looking camera: Robust ego-motion estimation and ground-layer detection," in *Proc. Computer Vision and Pattern Recognition (CVPR)*, June 2003, pp. 390–397.
[37] H. Wang, K. Yuan, W. Zou, and Q. Zhou, "Visual odometry based on locally planar ground assumption," in *Proc. IEEE Int. Conf. Information Acquisition*, 2005, pp. 59–64.
[38] J. Guerrero, R. Martinez-Cantin, and C. Sagues, "Visual map-less navigation based on homographies," *J. Robot. Syst.*, vol. 22, no. 10, pp. 569–581, 2005.
[39] D. Scaramuzza, "1-point-RANSAC structure from motion for vehicle-mounted cameras by exploiting non-holonomic constraints," *Int. J. Comput. Vis.*, vol. 95, no. 1, pp. 74–85, 2011.
[40] D. Scaramuzza, F. Fraundorfer, M. Pollefeys, and R. Siegwart, "Absolute scale in structure from motion from a single vehicle mounted camera by exploiting nonholonomic constraints," in *Proc. IEEE Int. Conf. Computer Vision (ICCV)*, Kyoto, Oct. 2009, pp. 1413–1419.
[41] F. Fraundorfer, D. Scaramuzza, and M. Pollefeys, "A constricted bundle adjustment parameterization for relative scale estimation in visual odometry," in *Proc. IEEE Int. Conf. Robotics and Automation*, 2010, pp. 1899–1904.
[42] N. Sunderhauf, K. Konolige, S. Lacroix, and P. Protzel, "Visual odometry using sparse bundle adjustment on an autonomous outdoor vehicle," in *Tagungsband Autonome Mobile Systeme, Reihe Informatik aktuell*. Levi, Schanz, Lafrenz, and Avrutin, Eds. Berlin, Springer-Verlag, 2005, pp. 157–163.
[43] K. Konolige, M. Agrawal, and J. Sol, "Large scale visual odometry for rough terrain," in *Proc. Int. Symp. Robotics Research*, 2007.
[44] J. Tardif, M. G. M. Laverne, A. Kelly, and M. Laverne, "A new approach to vision-aided inertial navigation," in *Proc. IEEE/RSJ Int. Conf. Intelligent Robots and Systems*, 2010, pp. 4161–4168.
[45] A. I. Mourikis and S. Roumeliotis, "A multi-state constraint kalman filter for vision-aided inertial navigation," in *Proc. IEEE Int. Conf. Robotics and Automation*, 2007, pp. 3565–3572.
[46] E. Jones and S. Soatto, "Visual-inertial navigation, mapping and localization: A scalable real-time causal approach," *Int. J. Robot. Res.*, vol. 30, no. 4, pp. 407–430, 2010.
[47] H. Durrant-Whyte and T. Bailey, "Simultaneous localization and mapping (SLAM): Part I. The essential algorithms," *Robot. Automat. Mag.*, vol. 13, no. 2, pp. 99–110, 2006.

[48] T. Bailey and H. Durrant-Whyte, "Simultaneous localisation and mapping (SLAM): Part II. State of the art," *Robot. Automat. Mag.*, vol. 13, no. 3, pp. 108–117, 2006.

[49] A. Davison, "Real-time simultaneous localisation and mapping with a single camera," in *Proc. Int. Conf. Computer Vision*, 2003, pp. 1403–1410.

[50] G. Klein and D. Murray, "Parallel tracking and mapping for small ar workspaces," in *Proc. Int. Symp. Mixed and Augmented Reality*, 2007, pp. 225–234.

[51] H. Strasdat, J. Montiel, and A. Davison, "Real time monocular SLAM: Why filter?" in *Proc. IEEE Int. Conf. Robotics and Automation*, 2010, pp. 2657–2664.

[52] A. Chiuso, P. Favaro, H. Jin, and S. Soatto, "3-D motion and structure from 2-D motion causally integrated over time: Implementation," in *Proc. European Conf. Computer Vision*, 2000, pp. 734–750.

[53] M. C. Deans, "Bearing-only localization and mapping," Ph.D. dissertation, Carnegie Mellon Univ., Pittsburgh, 2002.

[54] L. A. Clemente, A. J. Davison, I. Reid, J. Neira, and J. D. Tardos, "Mapping large loops with a single hand-held camera," in *Proc. Robotics Science and Systems*, 2007.

[55] T. Lemaire and S. Lacroix, "Vision-based SLAM: Stereo and monocular approaches," *Int. J. Computer Vision*, vol. 74, no. 3, pp. 343–364, 2006.

[56] E. Eade and T. Drummond, "Monocular SLAM as a graph of coalesced observations," in *Proc. IEEE Int. Conf. Computer Vision*, 2007, pp. 1–8.

[57] G. Klein and D. Murray, "Improving the agility of keyframe-based SLAM," in *Proc. European Conf. Computer Vision*, 2008, pp. 802–815.

[58] K. Konolige and M. Agrawal, "FrameSLAM: From bundle adjustment to real-time visual mappping," *IEEE Trans. Robot.*, vol. 24, no. 5, pp. 1066–1077, 2008.

[59] A. Handa, M. Chli, H. Strasdat, and A. J. Davison, "Scalable active matching," in *Proc. Int. Conf. Computer Vision and Pattern Recognition*, 2010, pp. 1546–1553.

[60] J. Civera, O. Grasa, A. Davison, and J. Montiel, "1-point RANSAC for ekf filtering: Application to real-time structure from motion and visual odometry," *J. Field Robot.*, vol. 27, no. 5, pp. 609–631, 2010.

[61] H. Strasdat, J. Montiel, and A. J. Davison, "Scale drift-aware large scale monocular SLAM," in *Proc. Robotics Science and Systems*, 2010.

[62] C. Mei, G. Sibley, M. Cummins, P. Newman, and I. Reid, "RSLAM: A system for large-scale mapping in constant-time using stereo," *Int. J. Computer Vision*, vol. 94, no. 2, pp. 198–214, 2010.

[63] Y. Ma, S. Soatto, J. Kosecka, and S. Sastry, *An Invitation to 3D Vision, from Images to Models*. Berlin: Springer-Verlag, 2003.

[64] C. Geyer and K. Daniilidis, "A unifying theory for central panoramic systems and practical applications," in *Proc. European Conf. Computer Vision*, 2000, pp. 445–461.

[65] D. Scaramuzza, A. Martinelli, and R. Siegwart, "A flexible technique for accurate omnidirectional camera calibration and structure from motion," in *Proc. IEEE Int. Conf. Computer Vision Systems (ICVS) 2006*, Jan. 2006, pp. 45–53.

[66] D. Scaramuzza, "Omnidirectional vision: From calibration to robot motion estimation" Ph.D. dissertation, ETH Zurich, 2008.

[67] D. Scaramuzza, "Omnidirectional camera," in *Encyclopedia of Computer Vision*, K. Ikeuchi, Ed. Berlin: Springer-Verlag, 2012.

[68] J. Bouguet (2011). List of camera calibration toolboxes. [Online]. Available: http://www.vision.caltech.edu/bouguetj/calib.doc/htmls/links.html

[69] J. Bouguet. Camera calibration toolbox for Matlab. [Online]. Available: http://www.vision.caltech.edu/bouguetj/calib.doc/index.html

[70] D. Scaramuzza. (2006). Ocamcalib toolbox: Omnidirectional camera calibration toolbox for Matlab (uses planar grids) google for "*ocamcalib*." [Online]. Available: https://sites.google.com/site/scarabotix/ocamcalib-toolbox

[71] C. Mei. (2006). Omnidirectional camera calibration toolbox for Matlab (uses planar grids). [Online]. Available: http://homepages.laas.fr/~cmei/index.php/Toolbox

[72] J. Barreto. Omnidirectional camera calibration toolbox for Matlab (uses lines). [Online]. Available: http://www.isr.uc.pt/~jpbar/CatPack/pag1.htm

[73] Opencv: Open-source computer vision library. [Online]. Available: http://opencv.willowgarage.com/wiki/

[74] T. Huang and A. Netravalli, "Motion and structure from feature correspondences: A review," *Proc. IEEE*, vol. 82, no. 2, pp. 252–268, 1994.

[75] E. Kruppa, "Zur ermittlung eines objektes aus zwei perspektiven mit innerer orientierung," *Sitzungsberichte der Akademie der Wissenschaften, Wien, Mathematisch-Naturwissenschaftlichen Klasse, Abteilung IIa*, vol. 122, pp. 1939–1948, 1913.

[76] D. Nister, "An efficient solution to the five-point relative pose problem," in *Proc. Computer Vision and Pattern Recognition (CVPR '03)*, 2003, pp. II: 195–202.

[77] H. Goldstein, *Classical Mechanics*. Reading, MA: Addison-Wesley, 1981.

[78] K. S. Arun, T. S. Huang, and S. D. Blostein, "Least-squares fitting of two 3-d point sets," *IEEE Trans. Pattern Anal. Machine Intell.*, vol. 9, no. 5, pp. 698–700, 1987.

[79] F. Moreno-Noguer, V. Lepetit, and P. Fua, "Accurate non-iterative O(n) solution to the PnP problem," in *Proc. IEEE Int. Conf. Computer Vision*, 2007, pp. 1–8.

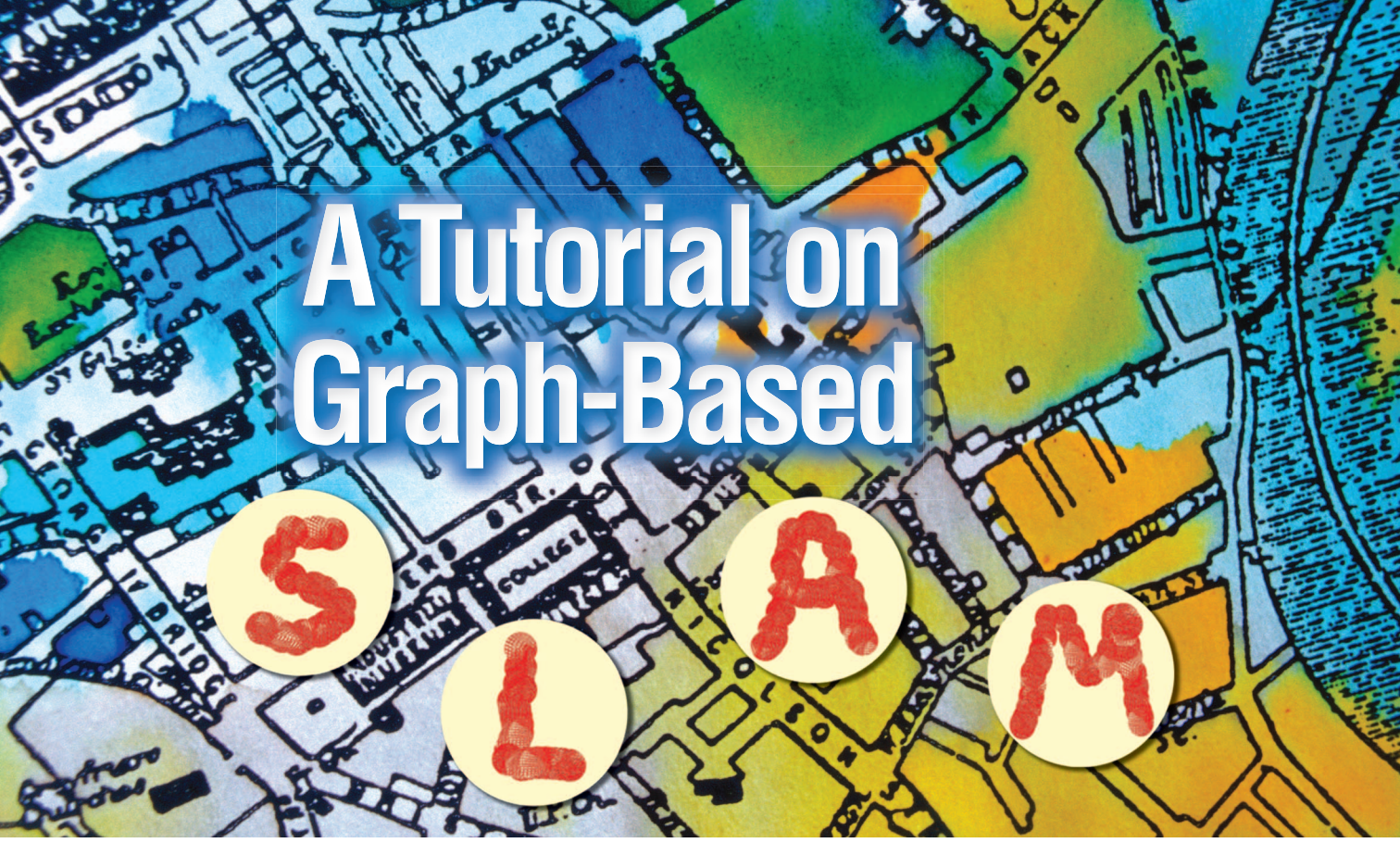
[80] L. Kneip, D. Scaramuzza, and R. Siegwart, "A novel parameterization of the perspective-three-point problem for a direct computation of absolute camera position and orientation," in *Proc. IEEE Conf. Computer Vision and Pattern Recognition*, 2011, pp. 2969–2976.

[81] D. Nister, "A minimal solution to the generalised 3-point pose problem," in *Proc. IEEE Conf. Computer Vision and Pattern Recognition*, 2004, pp. 560–567.

[82] C. Harris and M. Stephens, "A combined corner and edge detector," in *Proc. Alvey Vision Conf.*, 1988, pp. 147–151.

Davide Scaramuzza, GRASP Lab, Department of Computer and Information Science, School of Engineering and Applied Science, University of Pennsylvania, Philadelphia. E-mail: davide.scaramuzza@ieee.org.

Friedrich Fraundorfer, Computer Vision and Geometry Lab, Institute of Visual Computing, Department of Computer Science, ETH Zurich, Switzerland. E-mail: fraundorfer@inf.ethz.ch.



Giorgio Grisetti,
Rainer Kümmerle,
Cyrill Stachniss, and
Wolfram Burgard

***Abstract*—**Being able to build a map of the environment and to simultaneously localize within this map is an essential skill for mobile robots navigating in unknown environments in absence of external referencing systems such as GPS. This so-called simultaneous localization and mapping (SLAM) problem has been one of the most popular research topics in mobile robotics for the last two decades and efficient approaches for solving this task have been proposed. One intuitive way of formulating SLAM is to use a graph whose nodes correspond to the poses of the robot at different points in time and whose edges represent constraints between the poses. The latter are obtained from observations of the environment or from movement actions carried out by the robot. Once such a graph is constructed, the map can be computed by finding the spatial configuration of the nodes that is mostly consistent with the measurements modeled by the edges. In this paper, we provide an introductory description to the graph-based SLAM problem. Furthermore, we discuss a state-of-the-art solution that is based on least-squares error minimization and exploits the structure of the SLAM problems during optimization. The goal of this tutorial is to enable the reader to implement the proposed methods from scratch.

I. Introduction

To efficiently solve many tasks envisioned to be carried out by mobile robots including transportation, search and rescue, or automated vacuum cleaning robots need a map of the environment. The availability of an accurate map allows for the design of systems that can operate in complex environments only based on their on-board sensors and without relying on external reference system like, e.g., GPS. The acquisition of maps of indoor environments, where typically no GPS is available, has been a major research focus in the robotics community over the last decades. Learning maps under pose uncertainty is often referred to as the simultaneous localization and mapping (SLAM) problem. In the literature, a large variety of solutions to this problem is available. These approaches can be classified either as filtering or smoothing. Filtering approaches model the problem as an on-line state estimation where the state of the

Digital Object Identifier 10.1109/MITS.2010.939925
Date of publication: 4 February 2011

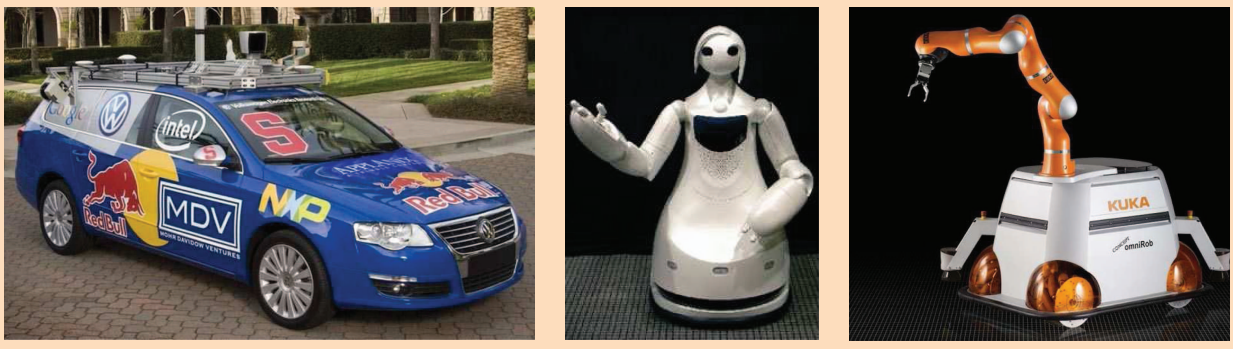
Graph-based SLAM methods have undergone a renaissance and currently belong to the state-of-the-art techniques with respect to speed and accuracy.

system consists in the current robot position and the map. The estimate is augmented and refined by incorporating the new measurements as they become available. Popular techniques like Kalman and information filters [28], [3], particle filters [22], [12], [9], or information filters [7], [31] fall into this category. To highlight their incremental nature, the filtering approaches are usually referred to as on-line SLAM methods. Conversely, smoothing approaches estimate the full trajectory of the robot from the full set of measurements [21], [5], [27]. These approaches address the so-called full SLAM problem, and they typically rely on least-square error minimization techniques.

Figure 1 shows three examples of real robotic systems that use SLAM technology: an autonomous car, a tour-guide robot, and an industrial mobile manipulation robot. Image (a) shows the autonomous car Junior as well as a model of a parking garage that has been mapped with that car. Thanks to the acquired model, the car is able to park itself autonomously at user selected locations in the garage. Image (b) shows the TPR-Robina robot developed by Toyota which is also used in the context of guided tours in museums. This robot uses SLAM technology to update its map whenever the environment has been changed. Robot manufacturers such as KUKA, recently presented mobile manipulators as shown in Image (c). Here, SLAM technology is needed to operate such devices in flexible way in changing industrial environments. Figure 2 illustrates 2D and 3D maps that can be estimated by the SLAM algorithm discussed in this paper.

An intuitive way to address the SLAM problem is via its so-called graph-based formulation. Solving a graph-based SLAM problem involves to construct a graph whose nodes represent robot poses or landmarks and in which an edge between two nodes encodes a sensor measurement that constrains the connected poses. Obviously, such constraints can be contradictory since observations are always affected by noise. Once such a graph is constructed, the crucial problem is to find a configuration of the nodes that is maximally consistent with the measurements. This involves solving a large error minimization problem.

The graph-based formulation of the SLAM problem has been proposed by Lu and Milios in 1997 [21]. However, it took several years to make this formulation popular due to the comparably high complexity of solving the error minimization problem using standard techniques. Recent insights into the structure of the SLAM problem and advancements in the fields of sparse linear algebra resulted in efficient approaches to the optimization problem at hand. Consequently, graph-based SLAM methods have undergone a renaissance and currently belong to the state-of-the-art techniques with respect to speed and accuracy. The aim of this tutorial is to introduce the SLAM problem in its probabilistic form and to guide the reader to the synthesis of an effective and state-of-the-art graph-based SLAM method. To understand this tutorial a good knowledge of linear algebra, multivariate minimization, and probability theory are required.



(a) (b) (c)

FIG 1 Applications of SLAM technology. (a) An autonomous instrumented car developed at Stanford. This car can acquire maps by utilizing only its on-board sensors. These maps can be subsequently used for autonomous navigation. (b) The museum guide robot TPR-Robina developed by Toyota (picture courtesy of Toyota Motor Company). This robot acquires a new map every time the museum is reconfigured. (c) The KUKA Concept robot "Omnirob", a mobile manipulator designed autonomously navigate and operate in the environment with the sole use of its on-board sensors (picture courtesy of KUKA Roboter GmbH).

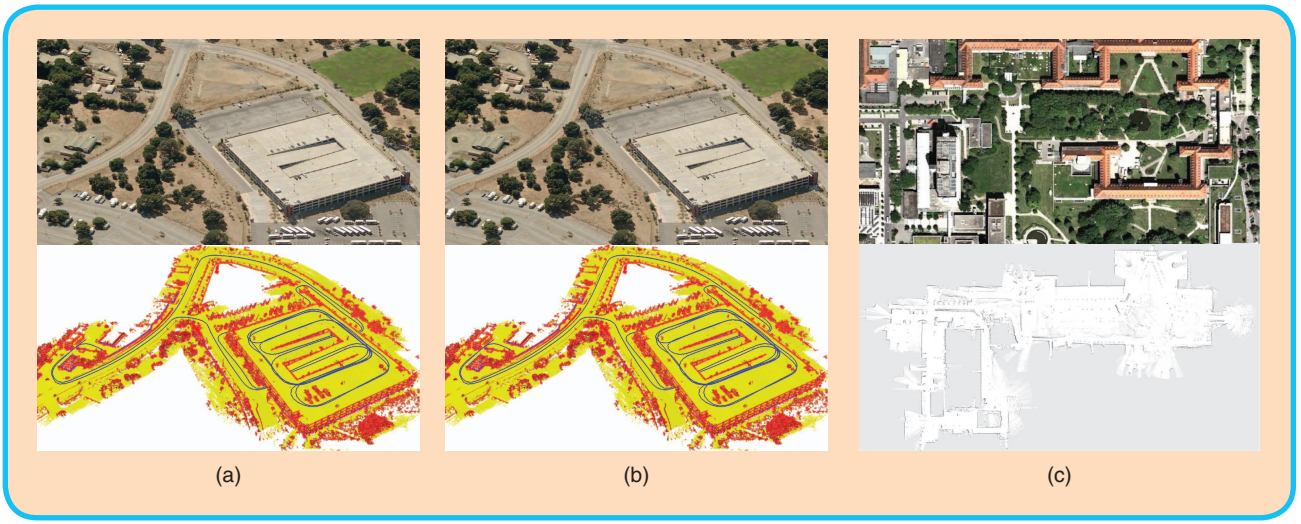


FIG 2 (a) A 3D map of the Stanford parking garage acquired with an instrumented car (bottom), and the corresponding satellite view (top). This map has been subsequently used to realize an autonomous parking behavior. (b) Point cloud map acquired at the university of Freiburg (courtesy of Kai. M. Wurm) and relative satellite image. (c) Occupancy grid map acquired at the hospital of Freiburg. Top: a bird's eye view of the area, bottom: the occupancy grid representation. The gray areas represent unobserved regions, the white part represents traversable space while the black points indicate occupied regions.

II. Probabilistic Formulation of SLAM

Solving the SLAM problem consists of estimating the robot trajectory and the map of the environment as the robot moves in it. Due to the inherent noise in the sensor measurements, a SLAM problem is usually described by means of probabilistic tools. The robot is assumed to move in an unknown environment, along a trajectory described by the sequence of random variables $\mathbf{x}_{1:T} = \{\mathbf{x}_1, \ldots, \mathbf{x}_T\}$. While moving, it acquires a sequence of odometry measurements $\mathbf{u}_{1:T} = \{\mathbf{u}_1, \ldots, \mathbf{u}_T\}$ and perceptions of the environment $\mathbf{z}_{1:T} = \{\mathbf{z}_1, \ldots, \mathbf{z}_T\}$. Solving the full SLAM problem consists of estimating the posterior probability of the robot's trajectory $\mathbf{x}_{1:T}$ and the map $\mathbf{m}$ of the environment given all the measurements plus an initial position $\mathbf{x}_0$:

$$p(\mathbf{x}_{1:T}, \mathbf{m} \mid \mathbf{z}_{1:T}, \mathbf{u}_{1:T}, \mathbf{x}_0). \tag{1}$$

The initial position $\mathbf{x}_0$ defines the position of the map and can be chosen arbitrarily. For convenience of notation, in the remainder of this document we will omit $\mathbf{x}_0$. The poses $\mathbf{x}_{1:T}$ and the odometry $\mathbf{u}_{1:T}$ are usually represented as 2D or 3D transformations in $SE(2)$ or in $SE(3)$, while the map can be represented in different ways. Maps can be parametrized as a set of spatially located landmarks, by dense representations like occupancy grids, surface maps, or by raw sensor measurements. The choice of a particular map representation depends on the sensors used, on the characteristics of the environment, and on the estimation algorithm. Landmark maps [28], [22] are often preferred in environments where locally distinguishable features can be identified and especially when cameras are used. In contrast, dense representations [33], [12], [9] are usually used in conjunction with range sensors. Independently of the type of the representation, the map is defined by the measurements and the locations where these measurements have been acquired [17], [18]. Figure 2 illustrates three typical dense map representations for 3D and 2D: multilevel surface maps, point clouds and occupancy grids. Figure 3 shows a typical 2D landmark based map.

Estimating the posterior given in (1) involves operating in high dimensional state spaces. This would not be tractable if the SLAM problem would not have a well defined structure. This structure arises from certain and commonly done assumptions, namely the static world

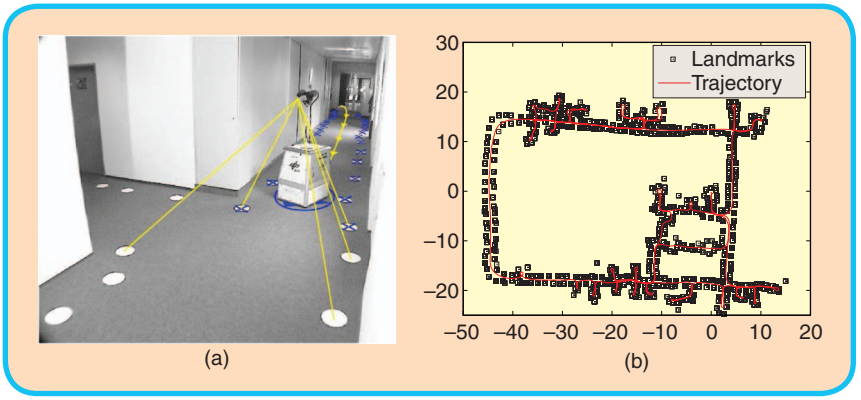


FIG 3 Landmark based maps acquired at the German Aerospace Center. In this setup the landmarks consist in white circles painted on the ground that are detected by the robot through vision, as shown in the left image. The right image illustrates the trajectory of the robot and the estimated positions of the landmarks. These images are courtesy of Udo Frese and Christoph Hertzberg.

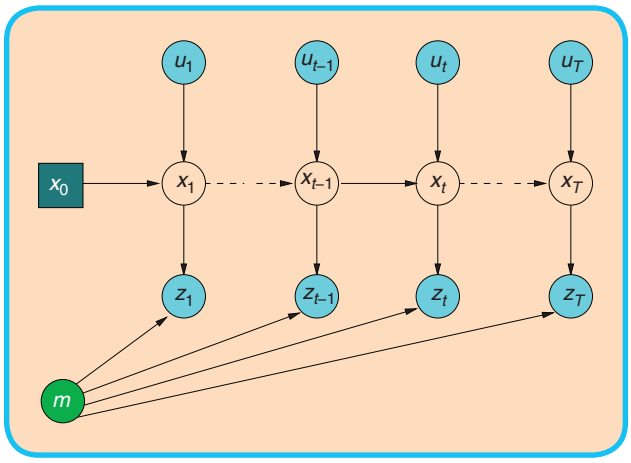


FIG 4 Dynamic Bayesian Network of the SLAM process.

assumption and the Markov assumption. A convenient way to describe this structure is via the dynamic Bayesian network (DBN) depicted in Figure 4. A Bayesian network is a graphical model that describes a stochastic process as a directed graph. The graph has one node for each random variable in the process, and a directed edge (or arrow) between two nodes models a conditional dependence between them.

In Figure 4, one can distinguish blue/gray nodes indicating the observed variables (here $\mathbf{z}_{1:T}$ and $\mathbf{u}_{1:T}$) and white nodes which are the hidden variables. The hidden variables $\mathbf{x}_{1:T}$ and $\mathbf{m}$ model the robot's trajectory and the map of the environment. The connectivity of the DBN follows a recurrent pattern characterized by the state transition model and by the observation model. The transition model $p(\mathbf{x}_t | \mathbf{x}_{t-1}, \mathbf{u}_t)$ is represented by the two edges leading to $\mathbf{x}_t$ and represents the probability that the robot at time t is in $\mathbf{x}_t$ given that at time $t-1$ it was in $\mathbf{x}_t$ and it acquired an odometry measurement $\mathbf{u}_t$.

The observation model $p(\mathbf{z}_t | \mathbf{x}_t, \mathbf{m}_t)$ models the probability of performing the observation $\mathbf{z}_t$ given that the robot is at location $\mathbf{x}_t$ in the map. It is represented by the arrows entering in $\mathbf{z}_t$. The exteroceptive observation $\mathbf{z}_t$ depends only on the current location $\mathbf{x}_t$ of the robot and on the (static) map $\mathbf{m}$. Expressing SLAM as a DBN highlights its temporal structure, and therefore this formalism is well suited to describe filtering processes that can be used to tackle the SLAM problem.

An alternative representation to the DBN is via the so-called "graph-based" or "network-based" formulation of the SLAM problem, that highlights the underlying spatial structure. In graph-based SLAM, the poses of the robot are modeled by nodes in a graph and labeled with their position in the environment [21], [18]. Spatial constraints between poses that result from observations z_t or from odometry measurements $\mathbf{u}_t$ are encoded in the edges between the nodes. More in detail, a graph-based SLAM algorithm constructs a graph out of the raw sensor measurements. Each node in the graph represents a robot position and a measurement acquired at that position. An edge between two nodes represents a spatial constraint relating the two robot poses. A constraint consists in a probability distribution over the relative transformations between the two poses. These transformations are either odometry measurements between sequential robot positions or are determined by aligning the observations acquired at the two robot locations. Once the graph is constructed one seeks to find the configuration of the robot poses that best satisfies the constraints. Thus, in graph-based SLAM the problem is decoupled in two tasks: constructing the graph from the raw measurements (graph construction), determining the most likely configuration of the poses given the edges of the graph (graph optimization). The graph construction is usually called front-end and it is heavily sensor dependent, while the second part is called back-end and relies on an abstract representation of the data which is sensor agnostic. A short example of a front-end for 2D laser SLAM is described in Section A. In this tutorial we will describe an easy-to-implement but efficient back-end for graph-based SLAM. Figure 5 depicts an uncorrected pose-graph and the corresponding corrected one.

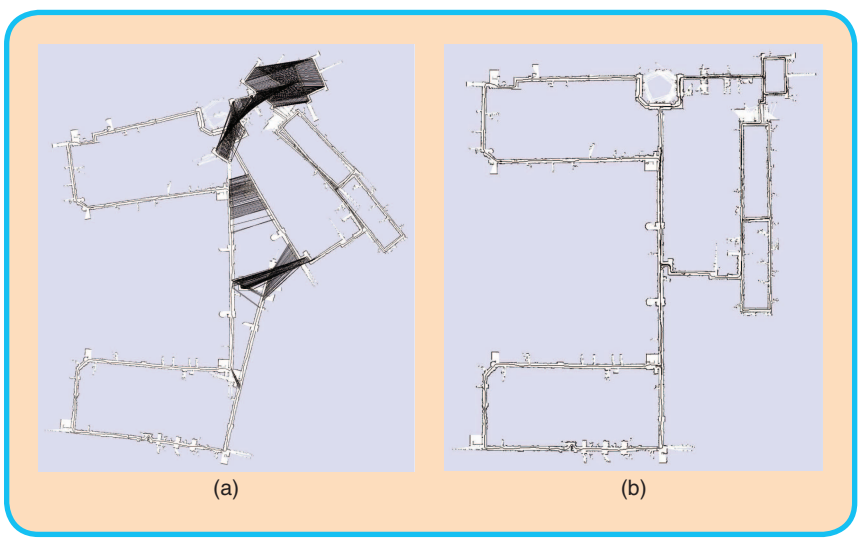


FIG 5 Pose-graph corresponding to a data-set recorded at MIT Killian Court (courtesy of Mike Bosse and John Leonard) (left) and after (right) optimization. The maps are obtained by rendering the laser scans according to the robot positions in the graph.

III. Related Work

There is a large variety of SLAM approaches available in the robotics community. Throughout this tutorial we

focus on graph-based approaches and therefore will consider such approaches in the discussion of related work. Lu and Milios [21] were the first to refine a map by globally optimizing the system of equations to reduce the error introduced by constraints. Gutmann and Konolige [11] proposed an effective way for constructing such a network and for detecting loop closures while running an incremental estimation algorithm. Since then, many approaches for minimizing the error in the constraint network have been proposed. For example, Howard *et al.* [15] apply relaxation to localize the robot and build a map. Frese *et al.* [8] propose a variant of Gauss-Seidel relaxation called multi-level relaxation (MLR). It applies relaxation at different resolutions. Dellaert and Kaess [5] were the first to exploit sparse matrix factorizations to solve the linearized problem in off-line SLAM. Subsequently Kaess *et al.* [16] presented iSAM, an online version that exploits partial reorderings to compute the sparse factorization.

Recently, Konolige *et al.* [19] proposed an open-source implementation of a pose-graph method that constructs the linearized system in an efficient way. Olson *et al.* [27] presented an efficient optimization approach which is based on the stochastic gradient descent and can efficiently correct even large pose-graphs. Grisetti *et al.* proposed an extension of Olson's approach that uses a tree parametrization of the nodes in 2D and 3D. In this way, they increase the convergence speed [10].

GraphSLAM [32] applies variable elimination techniques to reduce the dimensionality of the optimization problem. The ATLAS framework [2] constructs a two-level hierarchy of graphs and employs a Kalman filter to construct the bottom level. Then, a global optimization approach aligns the local maps at the second level. Similar to ATLAS, Estrada *et al.* proposed Hierarchical SLAM [6] as a technique for using independent local maps.

Most optimization techniques focus on computing the best map given the constraints and are called SLAM back-ends. In contrast to that, SLAM front-ends seek to interpret the sensor data to obtain the constraints that are the basis for the optimization approaches. Olson [25], for example, presented a front-end with outlier rejection based on spectral clustering. For making data associations in the SLAM front-ends statistical tests such as the χ^2 test or joint compatibility test [23] are often applied. The work of Nüchter et al. [24] aims at building an integrated SLAM system for 3D mapping. The main focus lies on the SLAM front-end for finding constraints. For optimization, a variant of the approach of Lu and Milios [21] for 3D settings is applied. The methods proposed in this paper can be effectively applied to all these front-ends.

IV. Graph-Based SLAM

A graph-based SLAM approach constructs a simplified estimation problem by abstracting the raw sensor measurements. These raw measurements are replaced by the edges

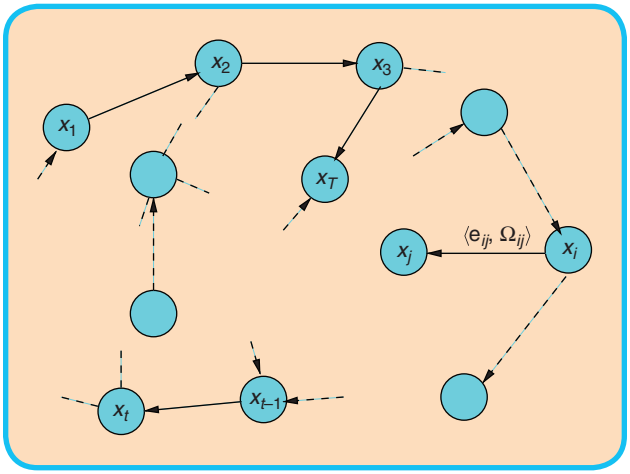


FIG 6 A pose-graph representation of a SLAM process. Every node in the graph corresponds to a robot pose. Nearby poses are connected by edges that model spatial constraints between robot poses arising from measurements. Edges $\mathbf{e}_{t-1\,t}$ between consecutive poses model odometry measurements, while the other edges represent spatial constraints arising from multiple observations of the same part of the environment.

in the graph which can then be seen as "virtual measurements". More in detail an edge between two nodes is labeled with a probability distribution over the relative locations of the two poses, conditioned to their mutual measurements. In general, the observation model $p(\mathbf{z}_t | \mathbf{x}_t, \mathbf{m}_t)$ is multimodal and therefore the Gaussian assumption does not hold. This means that a single observation $\mathbf{z}_t$ might result in multiple potential edges connecting different poses in the graph and the graph connectivity needs itself to be described as a probability distribution. Directly dealing with this multi-modality in the estimation process would lead to a combinatorial explosion of the complexity. As a result of that, most practical approaches restrict the estimate to the most likely topology. Thus, one needs to determine the most likely constraint resulting from an observation. This decision depends on the probability distribution over the robot poses. This problem is known as data association and is usually addressed by the SLAM front-end. To compute the correct data-association, a front-end usually requires a consistent estimate of the conditional prior over the robot trajectory $p(\mathbf{x}_{1:T} | \mathbf{z}_{1:T}, \mathbf{u}_{1:T})$. This requires to interleave the execution of the front-end and of the back-end while the robot explores the environment. Therefore, the accuracy and the efficiency of the back-end is crucial to the design of a good SLAM system. In this tutorial, we will not describe sophisticated approaches to the data association problem. Such methods tackle association by means of spectral clustering [27], joint compatibility branch and bound [23], or backtracking [13]. We rather assume that the given front-end provides consistent estimates.

If the observations are affected by (locally) Gaussian noise and the data association is known, the goal of a graph-based mapping algorithm is to compute a Gaussian approximation

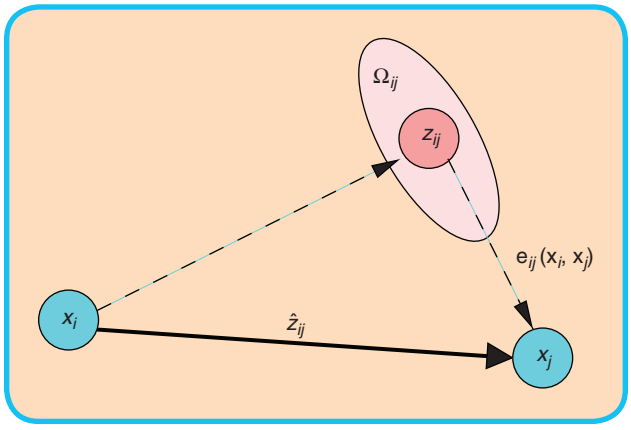


FIG 7 Aspects of an edge connecting the vertex $\mathbf{x}_i$ and the vertex $\mathbf{x}_j$. This edge originates from the measurement $\mathbf{z}_{ij}$. From the relative position of the two nodes, it is possible to compute the expected measurement $\hat{\mathbf{z}}_{ij}$ that represents $\mathbf{x}_j$ seen in the frame of $\mathbf{x}_i$. The error $\mathbf{e}_{ij}(\mathbf{x}_i, \mathbf{x}_j)$ depends on the displacement between the expected and the real measurement. An edge is fully characterized by its error function $\mathbf{e}_{ij}(\mathbf{x}_i, \mathbf{x}_j)$ and by the information matrix $\mathbf{\Omega}_{ij}$ of the measurement that accounts for its uncertainty.

of the posterior over the robot trajectory. This involves computing the mean of this Gaussian as the configuration of the nodes that maximizes the likelihood of the observations. Once this mean is known the information matrix of the Gaussian can be obtained in a straightforward fashion, as explained in Section IV-B. In the following we will characterize the task of finding this maximum as a constraint optimization problem. We will also introduce parts of the notation illustrated in Figure 6.

Let $\mathbf{x} = (\mathbf{x}_1, \ldots, \mathbf{x}_T)^T$ be a vector of parameters, where $\mathbf{x}_i$ describes the pose of node i. Let $\mathbf{z}_{ij}$ and $\mathbf{\Omega}_{ij}$ be respectively the mean and the information matrix of a virtual measurement between the node i and the node j. This virtual measurement is a transformation that makes the observations acquired from i maximally overlap with the observation acquired from j. Let $\hat{\mathbf{z}}_{ij}(\mathbf{x}_i, \mathbf{x}_j)$ be the prediction of a virtual measurement given a configuration of the nodes $\mathbf{x}_i$ and $\mathbf{x}_j$. Usually this prediction is the relative transformation between the two nodes. The log-likelihood l_{ij} of a measurement $\mathbf{z}_{ij}$ is therefore

$$\mathrm{l}_{ij} \propto [\mathbf{z}_{ij} - \hat{\mathbf{z}}_{ij}(\mathbf{x}_i, \mathbf{x}_j)]^T \mathbf{\Omega}_{ij} [\mathbf{z}_{ij} - \hat{\mathbf{z}}_{ij}(\mathbf{x}_i, \mathbf{x}_j)]. \tag{2}$$

Let $\mathbf{e}(\mathbf{x}_i, \mathbf{x}_j, \mathbf{z}_{ij})$ be a function that computes a difference between the expected observation $\hat{\mathbf{z}}_{ij}$ and the real observation $\mathbf{z}_{ij}$ gathered by the robot. For simplicity of notation, we will encode the indices of the measurement in the indices of the error function

$$\mathbf{e}_{ij}(\mathbf{x}_i, \mathbf{x}_j) = \mathbf{z}_{ij} - \hat{\mathbf{z}}_{ij}(\mathbf{x}_i, \mathbf{x}_j). \tag{3}$$

Figure 7 illustrates the functions and the quantities that play a role in defining an edge of the graph. Let $\mathcal{C}$ be the set of pairs of indices for which a constraint (observation) $\mathbf{z}$ exists.

The goal of a maximum likelihood approach is to find the configuration of the nodes $\mathbf{x}^*$ that minimizes the negative log likelihood $\mathbf{F}(\mathbf{x})$ of all the observations

$$\mathbf{F}(\mathbf{x}) = \sum_{\langle i,j \rangle \in \mathcal{C}} \underbrace{\mathbf{e}_{ij}^T \mathbf{\Omega}_{ij} \mathbf{e}_{ij}}_{\mathbf{F}_{ij}} \tag{4}$$

thus, it seeks to solve the following equation:

$$\mathbf{x}^* = \operatorname*{argmin}_{\mathbf{x}} \mathbf{F}(\mathbf{x}). \tag{5}$$

In the remainder of this section we will describe an approach to solve Eq. 5 and to compute a Gaussian approximation of the posterior over the robot trajectory. Whereas the proposed approach utilizes standard optimization methods, like the Gauss-Newton or the Levenberg-Marquardt algorithms, it is particularly efficient because it effectively exploits the structure of the problem.

We first describe a direct implementation of traditional nonlinear least-squares optimization. Subsequently, we introduce a workaround that allows to deal with the singularities in the representation of the robot poses in an elegant manner.

A. Error Minimization via Iterative Local Linearizations

If a good initial guess $\breve{\mathbf{x}}$ of the robot's poses is known, the numerical solution of Eq. (5) can be obtained by using the popular Gauss-Newton or Levenberg-Marquardt algorithms. The idea is to approximate the error function by its first order Taylor expansion around the current initial guess $\breve{\mathbf{x}}$

$$\mathbf{e}_{ij}(\breve{\mathbf{x}}_i + \Delta\mathbf{x}_i, \breve{\mathbf{x}}_j + \Delta\mathbf{x}_j) = \mathbf{e}_{ij}(\breve{\mathbf{x}} + \Delta\mathbf{x}) \tag{6}$$

$$\simeq \mathbf{e}_{ij} + \mathbf{J}_{ij}\Delta\mathbf{x}. \tag{7}$$

Here, $\mathbf{J}_{ij}$ is the Jacobian of $\mathbf{e}_{ij}(\mathbf{x})$ computed in $\breve{\mathbf{x}}$ and $\mathbf{e}_{ij} \overset{\text{def.}}{=} \mathbf{e}_{ij}(\breve{\mathbf{x}})$. Substituting Eq. (7) in the error terms $\mathbf{F}_{ij}$ of Eq. (4), we obtain:

$$\mathbf{F}_{ij}(\breve{\mathbf{x}} + \Delta\mathbf{x})$$

$$= e_{ij}(\breve{\mathbf{x}} + \Delta\mathbf{x})^T \mathbf{\Omega}_{ij} \mathbf{e}_{ij}(\breve{\mathbf{x}} + \Delta\mathbf{x}) \tag{8}$$

$$\simeq (\mathbf{e}_{ij} + \mathbf{J}_{ij}\Delta\mathbf{x})^T \mathbf{\Omega}_{ij} (\mathbf{e}_{ij} + \mathbf{J}_{ij}\Delta\mathbf{x}) \tag{9}$$

$$= \underbrace{\mathbf{e}_{ij}^T \mathbf{\Omega}_{ij} \mathbf{e}_{ij}}_{\mathrm{c}_{ij}} + 2\underbrace{\mathbf{e}_{ij}^T \mathbf{\Omega}_{ij} \mathbf{J}_{ij}}_{\mathbf{b}_{ij}} \Delta\mathbf{x} + \Delta\mathbf{x}^T \underbrace{\mathbf{J}_{ij}^T \mathbf{\Omega}_{ij} \mathbf{J}_{ij}}_{\mathbf{H}_{ij}} \Delta\mathbf{x} \tag{10}$$

$$= \mathrm{c}_{ij} + 2\mathbf{b}_{ij}\Delta\mathbf{x} + \Delta\mathbf{x}^T \mathbf{H}_{ij}\Delta\mathbf{x} \tag{11}$$

With this local approximation, we can rewrite the function $\mathbf{F}(\mathbf{x})$ in Eq. (4) as

$$\mathbf{F}(\breve{\mathbf{x}} + \Delta\mathbf{x}) = \sum_{\langle i,j \rangle \in \mathcal{C}} \mathbf{F}_{ij}(\breve{\mathbf{x}} + \Delta\mathbf{x}) \tag{12}$$

$$\simeq \sum_{\langle i,j\rangle \in \mathcal{C}} c_{ij} + 2\mathbf{b}_{ij}\Delta\mathbf{x} + \Delta\mathbf{x}^T\mathbf{H}_{ij}\Delta\mathbf{x} \quad (13)$$

$$= c + 2\mathbf{b}^T\Delta\mathbf{x} + \Delta\mathbf{x}^T\mathbf{H}\Delta\mathbf{x}. \quad (14)$$

The quadratic form in Eq. (14) is obtained from Eq. (13) by setting $c = \sum c_{ij}$, $\mathbf{b} = \sum \mathbf{b}_{ij}$, and $\mathbf{H} = \sum \mathbf{H}_{ij}$. It can be minimized in $\Delta\mathbf{x}$ by solving the linear system

$$\mathbf{H}\,\Delta\mathbf{x}^* = -\mathbf{b}. \quad (15)$$

The matrix $\mathbf{H}$ is the information matrix of the system, since it is obtained by projecting the measurement error in the space of the trajectories via the Jacobians. It is sparse by construction, having non-zeros between poses connected by a constraint. Its number of non-zero blocks is twice the number of constrains plus the number of nodes. This allows to solve Eq. (15) by sparse Cholesky factorization. An efficient yet compact implementation of sparse Cholesky factorization can be found in the library CSparse [4].

The linearized solution is then obtained by adding to the initial guess the computed increments

$$\mathbf{x}^* = \check{\mathbf{x}} + \Delta\mathbf{x}^*. \quad (16)$$

The popular Gauss-Newton algorithm iterates the linearization in Eq. (14), the solution in Eq. (15), and the update step in Eq. (16). In every iteration, the previous solution is used as the linearization point and the initial guess.

The procedure described above is a general approach to multivariate function minimization, here derived for the special case of the SLAM problem. The general approach, however, assumes that the space of parameters $\mathbf{x}$ is Euclidean, which is not valid for SLAM and may lead to sub-optimal solutions.

B. Considerations about the Structure of the Linearized System

According to Eq. (14), the matrix $\mathbf{H}$ and the vector $\mathbf{b}$ are obtained by summing up a set of matrices and vectors, one for every constraint. Every constraint will contribute to the system with an addend term. The *structure* of this addend depends on the Jacobian of the error function. Since the error function of a constraint depends only on the values of two nodes, the Jacobian in Eq. (7) has the following form:

$$\mathbf{J}_{ij} = \begin{pmatrix} \mathbf{0}\cdots\mathbf{0} & \underbrace{\mathbf{A}_{ij}}_{\text{node } i} & \mathbf{0}\cdots\mathbf{0} & \underbrace{\mathbf{B}_{ij}}_{\text{node } j} & \mathbf{0}\cdots\mathbf{0} \end{pmatrix}. \quad (17)$$

Here $\mathbf{A}_{ij}$ and $\mathbf{B}_{ij}$ are the derivatives of the error function with respect to $\mathbf{x}_i$ and $\mathbf{x}_j$. From Eq. (10) we obtain the following structure for the block matrix $\mathbf{H}_{ij}$:

$$\mathbf{H}_{ij} = \begin{pmatrix} \ddots & & & & \\ & \mathbf{A}_{ij}^T\mathbf{\Omega}_{ij}\mathbf{A}_{ij} & \cdots & \mathbf{A}_{ij}^T\mathbf{\Omega}_{ij}\mathbf{B}_{ij} & \\ & \vdots & \ddots & \vdots & \\ & \mathbf{B}_{ij}^T\mathbf{\Omega}_{ij}\mathbf{A}_{ij} & \cdots & \mathbf{B}_{ij}^T\mathbf{\Omega}_{ij}\mathbf{B}_{ij} & \\ & & & & \ddots \end{pmatrix} \quad (18)$$

$$\mathbf{b}_{ij} = \begin{pmatrix} \vdots \\ \mathbf{A}_{ij}^T\mathbf{\Omega}_{ij}\mathbf{e}_{ij} \\ \vdots \\ \mathbf{B}_{ij}^T\mathbf{\Omega}_{ij}\mathbf{e}_{ij} \\ \vdots \end{pmatrix} \quad (19)$$

For simplicity of notation we omitted the zero blocks.

Algorithm 1 summarizes an iterative Gauss-Newton procedure to determine both the mean and the information matrix of the posterior over the robot poses. Since most of the structures in the system are sparse, we recommend to use memory efficient representations to store the Hessian $\mathbf{H}$ of the system. Since the structure of the Hessian is known in advance from the connectivity of the graph, we recommend to pre-allocate the Hessian once at the beginning of the iterations and to update it in place by looping over all edges whenever a new linearization is required. Each edge contributes to the blocks $\mathbf{H}_{[ii]}$, $\mathbf{H}_{[ij]}$, $\mathbf{H}_{[ji]}$, and $\mathbf{H}_{[jj]}$ and to the blocks $\mathbf{b}_{[i]}$ and $\mathbf{b}_{[j]}$ of the coefficient vector. An additional optimization is to compute only the upper triangular part of $\mathbf{H}$, since it is symmetric. Note that the error of a constraint $\mathbf{e}_{ij}$ depends only on the relative position of the connected poses $\mathbf{x}_i$ and $\mathbf{x}_j$. Accordingly, the error $\mathbf{F}(\mathbf{x})$of a particular configuration of the poses $\mathbf{x}$ is invariant under a rigid transformation of all the poses. This results in Eq. 15 being under determined. To numerically solve this system it is therefore common practice to constrain one of the increments $\Delta\mathbf{x}_k$ to be zero. This can be done by adding the identity matrix to the kth diagonal block $\mathbf{H}[kk]$. Without loss of generality in Algorithm 1 we fix the first node $\mathbf{x}_1$. An alternative way to fix a particular node of the pose-graph consists in suppressing the kth block row and the kth block column of the linear system in Eq. 15.

C. Least Squares on a Manifold

A common approach in numeric to deal with non-Euclidean spaces is to perform the optimization on a manifold. A manifold is a mathematical space that is not necessarily Euclidean on a global scale, but can be seen as Euclidean on a local scale [20]. Note that the manifold-based approach described here is similar to the way of minimizing functions in $SO(3)$ as described by Taylor and Kriegman [30].

In the context of the SLAM problem, each parameter block $\mathbf{x}_i$ consists of a translation vector $\mathbf{t}_i$ and a rotational component α_i. The translation $\mathbf{t}_i$ clearly forms a Euclidean space, while the rotational components α_i span over the non-Euclidean 2D or 3D rotation group $SO(2)$ or $SO(3)$. To avoid singularities, these spaces are usually described in an over-parametrized way, e.g., by rotation matrices or quaternions. Directly applying Eq. (16) to these over-parametrized

Algorithm 1 Computes the mean $\mathbf{x}^*$ and the information matrix $\mathbf{H}^*$ of the multivariate Gaussian approximation of the robot pose posterior from a graph of constraints.

Require: $\breve{\mathbf{x}} = \breve{\mathbf{x}}_{1:T}$: initial guess. $\mathcal{C} = \{\langle \mathbf{e}_{ij}(\cdot), \mathbf{\Omega}_{ij} \rangle\}$: constraints
Ensure: $\mathbf{x}^*$: new solution, $\mathbf{H}^*$ new information matrix
  // find the maximum likelihood solution
  while ¬ converged **do**
    $\mathbf{b} \leftarrow \mathbf{0}$ $\mathbf{H} \leftarrow \mathbf{0}$
    for all $\langle \mathbf{e}_{ij}, \mathbf{\Omega}_{ij} \rangle > \in \mathcal{C}$ **do**
      // Compute the Jacobians $\mathbf{A}_{ij}$ and $\mathbf{B}_{ij}$ of the error function
      $$\mathbf{A}_{ij} \leftarrow \left.\frac{\partial \mathbf{e}_{ij}(\mathbf{x})}{\partial \mathbf{x}_i}\right|_{\mathbf{x}=\breve{\mathbf{x}}} \qquad \mathbf{B}_{ij} \leftarrow \left.\frac{\partial \mathbf{e}_{ij}(\mathbf{x})}{\partial \mathbf{x}_j}\right|_{\mathbf{x}=\breve{\mathbf{x}}}$$
      // compute the contribution of this constraint to the linear system
      $\mathbf{H}_{[ii]} += \mathbf{A}_{ij}^T \mathbf{\Omega}_{ij} \mathbf{A}_{ij}$ $\mathbf{H}_{[ij]} += \mathbf{A}_{ij}^T \mathbf{\Omega}_{ij} \mathbf{B}_{ij}$
      $\mathbf{H}_{[ji]} += \mathbf{B}_{ij}^T \mathbf{\Omega}_{ij} \mathbf{A}_{ij}$ $\mathbf{H}_{[jj]} += \mathbf{B}_{ij}^T \mathbf{\Omega}_{ij} \mathbf{B}_{ij}$
      // compute the coefficient vector
      $\mathbf{b}_{[i]} += \mathbf{A}_{ij}^T \mathbf{\Omega}_{ij} \mathbf{e}_{ij}$ $\mathbf{b}_{[j]} += \mathbf{B}_{ij}^T \mathbf{\Omega}_{ij} \mathbf{e}_{ij}$
    end for
    // keep the first node fixed
    $\mathbf{H}_{[11]} += \mathbf{I}$
    // solve the linear system using sparse Cholesky factorization
    $\mathbf{\Delta x} \leftarrow \text{solve}(\mathbf{H}\,\mathbf{\Delta x} = -\mathbf{b})$
    // update the parameters
    $\breve{\mathbf{x}} += \mathbf{\Delta x}$
  end while
  $\mathbf{x}^* \leftarrow \breve{\mathbf{x}}$
  $\mathbf{H}^* \leftarrow \mathbf{H}$
  // release the first node
  $\mathbf{H}^*_{[11]} -= \mathbf{I}$
  return $\langle \mathbf{x}^*, \mathbf{H}^* \rangle$

representations breaks the constraints induced by the over-parametrization. The over-parametrization results in additional degrees of freedom and thus introduces errors in the solution. To overcome this problem, one can use a minimal representation for the rotation (like, e.g., Euler angles in 3D). This, however, is subject to singularities. The singularities in the 2D case can be easily recovered by normalizing the angle, however in 3D this procedure is not straightforward.

An alternative idea is to consider the underlying space as a manifold and to define an operator $\boxplus$ that maps a local variation $\mathbf{\Delta x}$ in the Euclidean space to a variation on the manifold, $\mathbf{\Delta x} \mapsto \mathbf{x} \boxplus \mathbf{\Delta x}$. We refer the reader to the work of Hertzberg [14] for the mathematical details. With this operator, a new error function can be defined as

$$\breve{\mathbf{e}}_{ij}(\mathbf{\Delta \tilde{x}}_i, \mathbf{\Delta \tilde{x}}_j) \stackrel{\text{def.}}{=} \mathbf{e}_{ij}(\breve{\mathbf{x}}_i \boxplus \mathbf{\Delta \tilde{x}}_i, \breve{\mathbf{x}}_j \boxplus \mathbf{\Delta \tilde{x}}_j) \tag{20}$$

$$= \mathbf{e}_{ij}(\breve{\mathbf{x}} \boxplus \mathbf{\Delta \tilde{x}}) \simeq \breve{\mathbf{e}}_{ij} + \tilde{\mathbf{J}}_{ij}\,\mathbf{\Delta \tilde{x}}, \tag{21}$$

where $\breve{\mathbf{x}}$ spans over the original over-parametrized space, for instance quaternions. The term $\mathbf{\Delta \tilde{x}}$ is a small increment around the original position $\breve{\mathbf{x}}$ and is expressed in a minimal representation.

As an example, in 3D SLAM a good choice of the parametrization of the rotations is the *vector part* of the unit quaternion. In more detail, one can represent the increments $\mathbf{\Delta \tilde{x}}$ as 6D vectors $\mathbf{\Delta \tilde{x}}^T = (\mathbf{\Delta \tilde{t}}^T\, \tilde{\mathbf{q}}^T)$, where $\mathbf{\Delta \tilde{t}}$ denotes the translation and $\tilde{\mathbf{q}}^T = (\Delta q_x\, \Delta q_y\, \Delta q_z)^T$ is the vector part of the unit quaternion representing the 3D rotation. Conversely, $\breve{\mathbf{x}}^T = (\breve{\mathbf{t}}^T\, \breve{\mathbf{q}}^T)$ uses a quaternion $\breve{\mathrm{q}}$ to encode the rotational part. Thus, the operator $\boxplus$ can be expressed by first converting $\mathbf{\Delta \tilde{q}}$ to a full quaternion $\mathbf{\Delta q}$ and then applying the transformation $\mathbf{\Delta x}^T = (\mathbf{\Delta t}^T\, \mathbf{\Delta q}^T)$ to $\breve{\mathbf{x}}$. In the equations describing the error minimization, these operations can nicely be encapsulated by the $\boxplus$ operator. The Jacobian $\tilde{\mathbf{J}}_{ij}$ can be expressed by

$$\tilde{\mathbf{J}}_{ij} = \left.\frac{\partial \mathbf{e}_{ij}(\breve{\mathbf{x}} \boxplus \mathbf{\Delta \tilde{x}})}{\partial \mathbf{\Delta \tilde{x}}}\right|_{\mathbf{\Delta \tilde{x}}=\mathbf{0}} \tag{22}$$

Since in the previous equation $\mathbf{e}$ depends only on $\mathbf{\Delta \tilde{x}}_i$ and $\mathbf{\Delta \tilde{x}}_j$ we can further expand it as follows:

$$\tilde{\mathbf{J}}_{ij} = \Bigg(\cdots \underbrace{\left.\frac{\partial \mathbf{e}_{ij}(\breve{\mathbf{x}} \boxplus \mathbf{\Delta \tilde{x}})}{\partial \mathbf{\Delta \tilde{x}}_i}\right|_{\mathbf{\Delta \tilde{x}}=\mathbf{0}}}_{\tilde{\mathbf{A}}_{ij}} \cdots \quad \underbrace{\left.\frac{\partial \mathbf{e}_{ij}(\breve{\mathbf{x}} \boxplus \mathbf{\Delta \tilde{x}})}{\partial \mathbf{\Delta \tilde{x}}_j}\right|_{\mathbf{\Delta \tilde{x}}=\mathbf{0}}}_{\tilde{\mathbf{B}}_{ij}} \cdots \Bigg) \tag{23}$$

Using the rule for the partial derivatives and exploiting the fact that the Jacobian is evaluated in $\mathbf{\Delta \tilde{x}} = \mathbf{0}$, the non-zero blocks become:

$$\frac{\partial \mathbf{e}_{ij}(\breve{\mathbf{x}} \boxplus \mathbf{\Delta \tilde{x}}_i)}{\partial \mathbf{\Delta \tilde{x}}_i} = \underbrace{\frac{\partial \mathbf{e}_{ij}(\breve{\mathbf{x}})}{\partial \breve{\mathbf{x}}_i}}_{\mathbf{A}_{ij}} \cdot \underbrace{\left.\frac{\breve{\mathbf{x}}_i \boxplus \mathbf{\Delta \tilde{x}}_i}{\partial \mathbf{\Delta \tilde{x}}_i}\right|_{\mathbf{\Delta \tilde{x}}=\mathbf{0}}}_{\mathbf{M}_i} \tag{24}$$

$$\frac{\partial \mathbf{e}_{ij}(\breve{\mathrm{x}} \boxplus \mathbf{\Delta \tilde{x}}_j)}{\partial \mathbf{\Delta \tilde{x}}_j} = \underbrace{\frac{\partial \mathbf{e}_{ij}(\breve{\mathrm{x}})}{\partial \breve{\mathbf{x}}_j}}_{\mathbf{B}_{ij}} \cdot \underbrace{\left.\frac{\breve{\mathbf{x}}_j \boxplus \mathbf{\Delta \tilde{x}}_j}{\partial \mathbf{\Delta \tilde{x}}_j}\right|_{\mathbf{\Delta \tilde{x}}=\mathbf{0}}}_{\mathbf{M}_j}. \tag{25}$$

Accordingly, one can easily derive from the Jacobian not defined on a manifold of Eq. 17 a Jacobian on a manifold just by multiplying its non-zero blocks with the derivative of the $\boxplus$ operator computed in $\breve{\mathbf{x}}_i$ and $\breve{\mathbf{x}}_j$.

With a straightforward extension of the notation, we can insert Eq. (21) in Eq. (9). This leads to the following linear system:

$$\tilde{\mathbf{H}} \mathbf{\Delta \tilde{x}}^* = -\tilde{\mathbf{b}}. \tag{26}$$

Since the increments $\mathbf{\Delta \tilde{x}}^*$ are computed in the local Euclidean surroundings of the initial guess $\breve{\mathbf{x}}$, they need to be re-mapped into the original over-parametrized space by the $\boxplus$ operator. Accordingly, the update rule of Eq. (16) becomes

$$\mathbf{x}^* = \breve{\mathbf{x}} \boxplus \Delta\tilde{\mathbf{x}}^*. \tag{27}$$

Thus, formalizing the minimization problem on a manifold consists of first computing a set of increments in a local Euclidean approximation around the initial guess by Eq. (26), and second accumulating the increments in the global non-Euclidean space by Eq. (27). Note that the linear system computed on a manifold representation has the same structure of the linear system computed on an Euclidean space. One can easily derive a manifold version of a graph minimization from a non-manifold version, only by defining an $\boxplus$ operator and its Jacobian $\mathbf{M}_i$ w.r.t. the corresponding parameter block. Algorithm 2 provides a manifold version of the Gauss-Newton method for SLAM.

The Hessian $\tilde{\mathbf{H}}$ of the manifold problem no longer represents the information matrix of the trajectories but of the trajectory increments $\Delta\tilde{\mathbf{x}}$. To obtain the information matrix of the trajectory Algorithm 2 computes $\mathbf{H}$ in the original space of the poses $\mathbf{x}$.

V. Practical Applications

In this section we describe some applications of the proposed methods. In the first scenario we describe a complete 2D mapping system, and in the second scenario we briefly describe a 3D mapping system and we highlight the advantages of a manifold representation.

A. 2D Laser Based Mapping

We processed the data recorded with the mobile robot equipped with a laser range finder illustrated in Figure 8 at the Intel Research Laboratory in Seattle. This data consists of odometry measurements describing 2D transformations

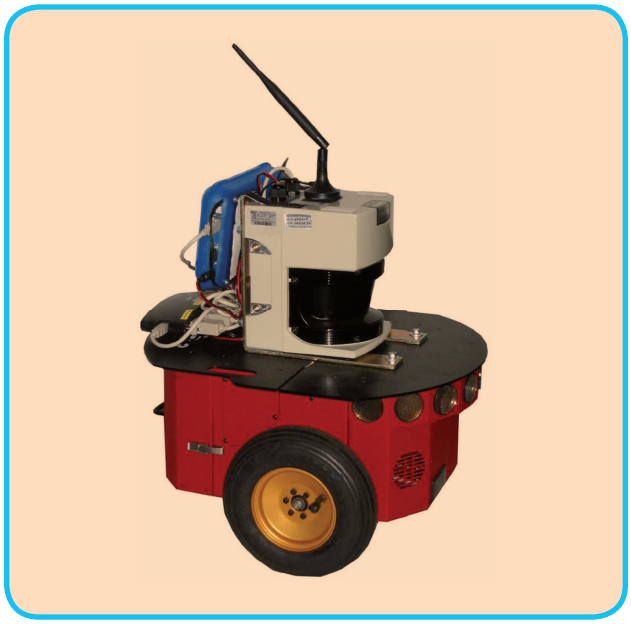


FIG 8 A typical robot used in 2D mapping experiments. The platform is a standard ActivMedia Pioneer 2 equipped with a SICK-LMS range finder.

Algorithm 2 Manifold version of Algorithm 1. While this algorithm has the same computational complexity, it is substantially more robust than the non-manifold version, especially in the 3D case.

Require: $\breve{\mathbf{x}} = \breve{\mathbf{x}}_{1:T}$: initial guess. $\mathcal{C} = \{\langle \mathbf{e}_{ij}(\cdot), \mathbf{\Omega}_{ij}\rangle\}$: constraints
Ensure: $\mathbf{x}^*$: new solution, $\breve{\mathbf{H}}^*$ new information matrix

```
//find the maximum likelihood solution
while ¬ converged do
  //Compute the auxiliary Jacobians M_{1:T} over the mani fold
  for all x̆_i ∈ x̆ do
    M_i ← ∂(x̆_i ⊞ Δx̃_i)/∂Δx̃_i |_{Δx̃=0}
  end for
  b̃ ← 0   H̃ ← 0
  for all ⟨e_ij, Ω_ij⟩ ∈ C do
    //Compute the Jacobians A_ij and B_ij of the error function
    A_ij ← ∂e_ij(x)/∂x_i |_{x=x̆}   B_ij ← ∂e_ij(x)/∂x_j |_{x=x̆}
    //Project the Jacobians through the manifold
    Ã_ij ← A_ij M_i   B̃_ij ← B_ij M_j
    // compute the nonzero Hessian blocks
    H̃_[ii] += Ã_ij^T Ω_ij Ã_ij   H̃_[ij] += Ã_ij^T Ω_ij B̃_ij
    H̃_[ji] += B̃_ij^T Ω_ij Ã_ij   H̃_[jj] += B̃_ij^T Ω_ij B̃_ij
    // compute the coefficient vector
    b̃_[i] += Ã_ij^T Ω_ij e_ij   b̃_[j] += B̃_ij^T Ω_ij e_ij
  end for
  //keep the first node fixed
  H_[11] += I
  //solve the linear system using sparse Cholesky factorization
  Δx̃ ← solve(H̃ Δx̃ = − b̃)
  //update the parameters
  for all x̆_i ∈ x̆ do
    x̆_i ← x̆_i ⊞ Δx̃_i
  end for
end while
x* ← x̆
//the maximum is found, now compute the Hessian in the original space
H* ← 0
for all ⟨e_ij, Ω_ij⟩ ∈ C do
  H_[ii] += A_ij^T Ω_ij A_ij   H_[ij] += A_ij^T Ω_ij B_ij
  H_[ji] += B_ij^T Ω_ij A_ij   H_[jj] += B_ij^T Ω_ij B_ij
end for
return ⟨x*, H*⟩
```

Since the robot travels at a velocity of around 1 m/s the graph optimization could be executed after adding every node instead of after detecting a loop closure.

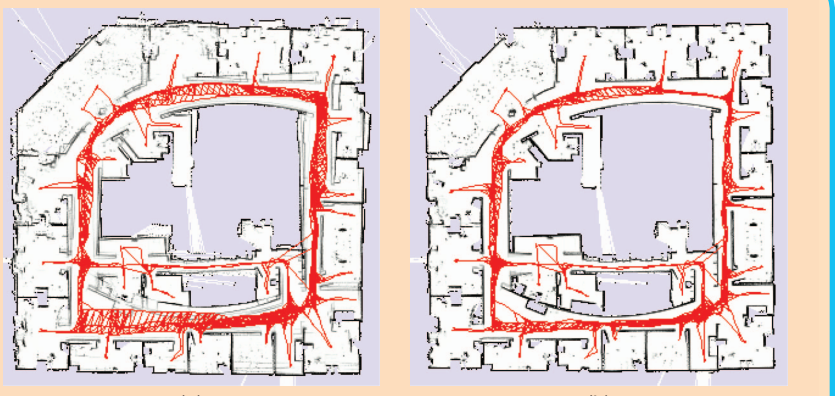


FIG 9 Intel Research Lab. Left: Unoptimized pose graph overlayed on top of the resulting map. Right: The optimized pose graph and the resulting consistent map.

corresponding to the movements of the platform between consecutive time frames, and 2D laser range data.

The graph is constructed in the following way:

- Whenever the robot moves more than 0.5 meters or rotates more than 0.5 radians, the algorithm adds a new vertex to the graph and labels it with the current laser observation.
- This laser scan is matched with the previously acquired one to improve the odometry estimate and the corresponding edge is added to the graph. We use a variant of the scan-matcher described by Olson [26].
- When the robot reenters a known area after traveling for a long time in a previously unknown region, the algorithm seeks for matches of the current scan with the past measurements (loop closing). If a matching between the current observation and the observation of another node succeeds, the algorithm adds a new edge to the graph. The edge is labeled with the relative transformation that makes the two scans to overlap best. Matching the current measurement with all previous scans would be extremely inefficient and error prone, since it does not consider the known prior about the robot location. Instead, the algorithm selects the candidate nodes in the past as the ones whose 3σ marginal covariances contains the current robot pose. These covariances can be obtained as the diagonal blocks of the inverse of a reduced Hessian $\mathbf{H}_{\text{red}}$. $\mathbf{H}_{\text{red}}$ is obtained from $\mathbf{H}$ by removing rows and the columns of the newly inserted robot pose. $\mathbf{H}_{\text{red}}$ is the information matrix of all the trajectory when assuming fixed the current position.
- The algorithm performs the optimization whenever a loop closure is detected.

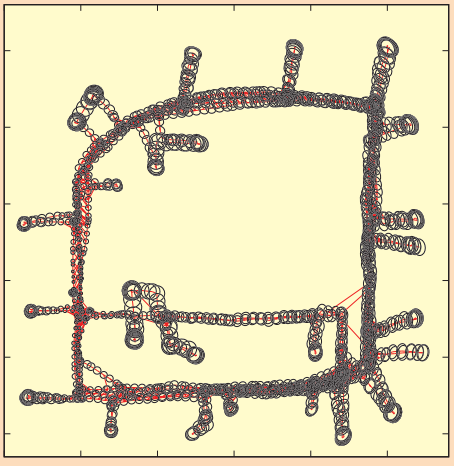


FIG 10 Pose uncertainty estimate for a real-world data set.

At the end of the run, the graph consists of 1,802 nodes and 3,546 edges. Even for this relatively large problem the optimization can be carried on in 100 ms on a standard laptop (Intel Core2@2.4 GhZ). Since the robot travels at a velocity of around 1 m/s the graph optimization could be executed after adding every node instead of after detecting a loop closure. Figure 9 shows the effect of the optimization process on the trajectory, while Figure 10 illustrates the uncertainty ellipses. The robot is located in the region where the ellipse become small. Note that the poses in $SE(2)$ do not need to be over parameterized, so in this case there is no advantage in utilizing manifolds.

B. 3D Laser Based Mapping

Extending to 3D the SLAM algorithm presented in the previous section is rather straightforward. One has only

to replace the 2D scan matching and loop closure detection with their 3D counterparts that operate on 3D point clouds instead than on single laser scans. In our implementation we utilize the popular ICP algorithm [1] and for determining the loop closures we use the algorithm by Steder *et al.* [29]. Additionally, each node of the graph and each constraint lives in $SE(3)$. Typical outputs of this algorithm are illustrated in Figures 2(a) and (b).

The minimum number of parameters required to represent an element of $SE(3)$ is 6, a possible choice consists in a 3D translation vector plus the three Euler angles. Utilizing this parametrization leads to Algorithm 1. However, this minimal representation is subject to singularities that can be avoided by utilizing an over-parametrized state space.

The time to compute the linear system is negligible compared to the time to solve it. Accordingly, the choice of the parametrization mainly affects the convergence speed, not the time required to performone iteration.

Appendix

In the following we will provide the definitions and the derivations for the Jacobians to implement the suggested algorithm. Due to space limitations we do not expand the Jacobians in the 3D case. However, these Jacobians can either be computed numerically or by using a computer algebra system.

Error Functions and Jacobians for the 2D case
The basic entities in the 2D case are defined as

$$\mathbf{x}_i^\top = (\mathbf{t}_i^\top, \theta_i) \tag{28}$$

$$\mathbf{z}_{ij}^\top = (\mathbf{t}_{ij}^\top, \theta_{ij}) \tag{29}$$

where $\mathbf{t}_i$ and $\mathbf{t}_{ij}$ are 2D vectors and θ_i and θ_{ij} are rotation angles which are normalized to $[-\pi, \pi)$. The error function is

$$\mathbf{e}_{ij}(\mathbf{x}) = \begin{pmatrix} \mathbf{R}_{ij}^\top (\mathbf{R}_i^\top (\mathbf{t}_j - \mathbf{t}_i) - \mathbf{t}_{ij}) \\ \theta_j - \theta_i - \theta_{ij} \end{pmatrix}, \tag{30}$$

where $\mathbf{R}_i$ and $\mathbf{R}_{ij}$ are the 2×2 rotation matrices of θ_i and θ_{ij}

$$\mathbf{R}_i = \begin{pmatrix} \cos(\theta_i) & -\sin(\theta_i) \\ \sin(\theta_i) & \cos(\theta_i) \end{pmatrix}. \tag{31}$$

The Jacobians of the error function are

$$\mathbf{A}_{ij} = \frac{\partial \mathbf{e}_{ij}(\mathbf{x})}{\partial \mathbf{x}_i} = \begin{pmatrix} -\mathbf{R}_{ij}^\top \mathbf{R}_i^\top & \mathbf{R}_{ij}^\top \frac{\partial \mathbf{R}_i^\top}{\partial \theta_i} (\mathbf{t}_j - \mathbf{t}_i) \\ \mathbf{0}^\top & -1 \end{pmatrix} \tag{32}$$

$$\mathbf{B}_{ij} = \frac{\partial \mathbf{e}_{ij}(\mathbf{x})}{\partial \mathbf{x}_j} = \begin{pmatrix} \mathbf{R}_{ij}^\top \mathbf{R}_i^\top & \mathbf{0} \\ \mathbf{0}^\top & 1 \end{pmatrix}. \tag{33}$$

The $\boxplus$ operator is defined as

$$\mathbf{x} \boxplus \Delta\tilde{\mathbf{x}} = \mathbf{x} + \Delta\tilde{\mathbf{x}} \tag{34}$$

The angles are normalized to $[-\pi, \pi)$ after applying the increments. The Jacobians of the manifold in the 2D case evaluate to the identity matrix:

$$M_i = \left. \frac{\mathbf{x}_i \boxplus \Delta\tilde{\mathbf{x}}_i}{\partial \Delta\tilde{\mathbf{x}}_i} \right|_{\Delta\tilde{\mathbf{x}}=0} = \mathbf{I}_3 \tag{35}$$

$$M_j = \left. \frac{\mathbf{x}_j \boxplus \Delta\tilde{\mathbf{x}}_j}{\partial \Delta\tilde{\mathbf{x}}_j} \right|_{\Delta\tilde{\mathbf{x}}=0} = \mathbf{I}_3 \tag{36}$$

Error Functions for the 3D case
The basic entities in the 3D case are defined as

$$\mathbf{x}_i^\top = (\mathbf{t}_i^\top, \mathbf{q}_i^\top) \tag{37}$$

$$\mathbf{z}_{ij}^\top = (\mathbf{t}_{ij}^\top, \mathbf{q}_{ij}^\top), \tag{38}$$

where $\mathbf{q}$ denotes the unit quaternion $\mathbf{q}^\top = (q_x, q_y, q_z, q_w)^\top$, i.e., $\|\mathbf{q}\| = 1$. The error function is

$$\mathbf{e}_{ij}(\mathbf{x}) = (\mathbf{z}_{ij}^{-1} \oplus (\mathbf{x}_i^{-1} \oplus \mathbf{x}_j))_{[1:6]}, \tag{39}$$

where $\oplus$ is the motion composition operator

$$\mathbf{x}_i \oplus \mathbf{x}_j = \begin{pmatrix} \mathbf{q}_i(\mathbf{t}_j) \\ \mathbf{q}_i \cdot \mathbf{q}_j \end{pmatrix} \tag{40}$$

and the operator $(\cdot)_{[1:6]}$ selects the first 6 elements of its vector argument.

The Jacobians of the error function are:

$$\mathbf{A}_{ij} = \frac{\partial \mathbf{e}_{ij}(\mathbf{x})}{\partial \mathbf{x}_i} \tag{41}$$

$$\mathbf{B}_{ij} = \frac{\partial \mathbf{e}_{ij}(\mathbf{x})}{\partial \mathbf{x}_j}. \tag{42}$$

The $\boxplus$ operator maps $\Delta\tilde{\mathbf{x}}_i^\top = (\Delta\tilde{\mathbf{t}}_i^\top, \Delta\tilde{\mathbf{q}}_i^\top)$ to the original space

$$\mathbf{x}_i \boxplus \Delta\tilde{\mathbf{x}}_i = \mathbf{x}_i \oplus \begin{pmatrix} \Delta\tilde{\mathbf{t}}_i \\ \Delta\tilde{\mathbf{q}}_i \\ \sqrt{1 - \|\Delta\tilde{\mathbf{q}}_i\|^2} \end{pmatrix}, \tag{43}$$

where $\Delta\tilde{\mathbf{t}}_i$ denotes the translation and $\Delta\tilde{\mathbf{q}}^\top = (\Delta q_x, \Delta q_y, \Delta q_z)^\top$ is the vector part of the unit quaternion representing the 3D rotation and thus $\|\Delta\tilde{\mathbf{q}}_i\| \leq 1$.

The Jacobians of the manifold in the 3D case are given by

$$M_i = \left. \frac{\mathbf{x}_i \boxplus \Delta\tilde{\mathbf{x}}_i}{\partial \Delta\tilde{\mathbf{x}}_i} \right|_{\Delta\tilde{\mathbf{x}}=0} \tag{44}$$

$$M_j = \left. \frac{\mathbf{x}_j \boxplus \Delta\tilde{\mathbf{x}}_j}{\partial \Delta\tilde{\mathbf{x}}_j} \right|_{\Delta\tilde{\mathbf{x}}=0}. \tag{45}$$

The algorithms presented in this paper can be used as a building blocks of more sophisticated methods, however optimized implementations of these algorithms can deal with surprisingly large problems.

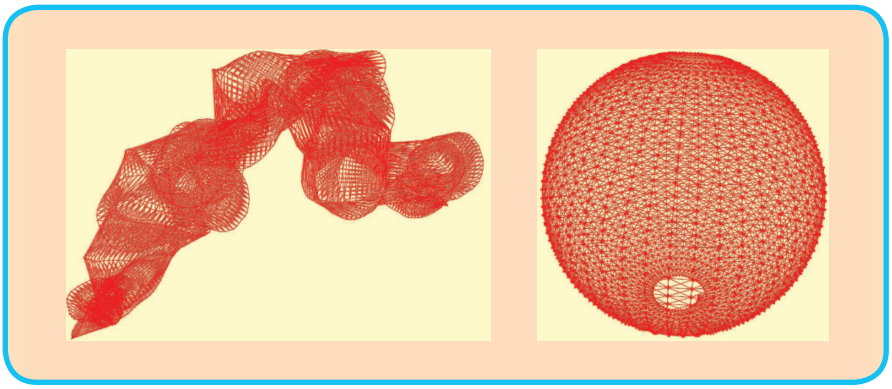


FIG 11 Pose-graph obtained by simulating a robot moving on a sphere. Left: Initial configuration. Right: After optimizing the pose graph the sphere has accurately been recovered by Algorithm 2.

Alternatively, one can describe the relative perturbations of the optimization problem $\Delta\tilde{\mathbf{x}}$ in a minimal representation while leaving the poses in the original over-parametrized space. This leads to Algorithm 2. In this section we compare these two variants of the optimization algorithm on a pose-graph obtained by a simulated robot. Note that the sparsity pattern of the Hessian is the same in both cases. Furthermore, the time to compute the linear system is negligible compared to the time to solve it. Accordingly, the choice of the parametrization mainly affects the convergence speed, not the time required to perform one iteration. To highlight this effect we show the evolution of the error per iteration during one optimization run by using the two algorithms.

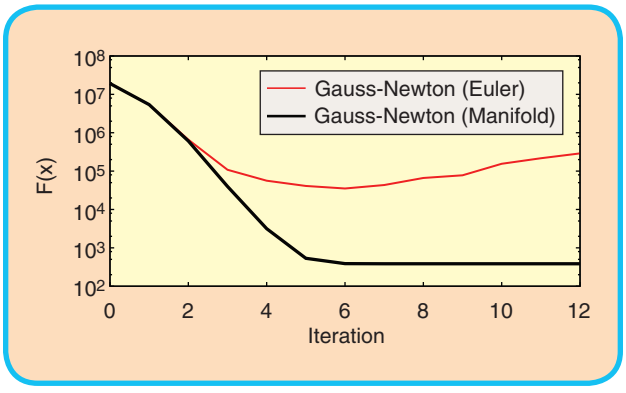


FIG 12 Evolution of the error $\mathbf{F}(\mathbf{x})$ for Gauss-Newton optimization with Euler angles and with manifold linearization to the 3D sphere data set.

We use a simulated 3D data set of a robot traveling on the surface of a sphere. The measurements were affected by a significant error, and initializing the system by using the odometry information resulted in the graph illustrated in the left part of Figure 11. Starting from this initial guess we executed the Gauss-Newton Algorithm with and without the manifold linearization, i.e., here by using Euler angles. Figure 12 shows the evolution of the error during the iterations of the two approaches. First both approaches are able to decrease the error. However, not appropriately considering the singularities leads to a divergence of Algorithm 1 while Algorithm 2 converges to the right solution.

VI. Conclusions

In this paper we presented a tutorial on graph-based SLAM. Our aim was to provide the reader with sufficient details and insights to allow for an easy implementation of the proposed methods. The algorithms presented in this paper can be used as a building blocks of more sophisticated methods, however optimized implementations of these algorithms can deal with surprisingly large problems.

About the Authors

Giorgio Grisetti is working as a postdoctoral researcher in the Autonomous Intelligent Systems Lab at Freiburg University. He was a Ph.D. student at University of Rome "La Sapienza" in the Intelligent Systems Lab headed by Daniele Nardi where he received his Ph.D. degree in April 2006. He is currently member of Department of Systems and Computer Engineering at "La Sapienza" University of Rome, as assistant professor. His research interests lie in the areas of mobile robotics. His previous and current contributions in robotics aims to provide effective solutions to various mobile robot navigation problems including SLAM, localization, and path planning.

Wolfram Burgard is a professor for computer science at the University of Freiburg where he heads of the Laboratory for Autonomous Intelligent Systems. He received his Ph.D. degree in Computer Science from the University of

Bonn in 1991. His areas of interest lie in artificial intelligence and mobile robots. In the past, Wolfram Burgard and his group developed several innovative probabilistic techniques for robot navigation and control. They cover different aspects such as localization, map-building, path-planning, and exploration. For his work, Wolfram Burgard received several best paper awards from outstanding national and international conferences. In 2008, Wolfram Burgard became Fellow of the European Coordinating Committee for Artificial Intelligence.

Rainer Kuemmerle studied computer science at the University of Freiburg and received his Master degree in April 2007. He is currently working as a PhD-student in the Autonomous Intelligent Systems Lab of the University of Freiburg headed by Wolfram Burgard. His research interests lie in the areas of outdoor navigation, 3D mapping, and localization.

Cyrill Stachniss is working as an academic advisor in the Lab for Autonomous Intelligent Systems at the University of Freiburg. He is an associate editor of the IEEE Transactions on Robotics and a Microsoft Research Faculty Fellow. In his research, he focuses on probabilistic techniques in the context of mobile robotics. His areas of research include autonomous exploration in combination with simultaneous localization and mapping, classification and learning approaches including scene analysis as well as computer controlled cars, vision, and navigation techniques.

References

[1] P. J. Besl and N. D. McKay, "A method for registration of 3-d shapes," *IEEE Trans. Pattern Anal. Machine Intell.*, vol. 14, no. 2, pp. 239–256, 1992.

[2] M. Bosse, P. M. Newman, J. J. Leonard, and S. Teller, "An ATLAS framework for scalable mapping," in *Proc. IEEE Int. Conf. Robotics and Automation (ICRA)*, 2003, pp. 1899–1906.

[3] J. A. Castellanos, J. M. M. Montiel, J. Neira, and J. D. Tardós, "The SPmap: A probabilistic framework for simultaneous localization and map building," *IEEE Trans. Robot. Automat.*, vol. 15, no. 5, pp. 948–953, 1999.

[4] T. A. Davis, *Direct Methods for Sparse Linear Systems* (SIAM Book Series on the Fundamentals of Algorithms). Philadelphia, PA: SIAM, 2006.

[5] F. Dellaert and M. Kaess, "Square root SAM: Simultaneous location and mapping via square root information smoothing," *Int. J. Robot. Res.*, 2006.

[6] C. Estrada, J. Neira, and J. D. Tardós, "Hierachical SLAM: Real-time accurate mapping of large environments," *IEEE Trans. Robot.*, vol. 21, no. 4, pp. 588–596, 2005.

[7] R. Eustice, H. Singh, and J. J. Leonard, "Exactly sparse delayed-state filters," in *Proc. IEEE Int. Conf. Robotics and Automation (ICRA)*, Barcelona, Spain, 2005, pp. 2428–2435.

[8] U. Frese, P. Larsson, and T. Duckett, "A multilevel relaxation algorithm for simultaneous localisation and mapping," *IEEE Trans. Robot.*, vol. 21, no. 2, pp. 1–12, 2005.

[9] G. Grisetti, C. Stachniss, and W. Burgard, "Improved techniques for grid mapping with rao-blackwellized particle filters," *IEEE Trans. Robot.*, vol. 23, no. 1, pp. 34–46, 2007.

[10] G. Grisetti, C. Stachniss, and W. Burgard, "Non-linear constraint network optimization for efficient map learning," *IEEE Trans. Intell. Transport. Syst.*, 2009.

[11] J.-S. Gutmann and K. Konolige, "Incremental mapping of large cyclic environments," in *Proc. IEEE Int. Symp. Computational Intelligence in Robotics and Automation (CIRA)*, 1999.

[12] D. Hähnel, W. Burgard, D. Fox, and S. Thrun, "An efficient FastSLAM algorithm for generating maps of large-scale cyclic environments from raw laser range measurements," in *Proc. IEEE/RSJ Int. Conf. Intelligent Robots and Systems (IROS)*, Las Vegas, NV, 2003, pp. 206–211.

[13] D. Hähnel, W. Burgard, B. Wegbreit, and S. Thrun, "Towards lazy data association in slam," in *Proc. Int. Symp. Robotics Research (ISRR)*, Siena, Italy, 2003, pp. 421–431.

[14] C. Hertzberg, "A framework for sparse, non-linear least squares problems on manifolds," Master's thesis, Univ. Bremen, 2008.

[15] A. Howard, M. J. Matarić, and G. Sukhatme, "Relaxation on a mesh: A formalism for generalized localization," in *Proc. IEEE/RSJ Int. Conf. Intelligent Robots and Systems (IROS)*, 2001.

[16] M. Kaess, A. Ranganathan, and F. Dellaert, "iSAM: Fast incremental smoothing and mapping with efficient data association," in *Proc. IEEE Int. Conf. Robotics and Automation (ICRA)*, Rome, Italy, 2007.

[17] K. Konolige, "A gradient method for realtime robot control," in *Proc. IEEE/RSJ Int. Conf. Intelligent Robots and Systems (IROS)*, 2000.

[18] K. Konolige, J. Bowman, J. D. Chen, P. Mihelich, M. Calonder, V. Lepetit, and P. Fua, "View-based maps," *Int. J. Robot. Res.*, vol. 29, no. 10, 2010.

[19] K. Konolige, G. Grisetti, R. Kümmerle, W. Burgard, B. Limketkai, and R. Vincent, "Sparse pose adjustment for 2d mapping," in *Proc. IEEE/RSJ Int. Conf. Intelligent Robots and Systems (IROS)*, 2010.

[20] J. M. Lee, *Introduction to Smooth Manifolds* (Graduate Texts in Mathematics, vol. 218). Berlin: Springer-Verlag, 2003.

[21] F. Lu and E. Milios, "Globally consistent range scan alignment for environment mapping," *Autonom. Robots*, vol. 4, pp. 333–349, 1997.

[22] M. Montemerlo, S. Thrun, D. Koller, and B. Wegbreit, "FastSLAM: A factored solution to simultaneous localization and mapping," in *Proc. Nat. Conf. Artificial Intelligence (AAAI)*, Edmonton, Canada, 2002, pp. 593–598.

[23] J. Neira and J. D. Tardós, "Data association in stochastic mapping using the joint compatibility test," *IEEE Trans. Robot. Automat.*, vol. 17, no. 6, pp. 890–897, 2001.

[24] A. Nüchter, K. Lingemann, J. Hertzberg, and H. Surmann, "6D SLAM with approximate data association," in *Proc. Int. Conf. Advanced Robotics (ICAR)*, 2005, pp. 242–249.

[25] E. Olson, "Robust and efficient robotic mapping," Ph.D. thesis, MIT, Cambridge, MA, June 2008.

[26] E. Olson, "Real-time correlative scan matching," in *Proc. IEEE Int. Conf. Robotics and Automation (ICRA)*, 2009.

[27] E. Olson, J. Leonard, and S. Teller, "Fast iterative optimization of pose graphs with poor initial estimates," in *Proc. IEEE Int. Conf. Robotics and Automation (ICRA)*, 2006, pp. 2262–2269.

[28] R. Smith, M. Self, and P. Cheeseman, "Estimating uncertain spatial realtionships in robotics," in *Autonomous Robot Vehicles*, I. Cox and G. Wilfong, Eds. Berlin: Springer-Verlag, 1990, pp. 167–193.

[29] B. Steder, G. Grisetti, and W. Burgard, "Robust place recognition for 3D range data based on point features," in *Proc. IEEE Int. Conf. Robotics and Automation (ICRA)*, 2010.

[30] C. J. Taylor and D. J. Kriegman, "Minimization on the Lie group SO(3) and related manifolds," Yale Univ., Tech. Rep. 9405, 1994.

[31] S. Thrun, Y. Liu, D. Koller, A. Y. Ng, Z. Ghahramani, and H. Durrant-Whyte, "Simultaneous localization and mapping with sparse extended information filters," *Int. J. Robot. Res.*, vol. 23, no. 7/8, pp. 693–716, 2004.

[32] S. Thrun and M. Montemerlo, "The graph SLAM algorithm with applications to large-scale mapping of urban structures," *Int. J. Robot. Res.*, vol. 25, no. 5-6, p. 403, 2006.

[33] R. Triebel, P. Pfaff, and W. Burgard, "Multi-level surface maps for outdoor terrain mapping and loop closing," in *Proc. IEEE/RSJ Int. Conf. Intelligent Robots and Systems (IROS)*, 2006.

ITS

Auton Robot
DOI 10.1007/s10514-016-9548-2

Low-drift and real-time lidar odometry and mapping

Ji Zhang[1] · **Sanjiv Singh**[1]

Received: 25 October 2014 / Accepted: 7 February 2016

Abstract Here we propose a real-time method for low-drift odometry and mapping using range measurements from a 3D laser scanner moving in 6-DOF. The problem is hard because the range measurements are received at different times, and errors in motion estimation (especially without an external reference such as GPS) cause mis-registration of the resulting point cloud. To date, coherent 3D maps have been built by off-line batch methods, often using loop closure to correct for drift over time. Our method achieves both low-drift in motion estimation and low-computational complexity. The key idea that makes this level of performance possible is the division of the complex problem of Simultaneous Localization and Mapping, which seeks to optimize a large number of variables simultaneously, into two algorithms. One algorithm performs odometry at a high-frequency but at low fidelity to estimate velocity of the laser scanner. Although not necessary, if an IMU is available, it can provide a motion prior and mitigate for gross, high-frequency motion. A second algorithm runs at an order of magnitude lower frequency for fine matching and registration of the point cloud. Combination of the two algorithms allows map creation in real-time. Our method has been evaluated by indoor and outdoor experiments as well as the KITTI odometry benchmark. The results indicate that the proposed method can achieve accuracy comparable to the state of the art offline, batch methods.

Keywords Ego-motion estimation · Mapping · Continuous-time · Lidar

✉ Ji Zhang
zhangji@cs.cmu.edu; zhangji@cmu.edu

Sanjiv Singh
ssingh@cmu.edu

[1] Robotics Institute at Carnegie Mellon University, Pittsburgh, USA

1 Introduction

3D Mapping remains a popular technology. The main issue with laser ranging in which the laser moves has to do with registration of the resulting point cloud. If the only motion is the pointing of a laser beam with known internal kinematics of the lidar from a fixed base, this registration is obtained simply. However, if the sensor base moves, as in many applications of interest, laser point registration has to do with both the internal kinematics and external motion. The second one has to contain knowledge of how the sensor is located and oriented for every range measurement. Since lasers can measure distance up to several hundred thousand times per second, high-rate pose estimation is a significant issue. A common way to solve this problem is to use an independent method of pose estimation (such as with an accurate GPS/INS system) to register the range data into a coherent point cloud in reference to a fixed coordinate frame. When independent measurements relative to a fixed coordinate frame are unavailable, the general technique used is to register points using some sort of odometry estimation, e.g. using combinations of wheel motion, gyros, and by tracking features in range or visual images.

Here we consider the case of creating maps using low-drift odometry with a mechanically scanned laser ranging device (optionally augmented with low-grade inertial measurements) moving in 6-DOF. A key advantage of only using laser ranging is that it is not sensitive to ambient lighting or optical texture in the scene. New developments in laser scanners have reduced the size and weight of such devices to the level that they can be attached to mobile robots (including flying, walking or rolling) and even to people who move around in an environment to be mapped. Since we seek to push the odometry toward the lowest possible drift in real-time, we

Published online: 18 February 2016

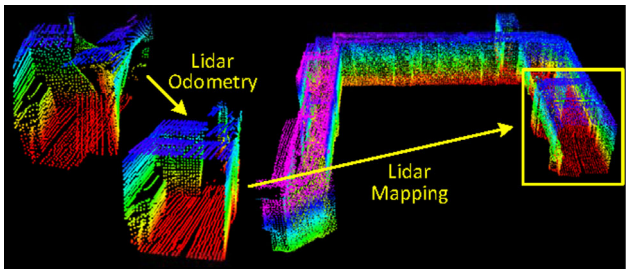


Fig. 1 The method aims at motion estimation and mapping using a moving 3D lidar. Since the laser points are received at different times, distortion is present in the point cloud due to motion of the lidar (shown in the left lidar cloud). Our proposed method decomposes the problem by two algorithms running in parallel. An odometry algorithm estimates velocity of the lidar and corrects distortion in the point cloud, then, a mapping algorithm matches and registers the point cloud to create a map. Combination of the two algorithms ensures feasibility of the problem to be solved in real-time

don't consider issues related to loop closure. Indeed, while loop closure could help further cancel the drift, we find that in many practical cases such as mapping a floor of a buildings, loop closure is unnecessary.

Our method, namely LOAM, achieves both low-drift in motion estimation in 6-DOF and low-computational complexity. The key idea that makes this level of performance possible is the division of the typically complex problem of simultaneous localization and mapping (illustrated in Fig. 1), which seeks to optimize a large number of variables simultaneously, into two algorithms. One algorithm performs odometry at a high-frequency but at low fidelity to estimate velocity of the laser scanner moving through the environment. Although not necessary, if an IMU is available, it can provide a motion prior and help account for gross, high-frequency motion. A second algorithm runs at an order of magnitude lower frequency for fine matching and registration of the point cloud. Specifically, both algorithms extract feature points located on edges and planar surfaces and match the feature points to edge-line segments and planar surface patches, respectively. In the odometry algorithm, correspondences of the feature points are found by ensuring fast computation, while in the mapping algorithm, by ensuring accuracy.

In the method, an easier problem is solved first as online velocity estimation, after which, mapping is conducted as batch optimization to produce high-precision motion estimation and maps. The parallel algorithm structure ensures feasibility of the problem to be solved in real-time. Further, since motion estimation is conducted at a higher frequency, mapping is given plenty of time to enforce accuracy. When staggered to run at an order of magnitude slower than the odometry algorithm, the mapping algorithm incorporates a large number of feature points and uses sufficiently many iterations to converge. The paper makes main contributions as follows,

- We propose a software system using dual-layer optimization to online estimate ego-motion and build maps;
- We carefully implement geometrical feature detection and matching to meet requirements of the system: feature matching in the odometry algorithm is coarse and fast to ensure high frequency, and is precise and slow in the mapping algorithm to ensure low-drift;
- We test the method thoroughly with a large number of datasets covering various types of environments;
- We make an honest attempt to present our work to a level of detail allowing readers to re-implement the method.

The rest of this paper is organized as follows. In Sect. 2, we discuss related work and how our work is unique compared to the state of the art. In Sect. 3, we pose the research problem formally. The lidar hardware and software systems are described in Sect. 4. The odometry algorithm is presented with details in Sect. 5, and the mapping algorithm in Sect. 6. Experimental results are shown in Sect. 7. Finally, a discussion and a conclusion are made in Sects. 7 and 8, respectively.

2 Related work

Lidar has become a useful range sensor in robot navigation. For localization and mapping, one way is to perform stop-and-scan to avoid motion distortion in point clouds (Nuchter et al. 2007). Also, when the lidar scanning rate is high compared to its extrinsic motion, motion distortion can be neglectable. In this case, ICP methods (Pomerleau et al. 2013) can be used to match laser returns between different scans. Additionally, a two-step method is proposed to remove the distortion (Hong et al. 2010): an ICP based velocity estimation step is followed by a distortion compensation step, using the computed velocity. A similar technique is also used to compensate for the distortion introduced by a single-axis 3D lidar (Moosmann and Stiller 2011). However, if the scanning motion is relatively slow, motion distortion can be severe. This is especially the case when a 2-axis lidar is used since one axis is typically much slower than the other. Often, other sensors are used to provide velocity measurements, with which, the distortion can be removed. For example, the lidar cloud can be registered by state estimation from visual odometry integrated with an IMU (Scherer et al. 2012). When multiple sensors such as a GPS/INS and wheel encoders are available concurrently, the problem is often solved through Kalman filers or particle filters, building maps in real-time.

If a 2-axis lidar is used without aiding from other sensors, motion estimation and distortion correction become one problem. In Barfoot et al.'s methods, the sensor motion is modeled as constant velocity (Dong and Barfoot 2012; Anderson and Barfoot 2013a) and with Gaussian processes

(Tong and Barfoot 2013; Anderson and Barfoot 2013b; Tong et al. 2013. Rosen et al. use Gaussian processes to model the continuous sensor motion and formulate the problem into a factor-graph optimization problem (Rosen et al. 2014). Additionally, Furgale et al. propose to use B-spline functions to model the sensor motion (Furgale et al. 2012). Our method uses a similar linear motion model as (Dong and Barfoot (2012); Anderson and Barfoot (2013a) in the odometry algorithm. In the mapping algorithm, however, rigid body transform is used. Another method is that of Bosse and Zlot (Bosse and Zlot 2009; Bosse et al. 2012; Zlot and Bosse 2012). They invent a 3D mapping device called Zebedee composed of a 2D lidar and an IMU attached to a hand-bar through a spring (Bosse et al. 2012). Mapping is conducted by hand-nodding the device. The trajectory is recovered by a batch optimization method that processes segmented datasets with boundary constraints added between the segments. In this method, measurements of the IMU are used to register the laser points and the optimization is used to correct the IMU drift and bias. Further, they use multiple 2-axis lidars to map an underground mine (Zlot and Bosse 2012). This method incorporates an IMU and uses loop closure to create large maps. The method runs faster than real-time. However, since it relies on batch processing to develop accurate maps, the method currently is hard to use in online applications to provide real-time state estimation and maps.

The same problem of motion distribution exists in vision-based state estimation. With a rolling-shutter camera, image pixels are perceived continuously over time, resulting in different read-out time for each pixel. The state-of-the-art visual odometry methods that deal with rolling-shutter effect benefit from an IMU (Guo et al. 2014; Li and Mourikis 2014). The methods use IMU mechanization to compensate for the motion given read-out time of the pixels. In this paper, we also have the option of using an IMU to cancel nonlinear motion, and the proposed method solves for linear motion.

From feature's perspective, Barfoot et al.'s methods (Dong and Barfoot 2012; Anderson and Barfoot 2013a, b; Tong and Barfoot 2013) create visual images from laser intensity returns and match visually distinct features (Bay et al. 2008) between images to recover motion. This requires dense point cloud with intensity values. On the other hand, Bosse and Zlot's method (Bosse and Zlot 2009; Bosse et al. 2012; Zlot and Bosse 2012) matches spatio-temporal patches formed of local point clusters. Our method has less requirement on point cloud density and does not require intensity values compared to Dong and Barfoot (2012), Anderson and Barfoot (2013a, b), and Tong and Barfoot (2013) since it extracts and matches geometric features in Cartesian space. It uses two types of point features, on edges and local planar surfaces, and matches them to edge line segments and local planar patches, respectively.

Our proposed method in real-time produces maps that are qualitatively similar to those by Bosse and Zlot. The distinction is that our method can provide motion estimates for guidance of an autonomous vehicle. The paper is an extended version of our conference paper (Zhang and Singh 2014). We evaluate the method with more experiments and present with more details.

3 Notations and task description

The problem addressed in this paper is to perform ego-motion estimation with point clouds perceived by a 3D lidar, and build a map for the traversed environment. We assume that the lidar is intrinsically calibrated with the lidar internal kinematics precisely known (the intrinsic calibration makes 3D projection of the laser points possible). We also assume that the angular and linear velocities of the lidar are smooth and continuous over time, without abrupt changes. The second assumption will be released by usage of an IMU, in Sects. 7.2 and 7.3.

As a convention in this paper, we use right uppercase superscription to indicate the coordinate systems. We define a sweep as the lidar completes one time of scan coverage. We use right subscription k, $k \in Z^+$ to indicate the sweeps, and $\mathcal{P}_k$ to indicate the point cloud perceived during sweep k. Let us define two coordinate systems as follows.

- Lidar coordinate system $\{L\}$ is a 3D coordinate system with its origin at the geometric center of the lidar (see Fig. 2). Here, we use the convention of cameras. The x-axis is pointing to the left, the y-axis is pointing upward, and the z-axis is pointing forward. We denote a point i received during sweep k as $\boldsymbol{X}^L_{(k,i)}$. Further, we use $\boldsymbol{T}^L_k(t)$ to denote the transform projecting a point received at time t to the beginning of the sweep k.
- World coordinate system $\{W\}$ is a 3D coordinate system coinciding with $\{L\}$ at the initial pose. We denote a point

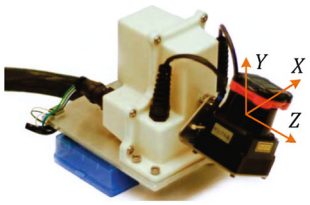


Fig. 2 An example 3D lidar using in experiment evaluation. We will use data from this sensor to illustrate the method. The sensor consists of a Hokuyo laser scanner driven by a motor for rotational motion, and an encoder that measures the rotation angle. The laser scanner has 180° field of view and 0.25° resolution. The scanning rate is 40 lines/s. The motor is controlled to rotate from −90° to 90° with the horizontal orientation of the laser scanner as zero

i in $\{W\}$ as $\boldsymbol{X}^W_{(k,i)}$ and denote $\boldsymbol{T}^W_k(t)$ as the transform projecting a point received at time t to $\{W\}$.

With assumptions and notations made, our lidar odometry and mapping problem can be defined as

Problem Given a sequence of lidar cloud $\mathcal{P}_k$, $k \in Z^+$, compute ego-motion of the lidar in the world, $\boldsymbol{T}^W_k(t)$, and build a map with $\mathcal{P}_k$ for the traversed environment.

4 System overview

4.1 Lidar hardware

The study of this paper is validated on four sensor systems: a back-and-forth spin lidar, a continuously-spinning lidar, a Velodyne HDL-32 lidar, and the sensor system used by the KITTI benchmark Geiger et al. (2012, 2013). We use the first lidar hardware as an example to illustrate the method, therefore we introduce the lidar hardware in the front of the paper to help readers understand the method. The rest sensors will be introduced in the experiment section. As shown in Fig. 2, the lidar is based on a Hokuyo UTM-30LX laser scanner which has 180° field of view with 0.25° resolution and 40 lines/s scanning rate. The laser scanner is connected to a motor controlled to rotate at 180°/s angular speed between −90 and 90° with the horizontal orientation of the laser scanner as zero. With this particular unit, a sweep is a rotation from −90 to 90° or in the inverse direction (lasting for 1 s). Here, note that for a continuously-spinning lidar, a sweep is simply a semi-spherical or a full-spherical rotation. An onboard encoder measures the motor rotation angle with 0.25° resolution, with which, the laser points are back-projected into the lidar coordinates, $\{L\}$.

4.2 Software system overview

Figure 3 shows a diagram of the software system. Let $\hat{\mathcal{P}}$ be the points received in a laser scan. During each sweep, $\hat{\mathcal{P}}$ is registered in $\{L\}$. The combined point cloud during sweep k forms $\mathcal{P}_k$. Then, $\mathcal{P}_k$ is processed in two algorithms. Lidar odometry takes the point cloud and computes the motion of the lidar between two consecutive sweeps. The estimated motion is used to correct distortion in $\mathcal{P}_k$. The algorithm runs at a frequency around 10 Hz. The outputs are further processed by lidar mapping, which matches and registers the undistorted cloud onto a map at a frequency of 1 Hz. Finally, the pose transforms published by the two algorithms are integrated to generate a transform output around 10 Hz, regarding the lidar pose with respect to the map. Sections 5 and 6 present the blocks in the software diagram in detail.

5 Lidar odometry

5.1 Feature point extraction

We start with extraction of feature points from the lidar cloud, $\mathcal{P}_k$. We notice that many 3D lidars naturally generate unevenly distributed points in $\mathcal{P}_k$. With the lidar in Fig. 2 as an example, the returns from the laser scanner has a resolution of 0.25° within a scan. These points are located on a scan plane. However, as the laser scanner rotates at an angular speed of 180°/s and generates scans at 40Hz, the resolution in the perpendicular direction to the scan planes is $180°/40 = 4.5°$. Considering this fact, the feature points are extracted from $\mathcal{P}_k$ using only information from individual scans, with co-planar geometric relationship.

We select feature points that are on sharp edges and planar surface patches. Let i be a point in $\mathcal{P}_k$, $i \in \mathcal{P}_k$, and let $\mathcal{S}$ be the set of consecutive points of i returned by the laser scanner in the same scan. Since the laser scanner generates point returns in CW or CCW order, $\mathcal{S}$ contains half of its points on each side of i and 0.25° intervals between two points (still with the lidar in Fig. 2 as an example). Define a term to evaluate the smoothness of the local surface,

$$c = \frac{1}{|\mathcal{S}| \cdot ||\boldsymbol{X}^L_{(k,i)}||} \left\| \sum_{j \in \mathcal{S}, j \neq i} (\boldsymbol{X}^L_{(k,i)} - \boldsymbol{X}^L_{(k,j)}) \right\|. \quad (1)$$

The term is normalized w.r.t. the distance to the lidar center. This is particularly made to remove scale effect and the term can be used for both near and far points.

The points in a scan are sorted based on the c values, then feature points are selected with the maximum c's, namely, edge points, and the minimum c's, namely planar points. To evenly distribute the feature points within the environment, we separate a scan into four identical subregions. Each subre-

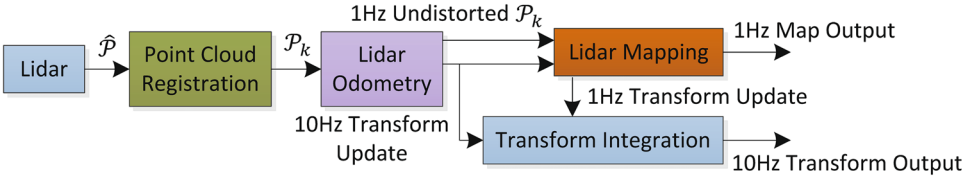


Fig. 3 Block diagram of the lidar odometry and mapping software system

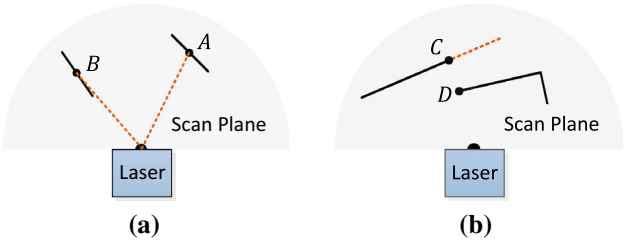


Fig. 4 **a** The *solid line segments* represent local surface patches. Point A is on a surface patch that has an angle to the laser beam (the *dotted orange line segments*). Point B is on a surface patch that is roughly parallel to the laser beam. We treat B as a unreliable laser return and do not select it as a feature point. **b** The *solid line segments* are observable objects to the laser. Point C is on the boundary of an occluded region (the *dotted orange line segment*), and can be detected as an edge point. However, if viewed from a different angle, the occluded region can change and become observable. We do not treat D as a salient edge point or select it as a feature point (Color figure online)

gion can provide maximally 2 edge points and 4 planar points. A point i can be selected as an edge or a planar point only if its c value is larger or smaller than a threshold (5×10^{-3}), and the number of selected points does not exceed the maximum point number of a subregion.

While selecting feature points, we want to avoid points whose surrounded points are selected, or points on local planar surfaces that are roughly parallel to the laser beams (point B in Fig. 4a). These points are usually considered as unreliable. Also, we want to avoid points that are on boundary of occluded regions (Li and Olson 2011). An example is shown in Fig. 4b. Point C is an edge point in the lidar cloud because its connected surface (the dotted line segment) is blocked by another object. However, if the lidar moves to another point of view, the occluded region can change and become observable. To avoid the aforementioned points to be selected, we find again the set of points $\mathcal{S}$. A point i can be selected only if $\mathcal{S}$ does not form a surface patch whose normal is within 10° to the laser beam, and there is no point in $\mathcal{S}$ that is disconnected from i by a gap in the direction of the laser beam and is at the same time closer to the lidar then point i (e.g. point B in Fig. 4b).

In summary, the feature points are selected as edge points starting from the maximum c value, and planar points starting from the minimum c value, and if a point is selected,

- The number of selected edge points or planar points cannot exceed the maximum of the subregion, and
- None of its surrounding point is already selected, and
- It cannot be on a surface patch whose normal is within 10° to the laser beam, or on boundary of an occluded region.

An example of extracted feature points from a corridor scene is shown in Fig. 5. The edge points and planar points are labeled in yellow and red colors, respectively.

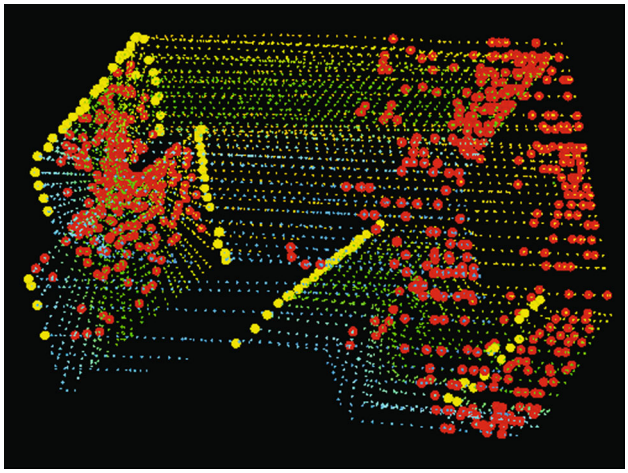


Fig. 5 An example of extracted edge points (*yellow*) and planar points (*red*) from lidar cloud taken in a corridor. Meanwhile, the lidar moves toward the wall on the *left side* of the figure at a speed of 0.5 m/s, this results in motion distortion on the wall (Color figure online)

5.2 Finding feature point correspondence

The odometry algorithm estimates motion of the lidar within a sweep. Let t_k be the starting time of a sweep k. At the end of sweep $k-1$, the point cloud perceived during the sweep, $\mathcal{P}_{k-1}$, is projected to time stamp t_k, illustrated in Fig. 6 (we will discuss transforms projecting the points in Sect. 5.3). We denote the projected point cloud as $\bar{\mathcal{P}}_{k-1}$. During the next sweep k, $\bar{\mathcal{P}}_{k-1}$ is used together with the newly received point cloud, $\mathcal{P}_k$, to estimate the motion of the lidar.

Let us assume that both $\bar{\mathcal{P}}_{k-1}$ and $\mathcal{P}_k$ are available for now, and start with finding correspondences between the two lidar clouds. With $\mathcal{P}_k$, we find edge points and planar points from the lidar cloud using the methodology discussed in the last section. Let $\mathcal{E}_k$ and $\mathcal{H}_k$ be the sets of edge points and planar points, respectively. We will find edge lines from $\bar{\mathcal{P}}_{k-1}$ as correspondences of the points in $\mathcal{E}_k$, and planar patches as correspondences of those in $\mathcal{H}_k$.

Note that at the beginning of sweep k, $\mathcal{P}_k$ is an empty set, which grows during the course of the sweep as more

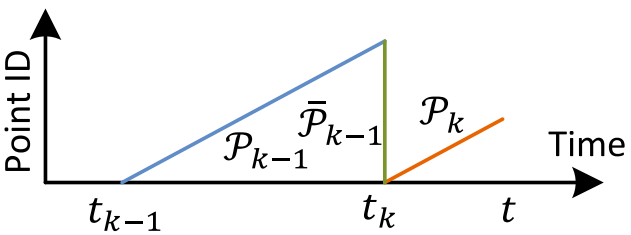
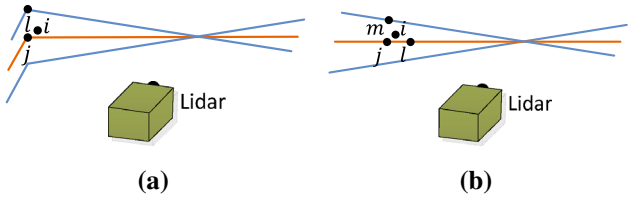


Fig. 6 Project point cloud to the end of a sweep. The *blue colored line segment* represents the point cloud perceived during sweep k, $\mathcal{P}_{k-1}$. At the end of sweep $k-1$, $\mathcal{P}_{k-1}$ is projected to time stamp t_k to obtain $\bar{\mathcal{P}}_{k-1}$ (the *green colored line segment*). Then, during sweep k, $\bar{\mathcal{P}}_{k-1}$ and the newly perceived point cloud $\mathcal{P}_k$ (the orange colored line segment) are used together to estimate the lidar motion (Color figure online)



(a) **(b)**

Fig. 7 Finding an edge line as the correspondence for an edge point in $\tilde{\mathcal{E}}_k$ (**a**), and a planar patch as the correspondence for a planar point in $\tilde{\mathcal{H}}_k$ (**b**). In both (**a**, **b**), j is the closest point to the feature point i, found in $\bar{\mathcal{P}}_{k-1}$. The *orange lines* represent the same scan of j, and the *blue lines* are the preceding and following scans. To find the edge line correspondence in **a**, we find another point, l, on the *blue lines*, and the correspondence is represented as $(j,\ l)$. To find the planar patch correspondence in **b**, we find another two points, l and m, on the orange line and the *blue line*, respectively. The correspondence is $(j,\ l,\ m)$ (Color figure online)

points are received. Lidar odometry recursively estimates the 6-DOF motion during the sweep, and gradually includes more points as $\mathcal{P}_k$ increases. $\mathcal{E}_k$ and $\mathcal{H}_k$ are projected to the beginning of the sweep (again, we will discuss transforms projecting the points later). Let $\tilde{\mathcal{E}}_k$ and $\tilde{\mathcal{H}}_k$ be the projected point sets. For each point in $\tilde{\mathcal{E}}_k$ and $\tilde{\mathcal{H}}_k$, we are going to find the closest neighbor point in $\bar{\mathcal{P}}_{k-1}$. Here, $\bar{\mathcal{P}}_{k-1}$ is stored in a 3D KD-tree (Berg et al. 2008) in $\{L_k\}$ for fast index.

Figure 7a represents the procedure of finding an edge line as the correspondence of an edge point. Let i be a point in $\tilde{\mathcal{E}}_k$, $i \in \tilde{\mathcal{E}}_k$. The edge line is represented by two points. Let j be the closest neighbor of i in $\bar{\mathcal{P}}_{k-1}$, $j \in \bar{\mathcal{P}}_{k-1}$, and let l be the closest neighbor of i in the preceding and following two scans to the scan of j. $(j,\ l)$ forms the correspondence of i. Then, to verify both j and l are edge points, we check the smoothness of the local surface based on (1) and require that both points have $c > 5 \times 10^{-3}$. Here, we particularly require that j and l are from different scans considering that a single scan cannot contain more than one points from the same edge line. There is only one exception where the edge line is on the scan plane. If so, however, the edge line is degenerated and appears as a straight line on the scan plane, and feature points on the edge line should not be extracted in the first place.

Figure 7b shows the procedure of finding a planar patch as the correspondence of a planar point. Let i be a point in $\tilde{\mathcal{H}}_k$, $i \in \tilde{\mathcal{H}}_k$. The planar patch is represented by three points. Similar to the last paragraph, we find the closest neighbor of i in $\bar{\mathcal{P}}_{k-1}$, denoted as j. Then, we find another two points, l and m, as the closest neighbors of i, one in the same scan of j but not j, and the other in the preceding and following scans to the scan of j. This guarantees that the three points are non-collinear. To verify that j, l, and m are all planar points, again, we check the smoothness of the local surface and require $c < 5 \times 10^{-3}$.

With the correspondences of the feature points found, now we derive expressions to compute the distance from a feature point to its correspondence. We will recover the lidar motion by minimizing the overall distances of the feature points in the next section. We start with edge points. For a point $i \in \tilde{\mathcal{E}}_k$, if $(j,\ l)$ is the corresponding edge line, $j, l \in \bar{\mathcal{P}}_{k-1}$, the point to line distance can be computed as

$$d_{\mathcal{E}} = \frac{\left|(\tilde{\boldsymbol{X}}^L_{(k,i)} - \bar{\boldsymbol{X}}^L_{(k-1,j)}) \times (\tilde{\boldsymbol{X}}^L_{(k,i)} - \bar{\boldsymbol{X}}^L_{(k-1,l)})\right|}{\left|\bar{\boldsymbol{X}}^L_{(k-1,j)} - \bar{\boldsymbol{X}}^L_{(k-1,l)}\right|}, \tag{2}$$

where $\tilde{\boldsymbol{X}}^L_{(k,i)}$, $\bar{\boldsymbol{X}}^L_{(k-1,j)}$, and $\bar{\boldsymbol{X}}^L_{(k-1,l)}$ are the coordinates of points i, j, and l in $\{L_k\}$, respectively. Then, for a point $i \in \tilde{\mathcal{H}}_k$, if $(j,\ l,\ m)$ is the corresponding planar patch, $j, l, m \in \bar{\mathcal{P}}_{k-1}$, the point to plane distance is

$$d_{\mathcal{H}} = \frac{\left|(\tilde{\boldsymbol{X}}^L_{(k,i)} - \bar{\boldsymbol{X}}^L_{(k-1,j)}) \cdot ((\bar{\boldsymbol{X}}^L_{(k-1,j)} - \bar{\boldsymbol{X}}^L_{(k-1,l)}) \times (\bar{\boldsymbol{X}}^L_{(k-1,j)} - \bar{\boldsymbol{X}}^L_{(k-1,m)}))\right|}{\left|(\bar{\boldsymbol{X}}^L_{(k-1,j)} - \bar{\boldsymbol{X}}^L_{(k-1,l)}) \times (\bar{\boldsymbol{X}}^L_{(k-1,j)} - \bar{\boldsymbol{X}}^L_{(k-1,m)})\right|}. \tag{3}$$

5.3 Motion estimation

The lidar motion is modeled with constant angular and linear velocities during a sweep. This allows us to linear interpolate the pose transform within a sweep for the points that are received at different times. Let t be the current time stamp, and recall that t_k is the starting time of the current sweep k. Let $\boldsymbol{T}^L_k(t)$ be the lidar pose transform between $[t_k,\ t]$. $\boldsymbol{T}^L_k(t)$ contains 6-DOF motion of the lidar, $\boldsymbol{T}^L_k(t) = [\tau^L_k(t),\ \theta^L_k(t)]^T$, where $\tau^L_k(t) = [t_x,\ t_y,\ t_z]^T$ is the translation and $\theta^L_k(t) = [\theta_x,\ \theta_y,\ \theta_z]^T$ is the rotation in $\{L_k\}$. Given $\theta^L_k(t)$, the corresponding rotation matrix can be defined by the Rodrigues formula (Murray and Sastry 1994),

$$\begin{aligned}\mathbf{R}^L_k(t) = e^{\hat{\theta}^L_k(t)} = \mathbf{I} + \frac{\hat{\theta}^L_k(t)}{||\theta^L_k(t)||}\sin||\theta^L_k(t)|| \\ + \left(\frac{\hat{\theta}^L_k(t)}{||\theta^L_k(t)||}\right)^2 (1 - ||\cos\theta^L_k(t)||).\end{aligned} \tag{4}$$

where $\hat{\theta}^L_k(t)$ is the skew symmetric matrix of $\theta^L_k(t)$.

Given a point i, $i \in \mathcal{P}_k$, let $t_{(k,i)}$ be its time stamp, and let $\boldsymbol{T}^L_{(k,i)}$ be the pose transform between $[t_k,\ t_{(k,i)}]$. $\boldsymbol{T}^L_{(k,i)}$ can be computed by linear interpolation of $\boldsymbol{T}^L_k(t)$,

$$\boldsymbol{T}^L_{(k,i)} = \frac{t_{(k,i)} - t_k}{t - t_k}\boldsymbol{T}^L_k(t). \tag{5}$$

Here, note that $\boldsymbol{T}^L_k(t)$ is a changing variable over time and the interpolation uses the transform of current time t. Recall that $\mathcal{E}_k$ and $\mathcal{H}_k$ are the sets of edge points and planar points

extracted from $\mathcal{P}_k$. The following equation helps project $\mathcal{E}_k$ and $\mathcal{H}_k$ to the beginning of the sweep, namely $\tilde{\mathcal{E}}_k$ and $\tilde{\mathcal{H}}_k$,

$$\tilde{\boldsymbol{X}}^L_{(k,i)} = \mathbf{R}^L_{(k,i)} \boldsymbol{X}^L_{(k,i)} + \tau^L_{(k,i)}, \tag{6}$$

where $\boldsymbol{X}^L_{(k,i)}$ is a point in $\mathcal{E}_k$ or $\mathcal{H}_k$ and $\tilde{\boldsymbol{X}}^L_{(k,i)}$ is the corresponding point in $\tilde{\mathcal{E}}_k$ or $\tilde{\mathcal{H}}_k$. $\mathbf{R}^L_{(k,i)}$ and $\tau^L_{(k,i)}$ are the rotation matrix and translation vector corresponding to $\boldsymbol{T}^L_{(k,i)}$.

Recall that (2) and (3) compute the distances between points in $\tilde{\mathcal{E}}_k$ and $\tilde{\mathcal{H}}_k$ and their correspondences. Combining (2) and (6), we can derive a geometric relationship between an edge point in $\mathcal{E}_k$ and the corresponding edge line,

$$f_{\mathcal{E}}(\boldsymbol{X}^L_{(k,i)}, \boldsymbol{T}^L_k(t)) = d_{\mathcal{E}}, \quad i \in \mathcal{E}_k. \tag{7}$$

Similarly, combining (3) and (6), we can establish another geometric relationship between a planar point in $\mathcal{H}_k$ and the corresponding planar patch,

$$f_{\mathcal{H}}(\boldsymbol{X}^L_{(k,i)}, \boldsymbol{T}^L_k(t)) = d_{\mathcal{H}}, \quad i \in \mathcal{H}_k. \tag{8}$$

Finally, we solve the lidar motion with the Levenberg-Marquardt method (Hartley and Zisserman 2004). Stacking (7) and (8) for each feature point in $\mathcal{E}_k$ and $\mathcal{H}_k$, we obtain a nonlinear function,

$$\boldsymbol{f}(\boldsymbol{T}^L_k(t)) = \boldsymbol{d}, \tag{9}$$

where each row of $\boldsymbol{f}$ corresponds to a feature point, and $\boldsymbol{d}$ contains the corresponding distances. We compute the Jacobian matrix of $\boldsymbol{f}$ with respect to $\boldsymbol{T}^L_k(t)$, denoted as $\mathbf{J}$, where $\mathbf{J} = \partial f / \partial \boldsymbol{T}^L_k(t)$. Then, (9) can be solved through nonlinear iterations by minimizing $\boldsymbol{d}$ toward zero,

$$\boldsymbol{T}^L_k(t) \leftarrow \boldsymbol{T}^L_k(t) - (\mathbf{J}^T\mathbf{J} + \lambda \mathrm{diag}(\mathbf{J}^T\mathbf{J}))^{-1}\mathbf{J}^T\boldsymbol{d}. \tag{10}$$

λ is a factor determined by the Levenberg-Marquardt method.

5.4 Lidar odometry algorithm

Lidar odometry algorithm is shown in Algorithm 1. The algorithm takes as inputs the point cloud from the last sweep, $\bar{\mathcal{P}}_{k-1}$, the growing point cloud of the current sweep, $\mathcal{P}_k$, and the pose transform from the last recursion as initial guess, $\boldsymbol{T}^L_k(t)$. If a new sweep is started, $\boldsymbol{T}^L_k(t)$ is set to zero to re-initialize (line 4–6). Then, the algorithm extracts feature points from $\mathcal{P}_k$ to construct $\mathcal{E}_k$ and $\mathcal{H}_k$ on line 7. For each feature point, we find its correspondence in $\bar{\mathcal{P}}_{k-1}$ (line 9–19). The motion estimation is adapted to a robust fitting framework (Andersen 2008). On line 15, the algorithm assigns a bisquare weight for each feature point as the following equation. The feature points that have larger distances to their correspondences are assigned with smaller weights, and the feature points with distances larger than a threshold are considered as outliers and assigned with zero weights.

Algorithm 1: Lidar Odometry

1 **input** : $\bar{\mathcal{P}}_{k-1}$, $\mathcal{P}_k$, $\boldsymbol{T}^L_k(t)$ from the last recursion at initial guess
2 **output** : $\bar{\mathcal{P}}_k$, newly computed $\boldsymbol{T}^L_k(t)$
3 **begin**
4 **if** at the beginning of a sweep **then**
5 $\boldsymbol{T}^L_k(t) \leftarrow \mathbf{0}$;
6 **end**
7 Detect edge points and planar points in $\mathcal{P}_k$, put the points in $\mathcal{E}_k$ and $\mathcal{H}_k$, respectively;
8 **for** a number of iterations **do**
9 **for** each edge point in $\mathcal{E}_k$ **do**
10 Find an edge line as the correspondence, then compute point to line distance based on (7) and stack the equation to (9);
11 **end**
12 **for** each planar point in $\mathcal{H}_k$ **do**
13 Find a planar patch as the correspondence, then compute point to plane distance based on (8) and stack the equation to (9);
14 **end**
15 Compute a bisquare weight for each row of (9);
16 Update $\boldsymbol{T}^L_k(t)$ for a nonlinear iteration based on (10);
17 **if** the nonlinear optimization converges **then**
18 Break;
19 **end**
20 **end**
21 **if** at the end of a sweep **then**
22 Project each point in $\mathcal{P}_k$ to t_{k+1} and form $\bar{\mathcal{P}}_k$;
23 Return $\boldsymbol{T}^L_k(t)$ and $\bar{\mathcal{P}}_k$;
24 **end**
25 **else**
26 Return $\boldsymbol{T}^L_k(t)$;
27 **end**
28 **end**

$$w = \begin{cases} (1-\alpha^2)^2 & -1 < \alpha < 1, \\ 0 & \text{otherwise}, \end{cases} \tag{11}$$

where

$$\alpha = \frac{r}{6.9459\sigma\sqrt{1-h}}.$$

In the above equation, r is the corresponding residual in the least square problem, σ is the absolute deviation of the residuals from the median, and h is the leverage value or the corresponding element on the diagonal of matrix $\mathbf{J}(\mathbf{J}^T\mathbf{J}))^{-1}\mathbf{J}^T$ where $\mathbf{J}$ is the same Jacobian matrix used in (10). Then, on line 16, the pose transform is updated for one iteration. The nonlinear optimization terminates if convergence is found, or the maximum iteration number is met. If the algorithm reaches the end of a sweep, $\mathcal{P}_k$ is projected to time stamp t_{k+1} using the estimated motion during the sweep, forming $\bar{\mathcal{P}}_k$. This makes ready for the next sweep to

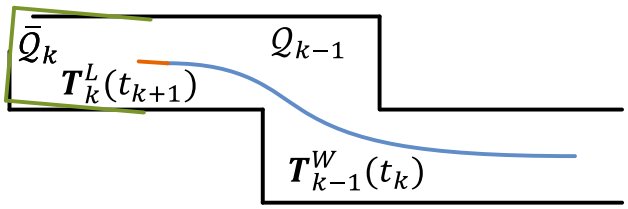


Fig. 8 Illustration of mapping process. The *blue curve* represents the lidar pose on the map, $\boldsymbol{T}_{k-1}^W(t_k)$, generated by the mapping algorithm at sweep $k-1$. The *orange curve* indicates the lidar motion during the entire sweep k, $\boldsymbol{T}_k^L(t_{k+1})$, computed by the odometry algorithm. With $\boldsymbol{T}_{k-1}^W(t_k)$ and $\boldsymbol{T}_k^L(t_{k+1})$, the undistorted point cloud published by the odometry algorithm is projected onto the map, denoted as $\bar{\mathcal{Q}}_k$ (the *green line segments*), and matched with the existing cloud on the map, $\mathcal{Q}_{k-1}$ (the *black colored line segments*) (Color figure online)

be matched to $\bar{\mathcal{P}}_k$. Otherwise, only the transform $\boldsymbol{T}_k^L(t)$ is returned by the algorithm for the next round of recursion.

6 Lidar mapping

The mapping algorithm runs at a lower frequency then the odometry algorithm, and is called only once per sweep. At the end of sweep k, lidar odometry generates a undistorted point cloud, $\bar{\mathcal{P}}_k$, and simultaneously a pose transform, $\boldsymbol{T}_k^L(t_{k+1})$, containing the lidar motion during the sweep, between $[t_k,\ t_{k+1}]$. The mapping algorithm matches and registers $\bar{\mathcal{P}}_k$ in the world coordinates, $\{W\}$, illustrated in Fig. 8. To explain the procedure, let us define $\mathcal{Q}_{k-1}$ as the point cloud on the map, accumulated until sweep $k-1$, and let $\boldsymbol{T}_{k-1}^W(t_k)$ be the pose of the lidar on the map at the end of sweep $k-1$, t_k. With the output from lidar odometry, the mapping algorithm extents $\boldsymbol{T}_{k-1}^W(t_k)$ for one sweep from t_k to t_{k+1}, to obtain $\boldsymbol{T}_k^W(t_{k+1})$, and transforms $\bar{\mathcal{P}}_k$ into the world coordinates, $\{W\}$, denoted as $\bar{\mathcal{Q}}_k$. Next, the algorithm matches $\bar{\mathcal{Q}}_k$ with $\mathcal{Q}_{k-1}$ by optimizing the lidar pose $\boldsymbol{T}_k^W(t_{k+1})$.

The feature points are extracted in the same way as in Sect. 5.1, but 10 times of feature points are used. To find correspondences for the feature points, we store the point cloud on the map, $\mathcal{Q}_{k-1}$, in 10 m cubic areas. The points in the cubes that intersect with $\bar{\mathcal{Q}}_k$ are extracted and stored in a 3D KD-tree (Berg et al. 2008) in $\{W\}$. We find the points in $\mathcal{Q}_{k-1}$ within a certain region ($10\,\text{cm} \times 10\,\text{cm} \times 10\,\text{cm}$) around the feature points. Let $\mathcal{S}'$ be a set of surrounding points. For an edge point, we only keep points on edge lines in $\mathcal{S}'$, and for a planar point, we only keep points on planar patches. The points are distinguished between edge points and planar points based on their c values. Here, we use the same threshold (5×10^{-3}) as in Sect. 5.1. Then, we compute the covariance matrix of $\mathcal{S}'$, denoted as $\mathbf{M}$, and the eigenvalues and eigenvectors of $\mathbf{M}$, denoted as $\boldsymbol{V}$ and $\boldsymbol{E}$, respectively. These values determine poses of the point clusters and hence the point-to-line and point-to-plane distances. Specifically, if $\mathcal{S}'$ is distributed on an edge line, $\boldsymbol{V}$ contains one eigenvalue

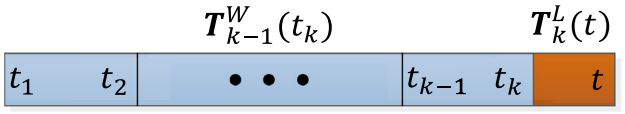


Fig. 9 Integration of pose transforms. The *blue colored region* illustrates the lidar pose from the mapping algorithm, $\boldsymbol{T}_{k-1}^W(t_k)$, generated once per sweep. The *orange colored region* is the lidar motion within the current sweep, $\boldsymbol{T}_k^L(t)$, computed by the odometry algorithm. The motion estimation of the lidar is the combination of the two transforms, at the same frequency as $\boldsymbol{T}_k^L(t)$ (Color figure online)

significantly larger than the other two, and the eigenvector in $\boldsymbol{E}$ associated with the largest eigenvalue represents the orientation of the edge line. On the other hand, if $\mathcal{S}'$ is distributed on a planar patch, $\boldsymbol{V}$ contains two large eigenvalues with the third one significantly smaller, and the eigenvector in $\boldsymbol{E}$ associated with the smallest eigenvalue denotes the orientation of the planar patch. The position of the edge line or the planar patch is calculated such that the line or the plane passes through the centroid of $\mathcal{S}'$.

To compute the distance from a feature point to its correspondence, we select two points on an edge line, and three points on a planar patch. This allows the distances to be computed using the same formulations as (2) and (3). Then, an equation is derived for each feature point as (7) or (8), but different in that all points in $\bar{\mathcal{Q}}_k$ share the same time stamp, t_{k+1}. The nonlinear optimization is solved again by the Levenberg-Marquardt method (Hartley and Zisserman 2004) adapted to robust fitting (Andersen 2008), and then $\bar{\mathcal{Q}}_k$ is registered on the map.

To evenly distribute the points, the map cloud is downsized by voxel-grid filters (Rusu and Cousins 2011) each time a new scan is merged with the map. The voxel-grid filters average all points in each voxel, leaving an averaged point in the voxel. Edge points and planar points use different voxel sizes. With edge points, the voxel size is $5\,\text{cm} \times 5\,\text{cm} \times 5\,\text{cm}$. With planar points, it is $10\,\text{cm} \times 10\,\text{cm} \times 10\,\text{cm}$. The map is truncated in a $500\,\text{m} \times 500\,\text{m} \times 500\,\text{m}$ region surrounding the sensor to limit the memory usage.

Integration of the pose transforms is illustrated in Fig. 9. The blue colored region represents the pose output from lidar mapping, $\boldsymbol{T}_{k-1}^W(t_k)$, generated once per sweep. The orange colored region represents the transform output from lidar odometry, $\boldsymbol{T}_k^L(t)$, at a frequency round 10Hz. The lidar pose with respect to the map is the combination of the two transforms, at the same frequency as lidar odometry.

7 Experiments

During experiments, the algorithms processing the lidar data run on a laptop computer with 2.5 GHz quad cores and 6Gib memory, on top of the robot operating system (ROS) (Quigley et al. 2009) in Linux. The method consumes a total of two

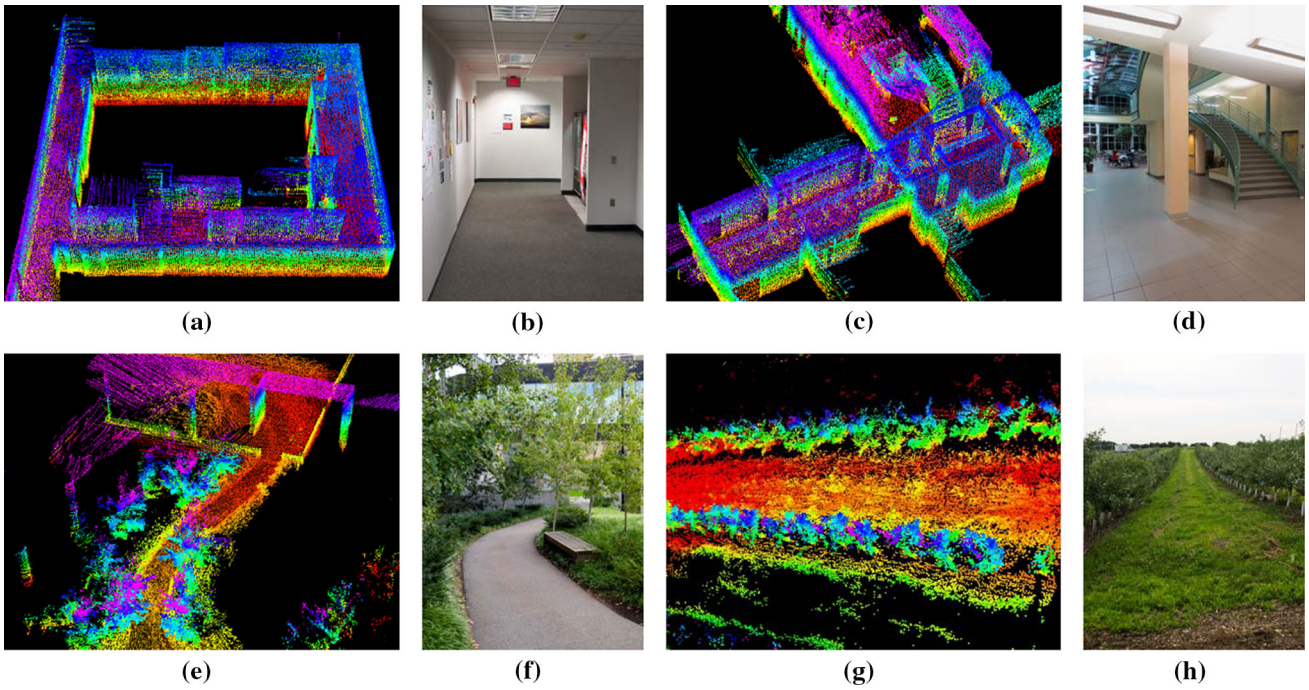


Fig. 10 Maps generated in **a**, **b** a narrow and long corridor, **c**, **d** a large lobby, **e**, **f** a vegetated road, and **g**, **h** an orchard between two rows of trees. The lidar is placed on a cart in indoor tests, and mounted on a ground vehicle in outdoor tests. All tests use a speed of 0.5m/s

threads, the odometry and mapping programs run on two separate threads.

7.1 Accuracy tests

The method has been tested in indoor and outdoor environments using the lidar in Fig. 2. During indoor tests, the lidar is placed on a cart together with a battery and a laptop computer. One person pushes the cart and walks. Figure 10a, c show maps built in two representative indoor environments, a narrow and long corridor and a large lobby. Figure 10b, d show two photos taken from the same scenes. In outdoor tests, the lidar is mounted to the front of a ground vehicle. Figrue 10e, g show maps generated from a vegetated road and an orchard between two rows of trees, and photos are presented in Fig. 10f, h, respectively. During all tests, the lidar moves at a speed of 0.5 m/s.

To evaluate local accuracy of the maps, we collect a second set of lidar clouds from the same environments. The lidar is kept stationary and placed at a few different places in each environment during data selection. The two point clouds are matched and compared using the point to plane ICP method (Rusinkiewicz and Levoy 2001). After matching is complete, the distances between one point cloud and the corresponding planar patches in the second point cloud are considered as matching errors. Figure 11 shows the density of error distributions. It indicates smaller matching errors in indoor environments than in outdoor. The result is reasonable because feature matching in natural environments is less exact than in manufactured environments.

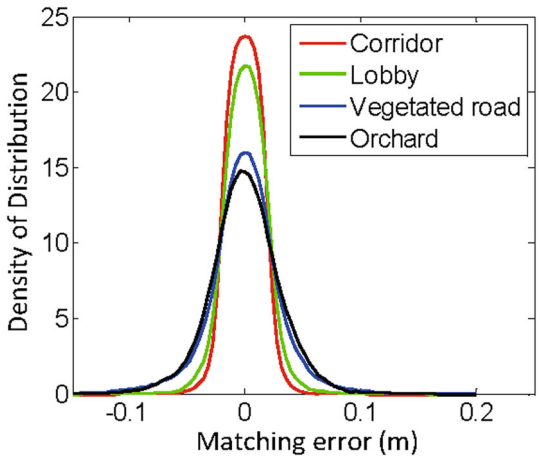


Fig. 11 Matching errors for corridor (*red*), lobby (*green*), vegetated road (*blue*) and orchard (*black*), corresponding to the four scenes in Fig. 10 (Color figure online)

Further, we want to understand how lidar odometry and lidar mapping function and contribute to the final accuracy. To this end, we take the dataset in Fig. 1 and show output of each algorithm. The trajectory is 32m in length. Figure 12a uses lidar odometry output to register laser points directly, while Fig. 12b is the final output optimized by lidar mapping. As we mention at the beginning, the role of lidar odometry is to estimate velocity and remove motion distortion in point

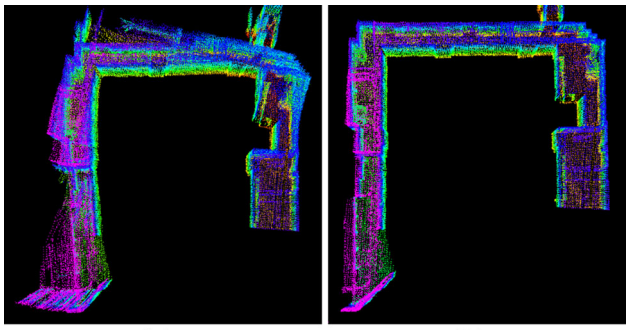


Fig. 12 Comparison between **a** lidar odometry output and **b** final lidar mapping output with the dataset in Fig. 1. The role of lidar odometry is to estimate velocity and remove motion distortion in point clouds. This algorithm has a low fidelity. Lidar mapping further performs careful scan matching to warrant accuracy on the map

clouds. The low-fidelity of lidar odometry cannot warrant accurate mapping. On the other hand, lidar mapping further performs careful scan matching to warrant accuracy on the map. Table 1 shows computation time brake-down of the two programs in the accuracy tests. We see lidar mapping takes totally 6.4 times of computation of lidar odometry to remove drift. Here, note that lidar odometry is called 10 times while lidar mapping is called once, resulting in same level of computation load on the two threads.

Additionally, we conduct tests to measure accumulated drift of the motion estimate. We choose corridor for indoor experiments that contains a closed loop. This allows us to start and finish at the same place. The motion estimation generates a gap between the starting and finishing positions, which indicates the amount of drift. For outdoor experiments, we choose orchard environment. The ground vehicle that carries the lidar is equipped with a high accuracy GPS/INS for ground truth acquisition. The measured drifts are compared to the distance traveled as the relative accuracy, and listed in Table 2. Specifically, Test 1 uses the same datasets with Fig. 10a, g. In general, the indoor tests have a relative accuracy around 1 % and the outdoor tests are around 2.5 %.

7.2 Tests with IMU assistance

We attach an Xsens MTi-10 IMU to the lidar to deal with fast velocity changes. The point cloud is pre-processed in two ways before sending to the proposed method, (1) with orientation from the IMU, the point cloud received in one sweep is rotated to align with the initial orientation of the lidar in that sweep, (2) with acceleration measurement, the motion distortion is partially removed as if the lidar moves at a constant velocity during the sweep. Here, the IMU orientation is obtained by integrating angular rates from gyros and readings from accelerometers in a Kalman filter (Thrun et al. 2005). After IMU pre-processing, the motion left to solve is the orientation drift from the IMU, assumed to be linear within a sweep, and the linear velocity. Hence, it satisfies the assumption that the lidar has linear motion within a sweep. The point cloud is then processed by the lidar odometry and mapping programs.

Table 1 Computation break-down for accuracy tests

Program	Build	Match	Others (ms)	Total (ms)
	KD-tree (ms)	Features (ms)		
Odometry	11	23	14	48
Mapping	58	134	117	309

Table 2 Relative errors for motion estimation drift

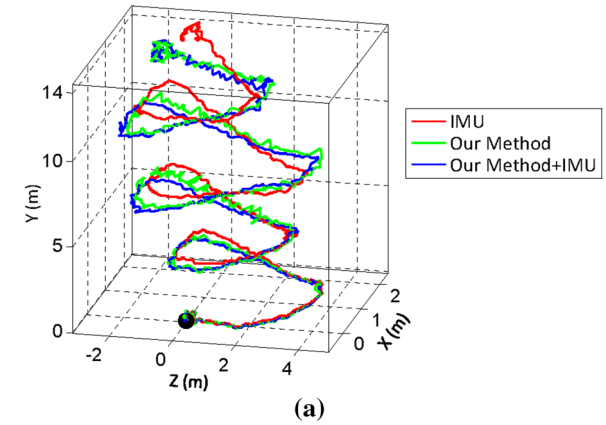
Environment	Test 1		Test 2	
	Distance (m)	Error (%)	Distance (m)	Error (%)
Corridor	58	0.9	46	1.1
Orchard	52	2.3	67	2.8

Figure 13a shows a sample result. A person holds the lidar and walks on a staircase. When computing the red curve, we use orientation provided by the IMU, and our method only estimates translation. The orientation drifts over 25° during 5 mins of data collection. The green curve relies only on the optimization in our method, assuming no IMU is available. The blue curve uses the IMU data for preprocessing followed by the proposed method. We observe small difference between the green and blue curves. Figure 13b presents the map corresponding to the blue curve. In Fig. 13c, we compare two closed views of the maps in the yellow rectangular in Fig. 13b. The upper and lower figures correspond to the blue and green curves, respectively. Careful comparison finds that the edges in the upper figure are sharper than those in the lower figure.

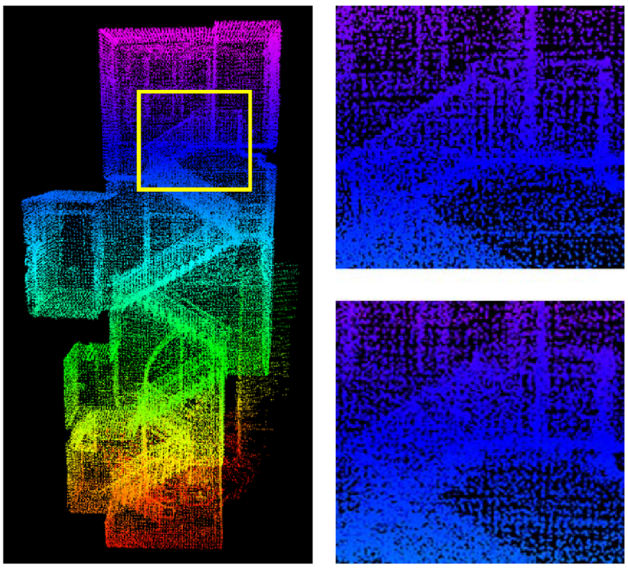
Table 3 compares relative errors in motion estimation with and without using the IMU. The lidar is held by a person walking at a speed of 0.5 m/s and moving the lidar up and down at a magnitude around 0.5 m. The ground truth is manually measured by a tape ruler. In all four tests, using the proposed method with assistance from the IMU gives the highest accuracy, while using orientation from the IMU only leads to the lowest accuracy. The results indicate that the IMU is effective in canceling the nonlinear motion, with which, the proposed method handles the linear motion.

7.3 Tests with micro-helicopter datasets

We further evaluate the method with data collected from an octo-rotor micro aerial vehicle. As shown in Fig. 14, the helicopter is mounted with a 2-axis lidar, which shares the same design with the one in Fig. 2 except that the laser scanner



(a)



(b) (c)

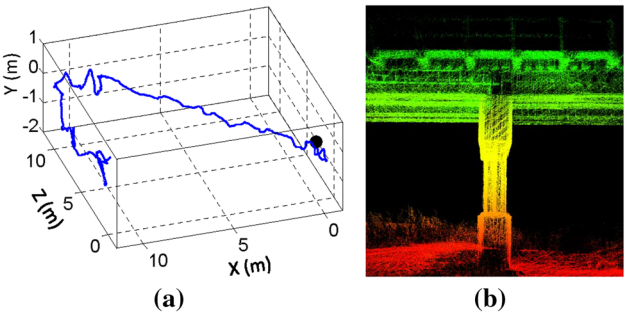
Fig. 13 Comparison of results with/without aiding from an IMU. A person holds the lidar and walks on a staircase. The *black dot* is the starting point. In **a**, the *red curve* is computed using orientation from the IMU and translation estimated by our method, the green curve relies on the optimization in our method only, and the blue curve uses the IMU data for preprocessing followed by the method. **b** is the map corresponding to the *blue curve*. In **c**, the upper and lower figures correspond to the *blue* and *green curves*, respectively, using the region labeled by the *yellow rectangle* in **b**. The edges in the *upper figure* are sharper, indicating more accuracy on the map (Color figure online)

Table 3 Motion estimation errors with/without using IMU

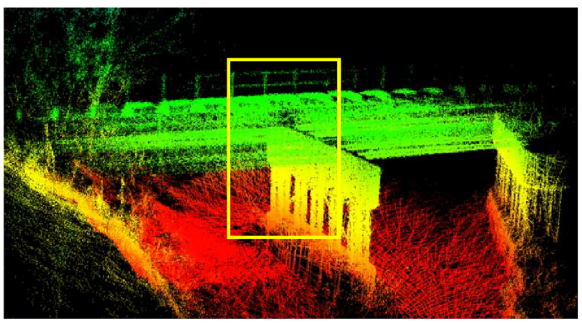
Environment	Distance (m)	Error		
		IMU (%)	Ours (%)	Ours+IMU (%)
Corridor	32	16.7	2.1	0.9
Lobby	27	11.7	1.7	1.3
Vegetated road	43	13.7	4.4	2.6
Orchard	51	11.4	3.7	2.1

(a) (b)

Fig. 14 **a** Octo-rotor helicopter used in the study. A 2-axis lidar is mounted to the font of the helicopter with a zoomed in view in **b**. The lidar is based on a Hokuyo laser scanner, sharing the same design with the one in Fig. 2 except the laser scanner spins continuously



(a) (b)



(c)

Fig. 15 Results from a small bridge. **a** shows trajectory of the helicopter. The *black dot* is the starting position. **b**, **c** show the map built by the proposed method, where **b** is a zoomed in view of the area inside the *yellow rectangle* in **c**. The helicopter is manually flown during data collection. Starting from one side, it flies underneath the bridge, turns back, and flies underneath the bridge again (Color figure online)

spins continuously. For such a lidar unit, a sweep is defined as a semi-spherical rotation on the slow axis, lasting for one second. A Microstrain 3DM-GX3-45 IMU is also mounted on the helicopter. The odometry and mapping programs process both lidar and IMU data.

We show results from two datasets in Figs. 15 and 16. For both tests, the helicopter is manually flown at a speed of 1 m/s. In Fig. 15, the helicopter starts from one side of the bridge, flies underneath the bridge and turns back to fly underneath the bridge for the second time. In Fig. 16, the helicopter starts with taking-off from the ground and ends with landing back on the ground. In Figs. 15a and 16a, we show trajectories of the flights, and in Figs. 15b and 16b,

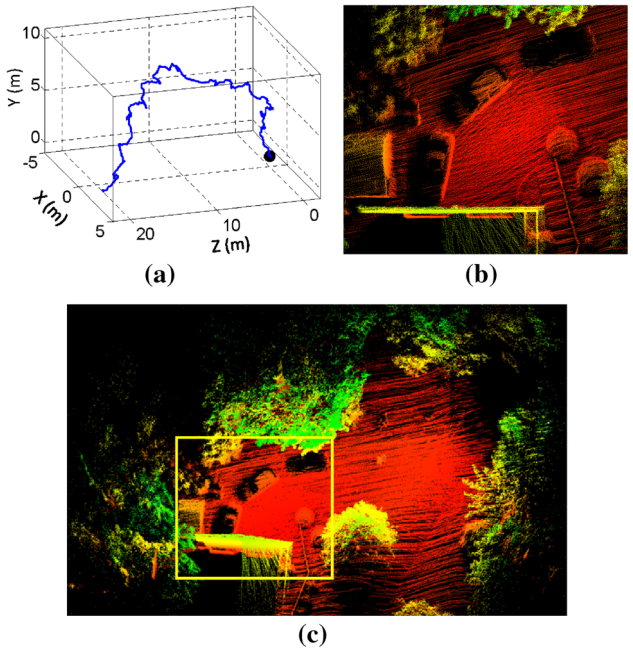


Fig. 16 Results from the front of a house. **a** shows trajectory of the helicopter. The *black dot* is the starting position. **b**, **c** show the map built by the proposed method, where **b** is a zoomed in view of the area inside the *yellow rectangle* in **c**. The helicopter is manually flown, starting with taking-off from the ground and ending with landing on the ground (Color figure online)

we show zoomed in views of the maps, corresponding to the areas inside the yellow rectangles in Figs. 15c and 16c. We are not able to acquire ground truth for the helicopter poses or the maps. For relative small environments as in both tests, the method continuously re-localizes on the maps built at the beginning of the tests. Hence, calculating loop closure errors becomes meaningless. Instead, we can only visually exam accuracy of the maps in the zoomed in views.

7.4 Tests with a Velodyne lidar

These experiments use a Velodyne HDL-32E lidar mounted on two vehicles shown in Fig. 17. Figure 17a is a utility vehicle driven on sidewalks and off-road terrains. Figure 17b is a passenger vehicle driven on streets. For both vehicles, the lidar is mounted high on the top to avoid possible occlusions by the vehicle body.

Fig. 17 Vehicles carrying a Velodyne HDL-32E lidar for data logging. **a** is a utility vehicle driven on sidewalks and off-road terrains. **b** is a passenger vehicle driven on streets

The Velodyne HDL-32E is a single-axis laser scanner. It projects 32 laser beams simultaneously into the 3D environment. We treat each plane formed by a laser beam as a scan plane. A sweep is defined as a full-circle rotation of the laser scanner. The lidar acquires scans at 10 Hz by default. We configure lidar odometry to run at 10 Hz processing individual scans. Lidar mapping stacks scans for a second to do the batch optimization. The computation brake-down for the two programs is shown in Table 4.

Figure 18 shows results of mapping the university campus. The data is logged with the vehicle in Fig. 17a for 1.0 km

Table 4 Computation break-down for Velodyne HDL-32E tests

Program	Build	Match	Others (ms)	Total (ms)
	KD-tree (ms)	Features (ms)		
Odometry	18	35	16	69
Mapping	131	283	149	563

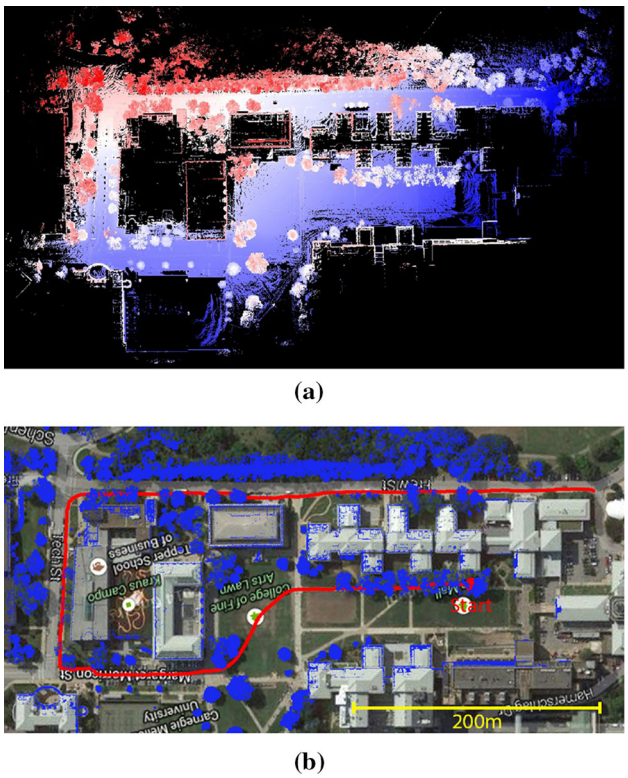
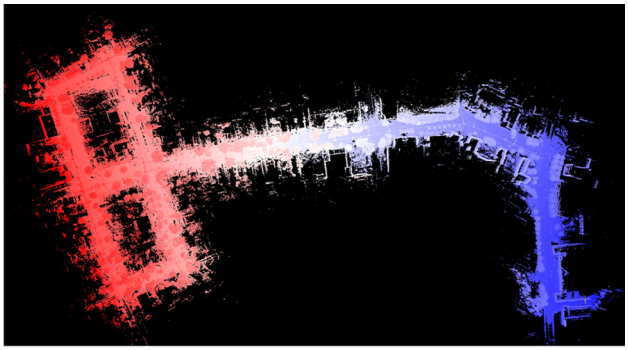
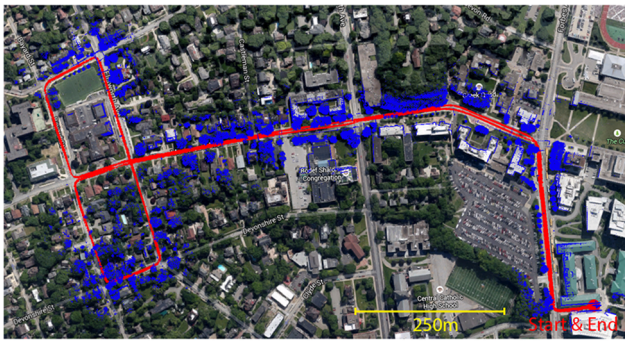


Fig. 18 Results of mapping university campus. The overall run is 1.0 km and the vehicle speed is at 2–3 m/s. **a** shows the final map built and **b** shows the trajectory and registered laser points overlaid on a satellite image. The horizontal position error from matching with the satellite image is $\leq$1m



(a)



(b)

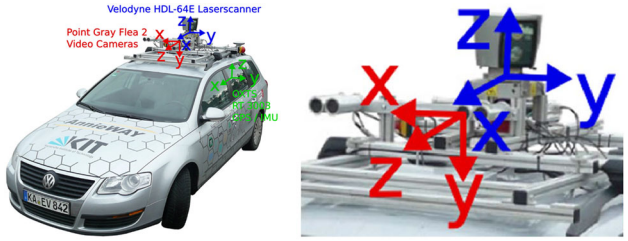
Fig. 19 Results of mapping streets. The path is 3.6 km in length and the vehicle speed is at 11–18 m/s. The figures are in the same arrangement with Fig. 18. The horizontal position error from matching with the satellite image is ≤2m

of travel. The driving speed during the test is maintained at 2–3 m/s. Figure 18a shows the final map built. Figure 18b shows the estimated trajectory (red curve) and the registered laser points overlaid on a satellite image. By matching the trajectory to the sidewalk (the vehicle is not driven on the street) and the laser points to the building walls, we determine the horizontal position drift is ≤1 m. By comparison of mapped buildings from both sides, we are able to determine the vertical drift to be ≤1.5 m. This results in the overall position error to be ≤0.2 % of the distance traveled.

Figure 19 shows results from another test. We drive the vehicle in Fig. 17b on streets for 3.6 km. Except waiting for traffic lights, the vehicle speed is mostly between 11–18 m/s. The figures in Fig. 19 are organized in the same way as Fig. 18. By comparison with the satellite image, we determine the horizontal position error is ≤ 2m. For vertical accuracy, however, we are not able to evaluate.

7.5 Tests with KITTI datasets

Finally, we test the method using the KITTI odometry benchmark (Geiger et al. 2012, 2013). The datasets are logged with sensors mounted on top of a passenger vehicle in road driving scenarios. As shown in Fig. 20, the vehicle is equipped with



(a) (b)

Fig. 20 **a** Vehicle used by the KITTI benchmark for data logging. The vehicle is mounted with a Velodyne lidar, stereo cameras, and a high accuracy GPS/INS for ground truth acquisition. Our method uses data from the Velodyne lidar. **b** Zoomed in view of the sensors. Images taken from http://www.cvlibs.net/datasets/kitti/

color stereo cameras, monochrome stereo cameras, a Velodyne HDL-64E laser scanner, and a high accuracy GPS/INS for ground truth. The laser data is logged at 10 Hz and used by the method for motion estimation. To reach the maximum accuracy possible, the scan data is processed in a slightly different way than in Sect. 7.4. Instead of stacking 10 scans from lidar odometry, lidar mapping runs at the same frequency as lidar odometry and processes each individual scan. This results in the system running at 10 % of the real-time speed, taking one second to process a scan.

The datasets contain 11 tests with the GPS/INS ground truth provided. The maximum driving speed in the datasets reaches 85 km/h (23.6 m/s). The data covers mainly three types of environments: “urban“ with buildings around, “country” on small roads with vegetations in the scene, and “highway” where roads are wide and the vehicle speed is fast. Figure 21 presents sample results from the three environments. On the top row, we show estimated trajectories of the vehicle compared to the GPS/INS ground truth. On the middle and bottom rows, the map and a corresponding image is shown from each dataset. The maps are color coded by elevation. The complete test results with the 11 datasets are listed in Table 5. The three tests from left to right in Fig. 21 are datasets 0, 3, and 1 in the table. Here, the accuracy is calculated by averaging relative position errors using segmented trajectories at 100, 200, . . . , 800 m lengthes, based on 3D coordinates.

Our resulting accuracy is ranked #2 on the KITTI odometry benchmark[1] irrespective of sensing modality, with an average of 0.88 % position error compared to the distance traveled. The results outperform the state of the art vision-based methods (stereo visual odometry methods) (Persson et al. 2015; Badino and Kanade 2011, 2013; Lu et al. 2013; Bellavia et al. 2013 for over 14 % in position error and 24 % in orientation error. In fact, the method is also partially used by the #1 ranked method which runs visual odometry for motion

[1] www.cvlibs.net/datasets/kitti/eval_odometry.php.

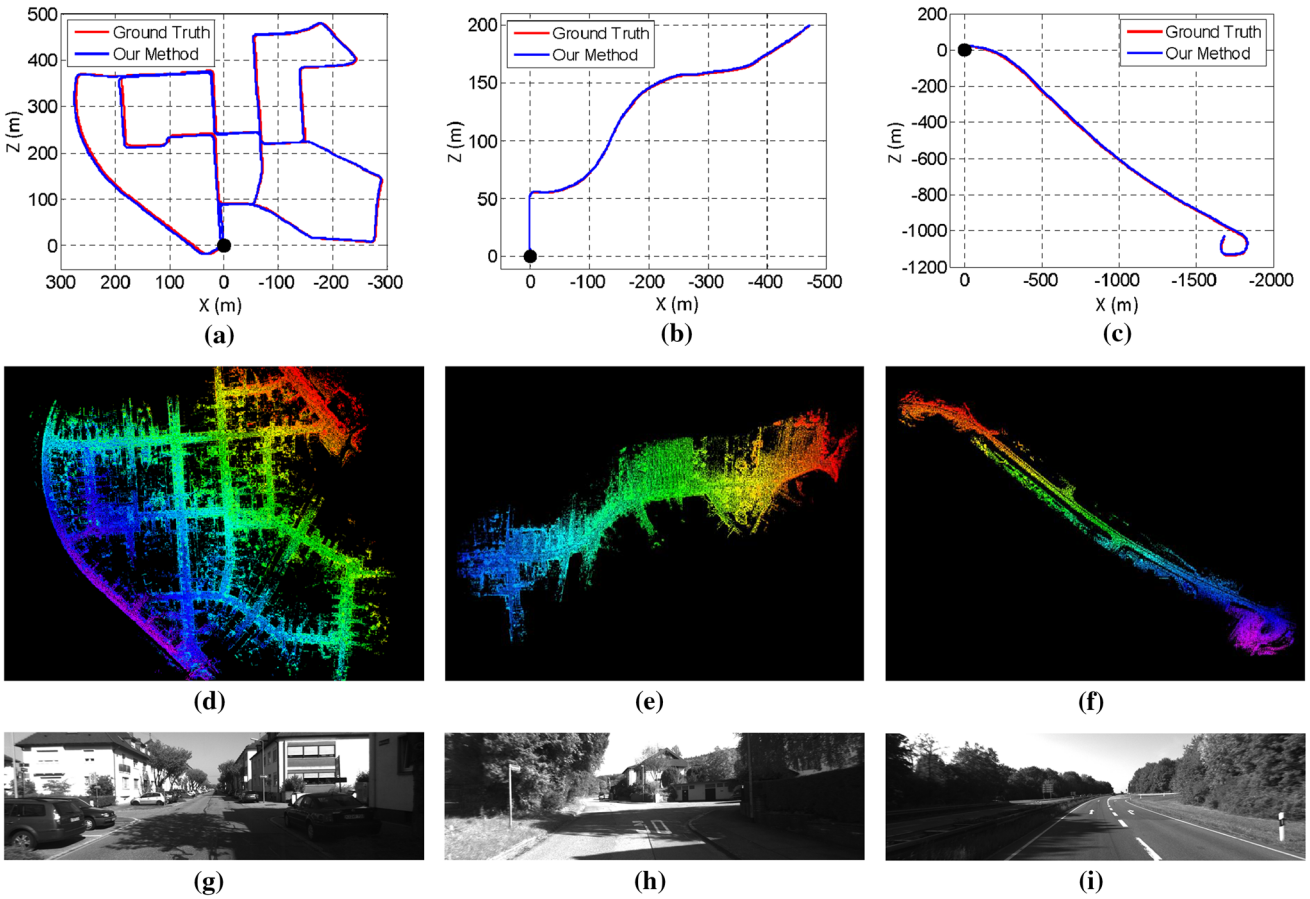


Fig. 21 Sample results using the KITTI benchmark datasets. The datasets are chosen from three types of environments: urban, country, and highway from *left* to *right*, corresponding to tests number 0, 3, and 1 in Table 5. In **a**–**c**, we compare estimated trajectories of the vehicle to the GPS/INS ground truth. The *black dots* are starting positions. **d**–**f** show maps corresponding to **a**–**c**, color coded by elevation. An image is shown from each dataset to illustrate the environment, in **g**–**i**

Table 5 Configurations and results of the KITTI benchmark datasets

Data no.	Configuration		Mean relative position error (%)
	Distance (m)	Environment	
0	3714	Urban	0.78
1	4268	Highway	1.43
2	5075	Urban + Country	0.92
3	563	Country	0.86
4	397	Country	0.71
5	2223	Urban	0.57
6	1239	Urban	0.65
7	695	Urban	0.63
8	3225	Urban + Country	1.12
9	1717	Urban + Country	0.77
10	919	Urban + Country	0.79

The errors are measured using segments of trajectories at 100, 200, ..., 800 m lengthes based on 3D coordinates, as averaged percentages of the segment lengthes

estimation and the proposed method for motion refinement (Zhang and Singh 2015). We think laser-based state estimation is superior than vision-based methods due to the capability of lidars in measuring far points. The lidar range errors are relatively constant w.r.t. the distance measured, and points far away from the vehicle ensure orientation accuracy during scan matching.

8 Discussion

When building a complex system, one question is which components in the system function as the keys to ensure the performance. Our first answer is the dual-layer data processing structure. This allows us to break down the state estimation problem into two problems that are much easier to solve. Lidar odometry only cares about velocity of the sensor and motion distortion removal. The velocity estimates from lidar odometry are not precise (see Fig. 12a) but are good enough to de-wrap point clouds. After which, lidar mapping

only needs to consider rigid body transform for precise scan matching.

Our implementation of geometrical feature detection and matching is a means to realize online and real-time processing. Particularly in lidar odometry, matching does not have to be very precise but high-frequency is more important for point cloud de-wrapping. The implementation takes processing speed as its priority. However, if sufficient amount of computation is available, e.g. with GPU acceleration, the implementation is less necessary.

We do have the choice of choosing the frequency ratio between lidar odometry and lidar mapping. Setting the ratio to be 10 is our preference (lidar mapping is an order of magnitude slower than lidar odometry). This means lidar mapping stacks 10 scan outputs from lidar odometry for one time of batch optimization. Setting the ratio to be higher will typically cause more drift. Also, each time lidar mapping finishes processing, a jump will be introduced to the motion estimates. The ratio is able to keep the motion estimates to be smooth. On the other hand, setting the ratio to be lower will cause more computation and is usually not necessary. The ratio also balances computation load on the two CPU threads. Changing the ratio will put more computation load on one thread than the other.

9 Conclusion and future work

Motion estimation and mapping using point clouds from a rotating laser scanner can be difficult because the problem involves recovery of motion and correction of motion distortion in lidar clouds. The proposed method divides and solves the problem by two algorithms running in parallel. Cooperation of the two algorithms allows accurate motion estimation and mapping to be realized in real-time. The method has been tested in a large number of experiments covering various types of environments, using author collected data as well as datasets from the KITTI odometry benchmark. Since the current method does not recognize loop closure, our future work involves developing a method to correct motion estimation drift by closing the loop.

Acknowledgments The paper is based upon work supported by the National Science Foundation under Grant No. IIS-1328930. Special thanks are given to D. Huber, S. Scherer, M. Bergerman, M. Kaess, L. Yoder, S. Maeta for their insightful inputs and invaluable help.

References

Andersen, R. (2008). Modern methods for robust regression. *Sage University Paper Series on Quantitative Applications in the Social Sciences*.

Anderson, S. & Barfoot, T. (2013). Towards relative continuous-time SLAM. In *IEEE International Conference on Robotics and Automation (ICRA)*, Karlsruhe, Germany.

Anderson, S., & Barfoot, T. (2013). RANSAC for motion-distorted 3D visual sensors. In *2013 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, Tokyo, Japan.

Badino, H., & Kanade, T. (2011). A head-wearable short-baseline stereo system for the simultaneous estimation of structure and motion. In *IAPR Conference on Machine Vision Application*, Nara, Japan.

Badino, A.Y.H., & Kanade, T. (2013). Visual odometry by multi-frame feature integration. In *Workshop on Computer Vision for Autonomous Driving (Collocated with ICCV 2013)*. Sydney, Australia.

Bay, H., Ess, A., Tuytelaars, T., & Gool, L. (2008). SURF: Speeded up robust features. *Computer Vision and Image Understanding*, *110*(3), 346–359.

Bellavia, F., Fanfani, M., Pazzaglia, F., & Colombo, C. (2013). Robust selective stereo slam without loop closure and bundle adjustment. *Lecture Notes in Computer Science*, *8156*, 462–471.

Bosse, M., & and Zlot, R. (2009). Continuous 3D scan-matching with a spinning 2D laser. In *IEEE International Conference on Robotics and Automation*, Kobe, Japan.

Bosse, M., Zlot, R., & Flick, P. (2012). Zebedee: Design of a spring-mounted 3-D range sensor with application to mobile mapping. *IEEE Transactions on Robotics*, *28*(5), 1104–1119.

de Berg, M., van Kreveld, M., Overmars, M., & Schwarzkopf, O. (2008). *Computation geometry: Algorithms and applications* (3rd ed.). Berlin: Springer.

Dong, H. & Barfoot, T. (2012). Lighting-invariant visual odometry using lidar intensity imagery and pose interpolation. In *The 7th International Conference on Field and Service Robots*, Matsushima, Japan.

Furgale, P., Barfoot, T. & Sibley, G. (2012). Continuous-time batch estimation using temporal basis functions. In *IEEE International Conference on Robotics and Automation (ICRA)*, St. Paul, MN.

Geiger, A., Lenz, P. & Urtasun, R. (2012). Are we ready for autonomous driving? The kitti vision benchmark suite. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (pp. 3354–3361).

Geiger, A., Lenz, P., Stiller, C., & Urtasun, R. (2013). Vision meets robotics: The KITTI dataset. *International Journal of Robotics Research*, *32*, 1229–1235.

Guo, C.X., Kottas, D.G., DuToit, R.C., Ahmed, A., Li, R. & Roumeliotis, S.I. (2014). Efficient visual-inertial navigation using a rolling-shutter camera with inaccurate timestamps. In *Proceedings of Robotics: Science and Systems*, Berkeley, CA.

Hartley, R., & Zisserman, A. (2004). *Multiple view geometry in computer vision*. New York: Cambridge University Press.

Hong, S., Ko, H. & Kim, J. (2010). VICP: Velocity updating iterative closest point algorithm. In *IEEE International Conference on Robotics and Automation (ICRA)*, Anchorage, Alaska.

Li, Y. & Olson, E. (2011) Structure tensors for general purpose LIDAR feature extraction. In *IEEE International Conference on Robotics and Automation*, Shanghai, China, May 9–13.

Li, M., & Mourikis, A. (2014). Vision-aided inertial navigation with rolling-shutter cameras. *International Journal of Robotics Research*, *33*(11), 1490–1507.

Lu, W., Xiang, Z., & Liu, J. (2013). High-performance visual odometry with two-stage local binocular ba and gpu. In *IEEE Intelligent Vehicles Symposium*. Gold Coast City, Australia.

Moosmann, F., & Stiller, C. (2011). Velodyne SLAM. In *IEEE Intelligent Vehicles Symposium (IV)*. Baden-Baden, Germany.

Murray, R., & Sastry, S. (1994). *A mathematical introduction to robotic manipulaton*. Boca Raton: CRC Press.

Nuchter, A., Lingemann, K., Hertzberg, J., & Surmann, H. (2007). 6D SLAM-3D mapping outdoor environments. *Journal of Field Robotics*, *24*(8–9), 699–722.

Persson, M., Piccini, T., Mester, R., & Felsberg, M. (2015). Robust stereo visual odometry from monocular techniques. In *IEEE Intelligent Vehicles Symposium*. Seoul, Korea.

Pomerleau, F., Colas, F., Siegwart, R., & Magnenat, S. (2013). Comparing ICP variants on real-world data sets. *Autonomous Robots*, *34*(3), 133–148.

Quigley, M., Gerkey, B., Conley, K., Faust, J., Foote, T., Leibs, J., et al. (2009). ROS: An open-source robot operating system. In *Workshop on Open Source Software (Collocated with ICRA 2009)*. Kobe, Japan.

Rosen, D., Huang, G., & Leonard, J. (2014). Inference over heterogeneous finite-/infinite-dimensional systems using factor graphs and Gaussian processes. In *IEEE International Conference on Robotics and Automation (ICRA)*, Hong Kong, China.

Rusinkiewicz, S. & Levoy, M. (2001). Efficient variants of the ICP algorithm. In *Third International Conference on 3D Digital Imaging and Modeling (3DIM)*, Quebec City, Canada.

Rusu, R.B., & Cousins, S. (2011). 3D is here: Point Cloud Library (PCL). In *IEEE International Conference on Robotics and Automation (ICRA)*, Shanghai, China, May 9–13.

Scherer, S., Rehder, J., Achar, S., Cover, H., Chambers, A., Nuske, S., et al. (2012). River mapping from a flying robot: State estimation, river detection, and obstacle mapping. *Autonomous Robots*, *32*(5), 1–26.

Thrun, S., Burgard, W., & Fox, D. (2005). *Probabilistic robotics*. Cambridge, MA: The MIT Press.

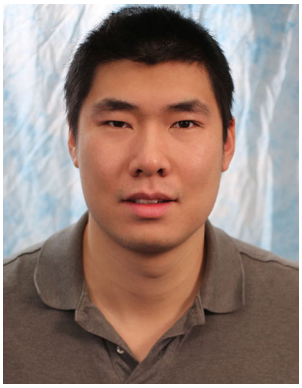
Tong, C. H. & Barfoot, T. (2013). Gaussian process Gauss-Newton for 3D laser-based visual odometry. In *IEEE International Conference on Robotics and Automation (ICRA)*, Karlsruhe, Germany.

Tong, C., Furgale, P., & Barfoot, T. (2013). Gaussian process Gauss-newton for non-parametric simultaneous localization and mapping. *International Journal of Robotics Research*, *32*(5), 507–525.

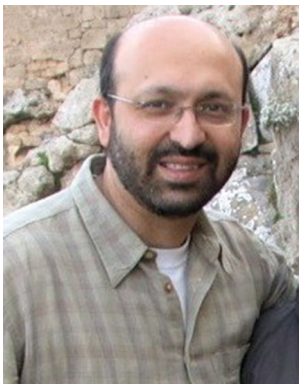
Zhang, J. & Singh, S. (2014). LOAM: Lidar odometry and mapping in real-time. In *Robotics: Science and Systems Conference (RSS)*, Berkeley, CA.

Zhang, J. & Singh, S. (2015). Visual-lidar odometry and mapping: Low-drift, robust, and fast. In *IEEE International Conference on Robotics and Automation (ICRA)*, Seattle, WA, May

Zlot, R. & Bosse, M. (2012). Efficient large-scale 3D mobile mapping and surface reconstruction of an underground mine. In *The 7th International Conference on Field and Service Robots*, Matsushima, Japan.



Ji Zhang is a Ph.D. candidate at the Robotics Institute of Carnegie Mellon University. His research interest is focused on robot navigation, perception and localization, lidar mapping, and computer vision.



Sanjiv Singh is a Research Professor at the Robotics Institute with a courtesy appointment in Mechanical Engineering. He is the founding editor of the Journal of Field Robotics. His research relates to the operation of robots in natural and in some cases, extreme environments. His recent work has two main themes: perception in natural and dynamic environments and multi-agent coordination. Currently, he leads efforts in collision avoidance for air vehicles (near earth and between aircraft) and ground vehicles using novel sensors that can look through obscurants. Another research area seeks to provide accurate tracking and situational awareness in dynamic environments, such as those encountered in search and rescue, using radio signals to compute location. This research seeks to assist emergency response personnel in their operations.

2011 IEEE International Conference on Robotics and Automation
Shanghai International Conference Center
May 9-13, 2011, Shanghai, China

Onboard IMU and Monocular Vision Based Control for MAVs in Unknown In- and Outdoor Environments

Markus Achtelik, Michael Achtelik, Stephan Weiss, Roland Siegwart

Abstract— **In this paper, we present our latest achievements towards the goal of autonomous flights of an MAV in unknown environments, only having a monocular camera as exteroceptive sensor. As MAVs are highly agile, it is not sufficient to directly use the visual input for position control at the framerates that can be achieved with small onboard computers. Our contributions in this work are twofold. First, we present a solution to overcome the issue of having a low frequent onboard visual pose update versus the high agility of an MAV. This is solved by filtering visual information with inputs from inertial sensors. Second, as our system is based on monocular vision, we present a solution to estimate the metric visual scale aid of an air pressure sensor. All computation is running onboard and is tightly integrated on the MAV to avoid jitter and latencies. This framework enables stable flights indoors and outdoors even under windy conditions.**

I. Introduction

The research in autonomous micro helicopters is advancing and evolving fast. Even though a lot of progress has been achieved in this topic during the past years, the community is still striving to achieve simple autonomous flights in unknown and GPS denied environments. Only after solving this issue, high level tasks such as autonomous exploration, swarming, and large trajectory planning can be tackled.

Stable flights and navigation with GPS are well explored and work out of the box [1]. However, GPS is not a reliable service as its availability can be limited by urban canyons and is completely unavailable in indoor environments. The alternative of using laser range finders is not optimal since these sensors have a restricted perception distance and are still heavy for MAVs.

Considering the above mentioned and to be independent of the (quality of the) GPS signal, a viable solution is to navigate with a vision based system. This ensures operations of the MAV indoors as well as outdoors. Recently we presented, to the best of our knowledge, the first vision based solution for completely autonomous flights in unknown and GPS denied environments [2]. In that work, the vision algorithms ran off-board on a ground station transferring the control commands wireless back to the helicopter. Since wireless communication is not always reliable, there is a strong need for computing all tasks of the control framework onboard.

The research leading to these results has received funding from the European Community's Seventh Framework Programme (FP7/2007-2013) under grant agreement n. 231855 (sFly). Markus Achtelik and Stephan Weiss are currently PhD students at the ETH Zurich (email: {markus.achtelik, stephan.weiss}@mavt.ethz.ch). Roland Siegwart is full professor at the ETH Zurich and head of the Autonomous Systems Lab (email: r.siegwart@ieee.org).

Michael Achtelik is CEO of Ascending Technologies GmbH, Germany (email: michael.achtelik@asctec.de)

Fig. 1. Outdoor autonomous flight using only onboard IMU aided monocular vision.

In this work, we develop our previous system to an onboard solution. As exteroceptive sensor, we still use one single camera because stereo vision loses its effect for large distances and small baselines. However, we include the onboard IMU more tightly as in the previous work, to tackle the issue of low-frequent vision updates. Also, we added a pressure sensor to estimate the absolute scale of the visual pose measurements. Finally, we optimized the visual framework for an embedded solution in order to cope with the limited calculation power onboard.

II. Related work

Previous work on position control using visual input has been done in several ways. Stable flights were shown using onboard cameras and landmarks placed in the environment such as blobs or other artificial markers in [3]. However, these approaches only work in controlled environments whereas we focus on stable flights without having any prior information about the environment nor GPS signals. This was successfully shown in [4] using a lightweight laserscanner and/or cameras in a stereo configuration and off-board computation. Offloading sensor data to a ground station has major drawbacks. Not only become delays a significant issue, but also the high bandwidth datalink has to be granted at any time. Compressing large data chunks (such as images) diminishes the issue but introduces artefacts. Furthermore, in the mentioned approach the stereo camera and the laser scanner have a limited range of operation.

An alternative approach is to use cameras for the localization task. However this vast information has to be processed accordingly. The most simple way is to install

a number of external cameras with known location and to have them track the MAV [5], [6], [7]. This method is very efficient for testing purposes and can be used to evaluate other approaches as ground truth reference. However it is not suitable for missions where the installation of an appropriate infrastructure is not feasible.

This approach can also be implemented the other way round: the camera is mounted on the helicopter and tracks a known pattern on the ground [8]. Hamel et al. [9] implemented a visual servoing based trajectory tracking to control an UAV with a camera observing n fixed points. Further methods have also been developed by fusing the visual data with IMU data [10].

Alternatively, stabilizing controllers can be built by means of optical flow considerations [11]. Herisse et al. [12] use an optical flow based PI-controller to stabilize a hovering MAV. They also implemented an automatic landing routine by contemplating the divergent optical flow. Hrabar et al. [13] developed a platform able to navigate through urban canyons. It was based on the analysis of the optical flow on both sides of the vehicle. Also, by having a forward looking stereo camera, they were able to avoid oncoming obstacles. Most recently, cheap systems sold as toys [14] were presented, performing stabilization based on optical flow and ultra sonic height sensors onboard.

An approach with offboard vehicle tracking equipment was implemented by Ahrens et al. [15]. Based on the visual SLAM algorithm of Davison et al. [16], they build a localization and mapping framework that is able to provide an almost drift-free pose estimation. With that they implemented a very efficient position controller and obstacle avoidance framework. However, due to the simplification they used in their feature tracking algorithm, a non-negligible drift persists. Also, they used an external Vicon localization system to control the aerial vehicle with millimeter precision (a system of external cameras that tracks the 3D pose of the vehicle). So far, they did not use the output of the visual SLAM based localization system for controlling the vehicle.

In this paper, we discuss the thorough implementation of a vision based controller framework onboard a micro helicopter. Compared to optical flow based approaches that drift over time, we focus on a solution that enables absolute position control. Note that we do not claim to have developed a novel controller. This has already been discussed in previous work [17], [7]. Rather, we highlight the issues of a full onboard implementation and their solutions. More precisely, our contributions are the following. First, we present a framework to tackle the issue of a very slow visual pose update versus the high agility of the micro helicopter. We solve this issue with filtering the visual information with inputs from inertial sensors at $1\,\mathrm{kHz}$ onboard the helicopter. This high frequency also enables us to estimate a reliable speed information that is crucial for such agile platforms. Since our visual framework consists of only one single fisheye camera, our second main contribution is to demonstrate how to recover the absolute scale of the visual pose estimation. We do so by filtering the visual pose with an onboard pressure sensor. The filter also compensates for the pressure sensor's drift. Last, we discuss a fast implementation of the visual framework [18] onboard the micro helicopter.

Our implementation of an onboard monocular vision-based MAV controller can be used in an unknown environment without the aid of any infrastructure based localization system, any beacons, artificial features, or any prior knowledge on the environment. In other words, our platform does not need any external assistance in order to navigate through an unexplored region and is not bound to a ground station. All our implementations are based on the Robot Operating System (ROS) [19]. This makes our work reusable for the community and represents as such a valuable contribution towards the fast development of autonomously flying MAVs.

The remainder of the paper is organized as follows: In Section III, we describe the platform we used. In Section IV, we present the algorithm and the implementation of the system. Experimental results and the evaluation are shown in Section V. Conclusions are given in Section VI.

III. Platform Description

A. Hardware

The MAV we use is a so-called quadrocopter, a helicopter driven by for rotors, symmetric to the center of mass. The control of the quadrocopter is performed solely by changing the rotation speed of the propellers and is described in more detail in [20]. For our experiments, we use the "AscTec Pelican" quadrocopter [1], which is a further development of the one described in [20]. The quadrocopter is equipped with rotors with 10" diameter which allow to carry a payload of about $500\,\mathrm{g}$. Depending on battery size and payload, flight times between 10 and 20 minutes can be achieved. Further key features are the Flight Control Unit (FCU) "AscTec Autopilot" as well as the flexible design enabling one to easily mount different payloads like computer boards or cameras. The FCU features a complete Inertial Measurement Unit (IMU) as well as two $32\,\mathrm{Bit}$, $60\,\mathrm{MHz}$ ARM-7 microcontrollers used for data fusion and flight control. One of these microcontrollers, the Low Level Processor (LLP) is responsible for the hardware management and IMU sensor data fusion. An attitude and GPS-based position controller is implemented as well on this processor. The LLP is delivered as a black box with defined interfaces to additional components and to the High Level Processor (HLP). To operate the quadrocopter, only the LLP is necessary. Therefore, the HLP is dedicated for custom code. All relevant and fused IMU data is provided at an update rate of $1\,\mathrm{kHz}$ via a highspeed serial interface. In particular, this comprises body accelerations, body angular velocities, magnetic compass, height measured by an air pressure sensor and the estimated attitude of the vehicle.

For the computationally more expensive onboard processing tasks, we outfitted the helicopter with a 1.6 GHz Intel Atom Based embedded computer, available from [1]. This computer is equipped with 1 GB RAM, a MicroSD card slot for the operating system, a 802.11n based miniPCI Express WiFi card and a Compact Flash slot. The miniPCIE WiFi

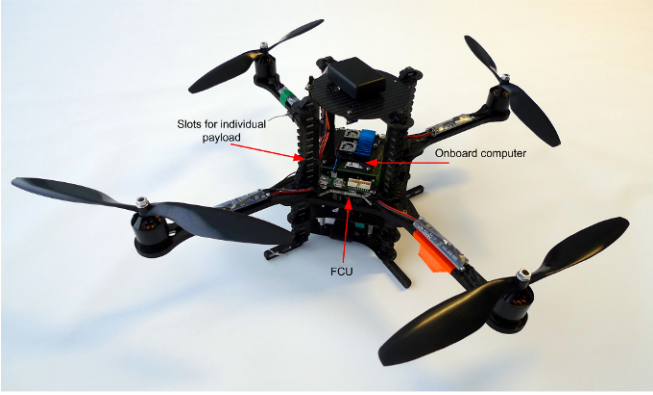


Fig. 2. Overview of the Pelican quadrocopter

card is preferred over USB to keep the USB bus free for devices like the cameras we use. We furthermore use a high speed CF-card that allows us data logging with up to 40 MByte/s.

As camera, we use a Point-Grey USB Firefly camera with a resolution of 752×480 px and a global shutter. The camera faces the ground with a $150°$ field-of-view lens since we are expecting the most stable features trackable over longer time in this configuration.

In this work, a position controller and the position data fusion algorithms are implemented on the HLP, based on the vision input from the onboard computer and the inertial data provided by the LLP. On the LLP, the attitude controller is used as inner loop. A GPS-based position controller is used as a fall-back in case of any failures on the HLP during outdoor experiments.

B. Software

To provide a maximum portability of our code and to avoid potential (binary) driver issues, we installed Ubuntu Linux 10.04 on our onboard computer which makes tedious crosscompiling unnecessary. Since we are running a couple of different subsystems that need to communicate between each other, we use the ROS [19] framework as a middleware. This is also used to communicate to the ground station over the WiFi datalink for monitoring and control purposes. The FCU is interfaced via a ROS node communicating over a serial link to the FCU's Higlevel Controller with firmware we developed for our purposes.

Software development on the HLP is done based on a SDK available for the AutoPilot FCU providing all communication routines to the LLP and a basic framework. The HLP communicates with the ROS framework on the onboard computer over a serial datalink and a ROS FCU-node handling the serial communication. This node subscribes to generic ROS pose messages with covariance, in our case from the vision framework, and forwards it to the HLP. Moreover, it allows to monitor the state of the fusion filter and the position controller, and to adjust their parameters online via the "dynamic reconfigure" functionality of ROS.

For the implementation of the position control loop and data fusion onboard the HLP, a Matlab/Simulink framework is used in combination with the Mathworks Real-Time Workshop Embedded Coder. The framework provides all necessary tools to design the control structure in Simulink, optimize it for fixed point computing, as well as compiling and flashing the HLP.

IV. Onboard Vision Based Position Control

A. Overview

In this section, we describe the essential components that we used to enable autonomous flights running all computation onboard. The basic structure can be seen in Fig. 3. We obtain absolute position estimates by a monocular visual SLAM (VSLAM) framework and estimate the absolute scale with the help of an air pressure sensor. Since this process (approx. $10\,\mathrm{Hz}$) is slow compared to the motion of the MAV, we fuse this information with inertial sensor data (angular rates and body acceleration) provided by the IMU at a rate of $1\,\mathrm{kHz}$. The outputs of that filter are finally fed into a position controller based on nonlinear dynamic inversion. While the computationally expensive VLSAM is run on the Atom onboard computer at approximately $10\,\mathrm{Hz}$, the fusion filter and the position controller are executed on the HLP (see Section III-A) at $1\,\mathrm{kHz}$ – just when new IMU readings arrive. This ensures minimum possible delays and allows us to handle the fast movements and disturbances of the MAV. The ground station is solely used for monitoring or sending highlevel commands such as waypoints.

B. Visual framework

The approach presented in this paper uses the visual SLAM (VSLAM) algorithm of Klein and Murray [18] in order to localize the MAV with the aid of a single camera

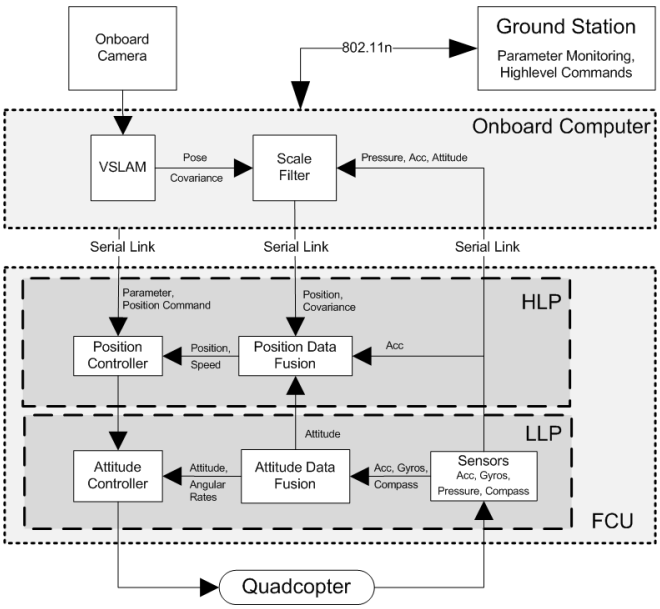


Fig. 3. System overview

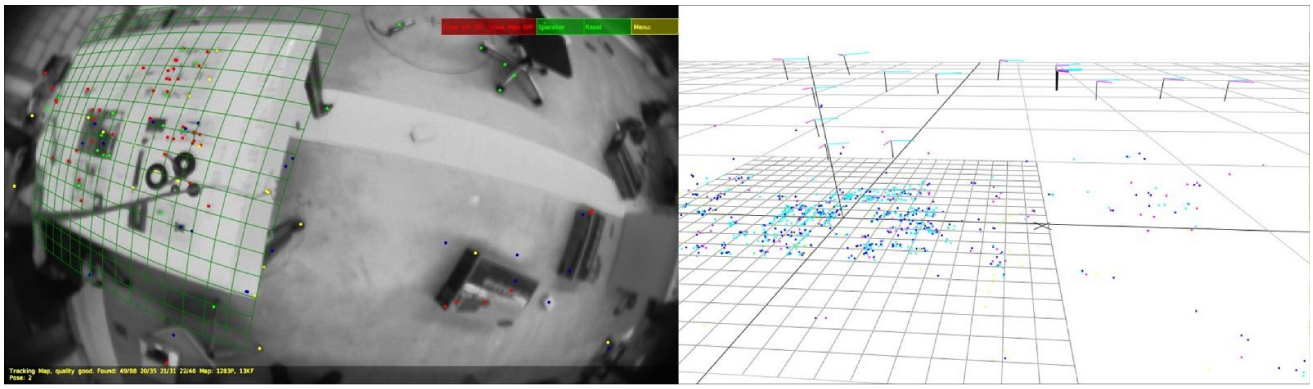


Fig. 4. Screenshot of Klein and Murray's SLAM algorithm. On the left, the tracking of the FAST corners can be observed, this is used for the localization of the camera. On the right, the 3D map that was build by the mapping thread is shown. The 3-axis coordinate frames represent the location where new keyframes where added.

(see Fig. 4). In short, the authors split the simultaneous localization and mapping task into two separately-scheduled threads: the tracking thread and the mapping thread. Splitting the SLAM algorithm into a mapping and a tracking part brings the advantage that both can run at different speeds. The tracker can thus yield fast pose updates while the mapper can use more powerful (slower) map optimization techniques. Compared to frame-by-frame SLAM the mapper does not process every camera frame. This eliminates to a great extend redundant information processing during slow movements or hovering. Furthermore, it is very easy to adapt and optimize independently each of the threads to our specific needs on the flying platform. These are the main reasons we choose this SLAM algorithm. We describe our modifications in the following.

As our whole framework uses ROS as middleware, we modified the VSLAM such that it exports the 6-DOF pose estimate, the map information and tracking quality as ROS messages to the HLP over the FCU-node. Similarly, we modified it to accept initialization and reset commands as ROS services such that we can remote control the algorithm from the ground station. Note that only during the VSLAM initialization phase these commands are sent from the ground station to the MAV. The VSLAM framework itself runs completely on the onboard computer.

The tracking part of the VSLAM algorithm is already well designed for our needs. We set it to use a maximum of 300 features per frame. For successful MAV navigation we only need our vincinity to be consistent, that is, it is sufficient to have only a local map well aligned with the gravity vector. The FCU's gravity estimate and the air pressure sensor can be used to compensate for map and scale drifts respectively (see Section IV-C). More important is that we do not have to store a global map. In fact only a few keyframes are sufficient for a local consistent pose estimate. Here, we limit the number of keyframes retained in the map. When a new keyframe is requested, the keyframe furthest away in the euclidean space is deleted. Also, all features corresponding to that keyframe are removed from the map. This ensures constant speed in map maintenance since only N keyframes and M features take part in the nonlinear map refinement. Also, it ensures constant speed in tracking in already explored areas since the number of features M is roughly constant.

With these modifications, the VSLAM algorithm runs approximately at $10\,\mathrm{Hz}$ on the onboard computer. The framerate can temporarily drop down to $5\,\mathrm{Hz}$ during a nonlinear map refinement when a new keyframe is added. Of course a loss of the local map or textureless regions are fatal for the here presented algorithm. However, failure of the VSLAM algorithm can be detected. In that case, the data fusion algorithm from Section IV-D is not updated any more. The vehicle will then drift away slowly since position information just relies on integration of acceleration sensors. This still leaves enough time for a safety pilot to take over control or for switching back to an alternative localization. Thus far, we experienced that in outdoor environments we rarely lack of features. Known difficult scenarios are environments like self-similar paved roads or uniform indoor floors, while natural scenes, such as bushes or unpaved roads usually provide sufficient texture.

C. Scale Estimation

Observe that, because we are using a single camera, the VSLAM framework can give us only the direction of translation but not its magnitude, that is, the absolute scale.

To recover the absolute scale—in order to pass proper position information to the fusion filter—there are basically two solutions. The first solution consists in measuring the size of an element in the scene. This quickly gets computationally infeasible on our setup and is very likely prone to errors. The other solution is to use additional sensors that provide absolute measurements. For instance, previous work has been done with ultrasonic range sensors. This works well in principle but limits the maximum operating height of the vehicle which is about $2-5\,\mathrm{m}$ for commonly available sensors.

Therefore, we use the accelerometers and the air pressure sensor of the FCU to recover the scale. The pressure sensor has the advantage of almost unlimited height, but the drawbacks are drift and noisy measurements. Simply using the

ratio of the height measured by the air pressure sensor and the (also noisy) height from VSLAM would lead to inaccurate results. Potential issues with taking measurements from the air pressure sensor are twofold. First, the zero height of the pressure sensor does not align with the VSLAM reference frame. Second, the measurements of the pressure sensor are drifting over time due to changing weather conditions. We solve these problems by designing an EKF using the pressure sensor as well as the accelerometers, and incorporating the scale and pressure sensor drift in the states.

The state $\boldsymbol{x}$ consists of the absolute height p_z, the climb rate v_z, the absolute scale λ and the pressure sensor bias b. As process input, we chose the acceleration a_z expressed in world coordinates. To gain the acceleration $\boldsymbol{a}$ in world coordinates from the measured body acceleration $\boldsymbol{a}_{body}$, we need to transform it by the attitude $\boldsymbol{R} \in SO(3)$ of the vehicle, estimated by the FCU. Finally, $\boldsymbol{a}$ needs to be corrected for the gravity.

$$\boldsymbol{a} = \begin{bmatrix} a_x & a_y & a_z \end{bmatrix}^T = \boldsymbol{R} \cdot \boldsymbol{a}_{body} - \begin{bmatrix} 0 & 0 & g \end{bmatrix}^T \quad (1)$$

As measurement $\boldsymbol{z}$, we chose the height from VSLAM $p_{z,v}$ and the height measured by the air pressure sensor $p_{z,p}$. To summarize:

$$\boldsymbol{x} = \begin{bmatrix} p_z & v_z & \lambda & b \end{bmatrix}^T \quad \boldsymbol{z} = \begin{bmatrix} p_{z,p} & p_{z,v} \end{bmatrix}^T \quad (2)$$

The differential equations governing the state are:

$$\dot{p}_z = v_z \qquad \dot{b} = n_b \quad (3)$$

$$\dot{v}_z = a_z + n_a \qquad \dot{\lambda} = n_\lambda \quad (4)$$

The noise n_a of the acceleration measurement is assumed to be white gaussian noise. Bias b and scale λ are modeled as random walks with their derivatives being white gaussian noise n_a and n_λ respectively. We have two measurements arriving not synchronized and at different rates, therefore we need measurement prediction functions $h_v(\boldsymbol{x})$ and $h_p(\boldsymbol{x})$ for the scaled height measurement $p_{z,v}$ from the vision algorithm and the absolute height measurement $p_{z,p}$ from the pressure sensor:

$$\hat{z}_v = h_v(\boldsymbol{x}) = p_z \cdot \lambda; \qquad \tilde{z}_v = p_{z,v} - h_v(\boldsymbol{x}) \quad (5)$$

$$\hat{z}_p = h_p(\boldsymbol{x}) = p_z + b; \qquad \tilde{z}_p = p_{z,p} - h_p(\boldsymbol{x}) \quad (6)$$

State update, Kalman gain and process covariance are finally computed following the standard EKF scheme. To estimate the bias properly, motion of the vehicle in the z axis is required, otherwise the scale will not be correctly estimated. The performance of the filter will be evaluated in Section V

D. Data Fusion

Fast data fusion algorithms are essential to match the high bandwidth of the quadrocopter's system dynamics. The attitude angles and angular rates of the quadrocopter are already provided by the LLP at $1\,\mathrm{kHz}$ update rate. For position control, fast data fusion algorithms of all kinematic measurements are needed. The LLP provides acceleration measurements at $1\,\mathrm{kHz}$ update rate and the vision system provides position and heading information at $5 - 10\,\mathrm{Hz}$. A position filter has been developed taking into account the computational limitations of the microcontroller hardware where a full state Kalman filter working at an update rate of at least $500\,\mathrm{Hz}$ is not feasible. This high update rate is needed to enable a high update rate in the position control loop to match the quadrocopter's system dynamics. In particular, the aim of the filter is to combine both the vision and the acceleration sensor to achieve a fused signal, featuring fast reactions on disturbances based on the high update rate of the acceleration sensors and steady state accuracy based on the vision signal. The filter is designed decoupled for all three axes x, y, z and works in a global (0-) frame. In the following, only the filter for the x-axis is described and applies for the other axes respectively. The body-fixed accelerations are rotated in the global frame by a simple rotational matrix based on the attitude angles provided by the LLP, as shown in (1). The filter is based on a Luenberger observer [21] with the position p_x, speed v_x, and the acceleration sensor bias b_x as state. The acceleration notated in the global (0-) frame, separated for each axis, is the system input. The measurement can be any (absolute) position input, which in our case is the position $\boldsymbol{p}_{v\lambda} = \boldsymbol{p}_v/\lambda$ from the VSLAM, corrected by the scale estimated with the method described in Section IV-C.

$$\boldsymbol{x} = \begin{bmatrix} p_x & v_x & b_x \end{bmatrix}^T \quad \boldsymbol{u} = \begin{bmatrix} a_x \end{bmatrix}^T \quad \boldsymbol{y} = \begin{bmatrix} p_{x,v\lambda} \end{bmatrix}^T \quad (7)$$

Again, we use a linear motion model to describe the system. Since this filter runs at 1 kHz, we consider this as continuous time system.

$$\dot{\hat{\boldsymbol{x}}} = \boldsymbol{A} \cdot \hat{\boldsymbol{x}} + \boldsymbol{L}\,(\boldsymbol{y} - \hat{\boldsymbol{y}}) + \boldsymbol{B} \cdot \boldsymbol{u} \quad (8)$$

$$\hat{\boldsymbol{y}} = \boldsymbol{H} \cdot \hat{\boldsymbol{x}} \quad (9)$$

with:

$$\boldsymbol{A} = \begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 0 & 0 & 0 \end{bmatrix} \quad \boldsymbol{B} = \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix}$$

$$\boldsymbol{H} = \begin{bmatrix} 1 & 0 & 0 \end{bmatrix} \quad \boldsymbol{L} = \begin{bmatrix} L_1 & L_2 & L_3 \end{bmatrix}^T$$

The elements of the matrix L are calculated based on considerations on the eigenvalues of the error dynamics. The state error is defined as $\tilde{\boldsymbol{x}} = \boldsymbol{x} - \hat{\boldsymbol{x}}$ and the error dynamics can be calculated as:

$$\dot{\tilde{\boldsymbol{x}}} = (\boldsymbol{A} - \boldsymbol{L}\boldsymbol{H})\,\tilde{\boldsymbol{x}} \quad (10)$$

We used Simulink and the Mathworks Real-Time Workshop Embedded Coder to implement the data fusion filter on the HLP (see Section III-B). All calculations are optimized for fixed point arithmetics and unnecessary matrix operations are dropped. Relevant parameters such as $\boldsymbol{L}$ as well as the states are connected to communication channels that can be accessed through ROS messages and services (see Section III-B) for debugging and parameter changes. In future, it is planned to use this functionality to extend the observer to a full state Kalman filter by computing the update step on the onboard computer to find optimal values for $\boldsymbol{L}$ when a new measurement arrives.

In a last step, methods were implemented to reset, hold and reinitialize the filter's integrators in case of loss or re-initialization of the input from VSLAM. The output of this position filter are position and speed signals. The filter is able to react fast to disturbances measured by the acceleration sensors far before it is possible to observe these disturbances by the visual sensor.

E. Position Controller

For position control, a cascade structure is used. As inner loop, the well tested attitude loop provided by the LLP of the FCU is used (see Section III-A). The outer loop is the position loop, and is implemented on the HLP based on the concept of nonlinear dynamic inversion. With an adequate knowledge of the plant dynamics, this control approach can transform the nonlinear system into an equivalent linear system without any simplification, through exact state transformation and suitable control inputs [22]. Based on this input-output linearization, standard linear control strategies like PD controllers can be applied. For the quadrocopter position controller, a control structure of relative degree two is implemented. That means, position and speed control are performed in one control loop on the onboard hardware. Fig. 5 shows the control structure including the rates of the different parts.

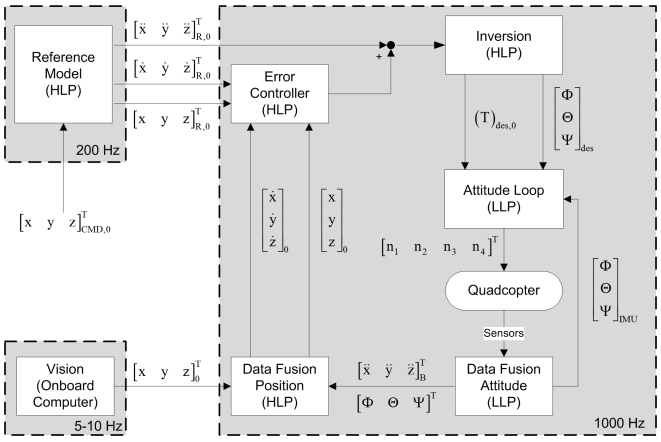


Fig. 5. Structure of the position controller. Subscript 0 denotes coordinates w.r.t a global frame, B w.r.t the current body frame. The names in braces denote on which physical device the corresponding part is executed

Input commands of the attitude loop are desired attitude angles roll, pitch and yaw $\begin{bmatrix}\Phi & \Theta & \Psi\end{bmatrix}^T_{des}$ and the thrust T_{des} commanded by the position loop. Outputs are the commanded rotational velocities $\begin{bmatrix}n_1 & \dots & n_4\end{bmatrix}^T$ of the four rotors. This control loop is implemented on the LLP, delivered with the FCU and is not the focus of this paper.

For the position control loop, the quadrocopter translation dynamics need to be modeled and inverted. The world frame (denoted by 0) is used as inertial frame in order to apply to Newton's law. Furthermore, the data fusion and the generation of reference trajectories, as described later, is performed in this frame. To simplify the inversion, the $\bar{0}$-frame has been introduced as a leveled frame with the same yaw angle Ψ as a local body frame denoted by B. In a first step, desired accelerations in the 0-frame can be transformed into the $\bar{0}$-frame by a simple rotation through the azimuth Ψ. Applying Newton's second law results in:

$$m \cdot \boldsymbol{a}_{\bar{0}} = \boldsymbol{f}_{\bar{0}} + \boldsymbol{f}_{g,\bar{0}} = \boldsymbol{M}_{\bar{0}B} \cdot \boldsymbol{f}_B + \boldsymbol{f}_{g,\bar{0}} \tag{11}$$

m denotes the mass of the quadrocopter, $\boldsymbol{a}$ the acceleration expressed in the $\bar{0}$-frame, $\boldsymbol{f}$ the forces on the quadrocopter in the $\bar{0}$-frame and the B-frame respectively, $\boldsymbol{f}_g$ the gravitational vector and $\boldsymbol{M}_{\bar{0}B}$ denotes the transformation matrix between the $\bar{0}$ and B-frame. Solved for the roll angle Φ, the pitch angle Θ and the thrust T, we get:

$$T = m \cdot \sqrt{a_x^2 + a_y^2 + (a_z - g)^2} \tag{12}$$

$$\Phi = \arctan \frac{m a_y}{T} \tag{13}$$

$$\Theta = \arctan \frac{a_x}{a_z - w} \tag{14}$$

Where g is the gravitational constant. By these equations, a transformation is given transforming so called pseudo controls $\boldsymbol{\nu} = \boldsymbol{a}$ into the controls of the system Φ, Θ, T. Therefore, a linear dynamic between the inputs, the position commands and the pseudo controls is achieved and linear control methods can be applied.

As only the second time derivatives of the position commands $\boldsymbol{p}_c$ can be commanded as pseudo controls, the command trajectory needs to be smooth such that its second time derivative exists. Therefore, linear reference models are used to generate the reference trajectories $\boldsymbol{p}_R$, computed in the 0-frame and governed by the following equation:

$$\ddot{\boldsymbol{p}}_R = \omega_0^2 \cdot (\boldsymbol{p}_c - \boldsymbol{p}_R) - 2\zeta\omega_0 \cdot \dot{\boldsymbol{p}}_R \tag{15}$$

The reference dynamics can be set by the natural frequency ω_0 and the relative damping ζ. For the controller presented in this paper, the damping is set to 1 to ensure aperiodic behavior and the natural frequency to 2.5 based on experiments. The error controller, computing the pseudo controls can now be designed and is governed by the following equation:

$$\boldsymbol{\nu} = \ddot{\boldsymbol{p}}_R + (\dot{\boldsymbol{p}}_R - \dot{\boldsymbol{p}}) \cdot k_d + (\boldsymbol{p}_R - \boldsymbol{p}) \cdot k_p \tag{16}$$

Where $\boldsymbol{p}$ is the filtered position and $\dot{\boldsymbol{p}}$ the filtered velocity from Section IV-D. k_p, k_d are the proportional and differential gains for the error controller.

At this point, one advantage of the system becomes obvious: Commanding a step signal for the position, a trajectory for acceleration, speed and position is computed. The trajectory for the acceleration is directly commanded to the vehicle and the trajectories for speed and position are controlled by the error controller. This results in acceleration at the beginning with maximum allowed value and slowing down with exactly the maximum available acceleration as well. An example of the generated reference trajectories is shown in Fig. 6

For the implementation, the Simulink framework is used as well, and all calculations are optimized for fixed-point arithmetics. Furthermore, limitations are introduced to limit

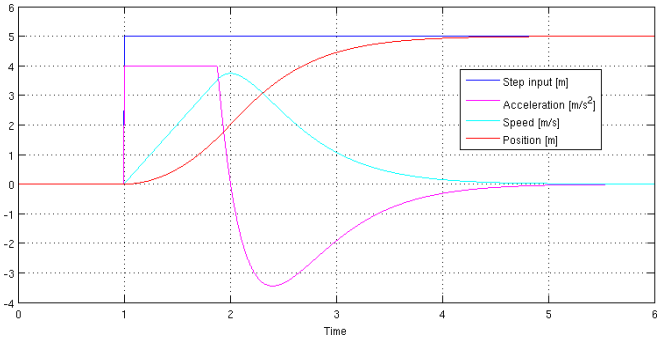


Fig. 6. Response of the reference model on an input step, in our case a change in the desired position. Note that the model outputs a negative acceleration after $t = 2\,\mathrm{s}$ to slow the vehicle down in advance in order to arrive the desired position fast and without overshoot. The acceleration was limited to $4\,\mathrm{m/s^2}$

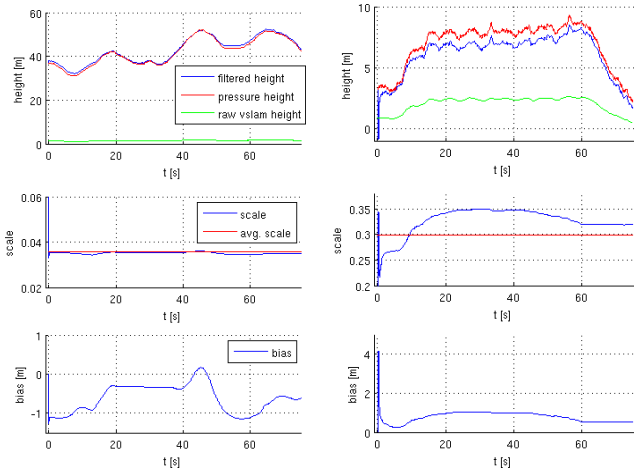


Fig. 7. Results of the scale filter for different heights. Top: estimated height from the filter (blue), raw height from VSLAM (green) and the raw height from the pressure sensor for comparison (red). Middle: absolute estimated scale compared to the average scale of the whole flight. Bottom: estimated sensor bias. Note the different scales on the plots

accelerations and attitude angles to reasonable values. A command filter was also implemented, giving the MAV's safety pilot the possibility to steer speed signals in the $\bar{0}$-frame with the sticks of his RC transmitter.

V. Experiments and Results

In this section, we evaluate the performance of the presented system, in particular of the scale filter and the position controller. The indoor experiments were made in the "Flying Machine Arena" [23] equipped with a Vicon motion capturing system which provided us ground truth with millimeter resolution at $200\,\mathrm{Hz}$. The outdoor experiments took place in a small park with grass and stones on the ground which provided sufficient texture.

The top plots of Fig. 7 show the height estimated by our filter (blue) compared to the raw measurements of the air pressure sensor (red) and the raw estimated height from the VSLAM in "VSLAM-units" (green). The middle plots show the estimated scale compared to the ratio of the mean height from the air pressure sensor and from the VSLAM. The estimated bias to compensate for the drifts of the air pressure sensor can be seen in the bottom plots. We initialized the filter with constant parameters at different height to verify that it still converges under different conditions. We unfortunately did not have ground truth accurate enough for these experiments. Therefore, we can only evaluate the plots qualitatively. What can be observed is that the filter converges after approximately $1\,\mathrm{s}$. For the pressure sensor bias, comparable drifts are observable when the vehicle is left on the ground over the same period of time.

For the position controller, the RMS error while hovering and the response to external disturbances or step inputs respectively is of interest. Fig. 8 and Fig. 9 show the trajectories of in- and outdoor flights. To gain the RMS error for the outdoor experiments, we computed the mean of the hovering phases and computed the error relative to this mean. Obviously, the RMS error outdoor is larger than indoor which is a result of the higher altitude and the less controlled environment, but the vehicle is still able to navigate stably. It can also be observed that the RMS error in the z-axis is significantly smaller than in the x/y-plane as depicted in Table I. This is because of the controller structure and dynamics of the MAV. While we can directly command acceleration in the z-axis through the thrust of the propellers, commands in the x/y-plane need to go through the attitude control loop first, and then result in acceleration (see Section IV-E, Fig. 5).

TABLE I
RMS error while hovering

Type	RMS error [m]	Height [m]
indoor x/y	0.069	1.4
indoor z	0.009	1.4
outdoor x/y	0.44	3.3
outdoor z	0.11	3.3

The left plot of Fig. 10 shows disturbances at $t = 6\,\mathrm{s}$ and $t = 20\,\mathrm{s}$ in the y-axis from pushing the vehicle. Note that the vehicle directly returns to its desired setpoint within the RMS with almost no overshoot. This is a result from the fast data fusion and of the position controller based on nonlinear dynamic inversion (see also Fig. 6). The middle

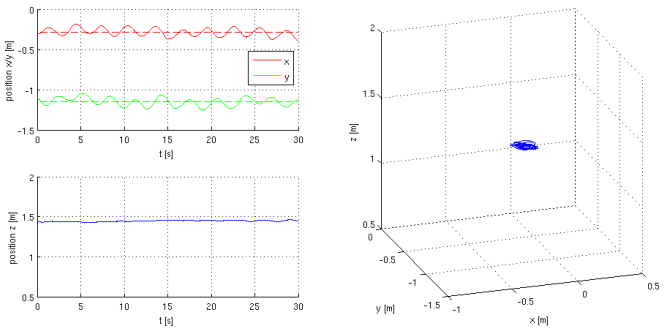


Fig. 8. Indoor hovering

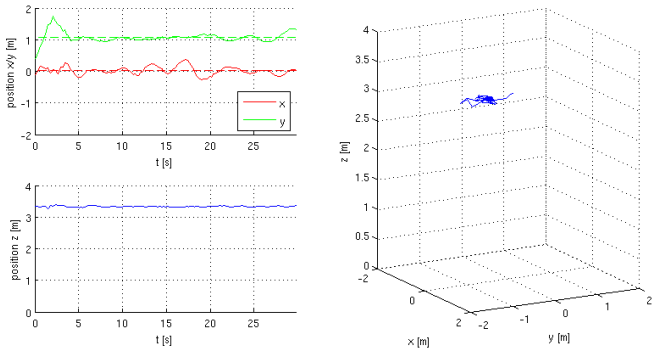


Fig. 9. Outdoor hovering

plot of Fig. 10 shows a longer term disturbance at $t = 14\,\mathrm{s}$ by pulling the vehicle and holding it for $3\,\mathrm{s}$. Again, the vehicle returns directly to the desired setpoint with almost no overshoot. The disturbances applied in the z-axis on the right plot of Fig. 10 look rather small. Because of the direct thrust command as explained above, the helicopter was massively working against the disturbance. Even though we disturbed the vehicle with reasonably strong impulses, we did not manage to deflect it more than $20\,\mathrm{cm}$.

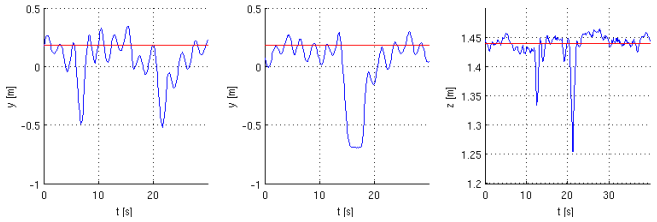


Fig. 10. Short and long disturbances in the y-axis on the left and the middle. Disturbances in the z-axis on the right. Note the different scale on the right plot

VI. Conclusions

We successfully stabilized a highly dynamic aerial vehicle based on onboard vision computation at a rate of only $10\,\mathrm{Hz}$, data fusion with IMU data and a well designed implementation. With the frameworks we developed, we are now able to try out relatively easy new control approaches on a real working system. This system can perform autonomous flights in unknown in- and outdoor environments, solely having a monocular camera as exteroceptive sensor while all computation is completely running onboard.

VII. Acknowledgments

We like to thank Prof. Raffaello D'Andrea and his group for giving us access and support for their "Flying Machine Arena" [23] to obtain ground truth for the indoor experiments. We also like to thank Prof. Florian Holzapfel of the Institute of Flight System Dynamics at the Technische Universität München and The Mathworks for their great cooperation concerning the used Simulink Framework for the microprocessor of the onboard FCU and the nonlinear control structures for the quadrocopter.

References

[1] Ascending Technologies GmbH, website, http://www.asctec.de.
[2] M. Bloesch, S. Weiss, D. Scaramuzza, and R. Siegwart, "Vision based mav navigation in unknown and unstructured environments," in *IEEE International Conference on Robotics and Automation*, 2010.
[3] D. Eberli, D. Scaramuzza, S. Weiss, and R. Siegwart, "Vision based position control for mavs using one single artificial landmark," in *International Conference & Exhibition on Unmanned Aerial Vehicles (UAV)*, 2010.
[4] M. Achtelik, A. Bachrach, R. He, S. Prentice, and N. Roy, "Stereo Vision and Laser Odometry for Autonomous Helicopters in GPS-denied Indoor Environments," in *Proceedings of the SPIE Unmanned Systems Technology XI*, 2009.
[5] E. Altug, J. Ostrowski, and R. Mahony, "Control of a quadrotor helicopter using visual feedback," in *International Conference on Robotics and Automation*, May 2002, pp. 72–77.
[6] S. Park, D. Won, M. Kang, T. Kim, H. Lee, and S. Kwon, "Ric (robust internal-loop compensator) based flight control of a quad-rotor type uav," in *International Conference on Intelligent Robots and Systems*, Aug. 2005, pp. 3542–3547.
[7] S. Klose, J. Wang, M. Achtelik, G. Panin, F. Holzapfel, and A. Knoll, "Markerless, Vision-Assisted Flight Control of a Quadrocopter," in *International Conference on Intelligent Robots and Systems*, October 2010.
[8] B. Ludington, E. Johnson, and G. Vachtsevanos, "Augmenting uav autonomy," *IEEE Robot. Automat. Mag.*, vol. 24, no. 5, pp. 63–71, Sept. 2006.
[9] T. Hamel, R. Mahony, and A. Chriette, "Visual servo trajectory tracking for a four rotor vtol aerial vehicle," in *International Conference on Robotics and Automation*, May 2002, pp. 2781–2786.
[10] T. Cheviron, T. Hamel, R. Mahony, and G. Baldwin, "Robust nonlinear fusion of inertial and visual data for position, velocity and attitude estimation of uav," in *International Conference on Robotics and Automation*, Apr. 2007, pp. 2010–2016.
[11] S. Zingg, D. Scaramuzza, S. Weiss, and R. Siegwart, "Mav obstacle avoidance using optical flow," in *International Conference on Robotics and Automation*, May 2010.
[12] B. herisse, F. Russotto, T. Hamel, and R. Mahony, "Hovering flight and vertical landing control of a vtol unmanned aerial vehicle using optical flow," in *International Conference on Intelligent Robots and Systems*, Sept. 2008, pp. 801–806.
[13] S. Hrabar, G. Sukhatme, P. Corke, K. Usher, and J. Roberts, "Combined optic-flow and stereo-based navigation of urban canyons for a uav," in *International Conference on Intelligent Robots and Systems*, Aug. 2005, pp. 3309–3316.
[14] Parrot S.A., "AR.Drone," website, http://ardrone.parrot.com.
[15] S. Ahrens, D. Levine, G. Andrews, and J. How, "Vision-based guidance and control of a hovering vehicle in unknown, gps-denied environments," in *International Conference on Robotics and Automation*, May 2009.
[16] A. Davison, I. Reid, N. Molton, and O. Strasse, "Monoslam: Real-time single camera slam," *IEEE Trans. Pattern Anal. Machine Intell.*, vol. 29, no. 6, pp. 1052–1067, June 2007.
[17] S. Bouabdallah and R. Siegwart, "Backstepping and sliding-mode techniques applied to an indoor micro quadrotor," in *International Conference on Robotics and Automation*, Apr. 2005, pp. 2247–2252.
[18] G. Klein and D. Murray, "Parallel tracking and mapping for small ar workspaces," in *International Symposium on Mixed and Augmented Reality*, Nov. 2007, pp. 225–234.
[19] M. Quigley, K. Conley, B. P. Gerkey, J. Faust, T. Foote, J. Leibs, R. Wheeler, and A. Y. Ng, "ROS: an open-source Robot Operating System," in *ICRA Workshop on Open Source Software*, 2009.
[20] D. Gurdan, J. Stumpf, M. Achtelik, K.-M. Doth, G. Hirzinger, and D. Rus, "Energy-efficient autonomous four-rotor flying robot controlled at 1 khz," in *IEEE International Conference on Robotics and Automation*, Roma, Italy, Apr. 2007, pp. 361 – 366.
[21] K. Narendra and A. Annaswamy, *Stable Adaptive Control*. Prentice Hall, 1990.
[22] A. Isidori, *Nonlinear Control Systems*, 3rd ed. Springer, 1995.
[23] S. Lupashin, A. Schöllig, M. Sherback, and R. D'Andrea, "A simple learning strategy for high-speed quadrocopter multi-flips," in *IEEE International Conference on Robotics and Automation*, May. 2010.

Incremental Sampling-based Algorithms for Optimal Motion Planning

Sertac Karaman Emilio Frazzoli

Abstract— **During the last decade, incremental sampling-based motion planning algorithms, such as the Rapidly-exploring Random Trees (RRTs), have been shown to work well in practice and to possess theoretical guarantees such as probabilistic completeness. However, no theoretical bounds on the quality of the solution obtained by these algorithms, e.g., in terms of a given cost function, have been established so far. The purpose of this paper is to fill this gap, by designing efficient incremental sampling-based algorithms with provable optimality properties. The first contribution of this paper is a negative result: it is proven that, under mild technical conditions, the cost of the best path returned by RRT converges almost surely to a non-optimal value, as the number of samples increases. Second, a new algorithm is considered, called the Rapidly-exploring Random Graph (RRG), and it is shown that the cost of the best path returned by RRG converges to the optimum almost surely. Third, a tree version of RRG is introduced, called RRT*, which preserves the asymptotic optimality of RRG while maintaining a tree structure like RRT. The analysis of the new algorithms hinges on novel connections between sampling-based motion planning algorithms and the theory of random geometric graphs. In terms of computational complexity, it is shown that the number of simple operations required by both the RRG and RRT* algorithms is asymptotically within a constant factor of that required by RRT.**

I. Introduction

Even though modern robots may posses significant differences in sensing, actuation, size, application, or workspace, the motion planning problem, i.e., the problem of planning a dynamically feasible trajectory through a complex environment cluttered with obstacles, is embedded and essential in almost all robotics applications. Moreover, this problem has several applications in other disciplines such as verification, computational biology, and computer animation [1]–[5].

Motion planning has been a highly active area of research since the late 1970s. Early approaches to the problem has mainly focused on the development of *complete* planners (see, e.g., [6]), which find a solution if one exists and return failure otherwise. However, it was established as early as 1979 that even a most basic version of the motion planning problem, called the piano mover's problem, is known to be PSPACE-hard [7], which strongly suggests that complete planners are doomed to suffer from computational complexity.

Tractable algorithms approach the motion planning problem by relaxing the completeness requirement to, for instance, *resolution completeness*, which amounts to finding a solution, if one exists, when the resolution parameter of the algorithm is set fine enough. Most motion planning methods that are based on gridding or cell decomposition fall into this category. A more recent line of research that has achieved a great success has focused on the construction of paths connecting randomly-sampled points. Algorithms such as Probabilistic RoadMaps (PRM) [8] have been shown to be *probabilistically complete*, i.e., such that the probability of finding a solution, if one exists, approaches one as the number of samples approaches infinity.

The PRM algorithm constructs a graph of feasible paths off-line, and is primarily aimed at multiple-query applications, in which several motion-planning problems need to be solved in the same environment. Incremental sampling-based algorithms have been developed for single-query, real-time applications; among the most influential of these, one can mention Rapidly-exploring Random Trees (RRT) [9], and the algorithm in [10]. These algorithms have been shown to be probabilistically complete, with an exponential decay of the probability of failure. Moreover, RRTs were demonstrated in various robotic platforms in major robotics events (see, e.g., [11]).

A class of incremental sampling-based motion planning algorithms that is worth mentioning at this point is the Rapidly-exploring Random Graphs (RRGs), which were proposed in [12] to find feasible trajectories that satisfy specifications other than the usual "avoid all the obstacles and reach the goal region". More generally, RRGs can handle specifications given in the form of deterministic μ-calculus, which includes the widely-used Linear Temporal Logic (LTL). RRGs incrementally build a graph of trajectories, since specifications given in μ-calculus, in general, require infinite-horizon looping trajectories, which are not included in trees.

To address the challenges posed by real-time applications, state-of-the-art motion planning algorithms, such as RRTs, are tailored to return a feasible solution quickly, paying almost no attention to the "quality" of the solution. On the other hand, in typical implementations [11], the algorithm is not terminated as soon as the first feasible solution is found; rather, all the available computation time is used to search for an improved solution, with respect to a performance metric such as time, path length, fuel consumption, etc. A shortcoming of this approach is that there is no guarantee that the computation will eventually converge to optimal trajectories. In fact, despite the clear practical need, there has been little progress in characterizing optimality properties of sampling-based motion planning algorithms, even though the importance of these problems was emphasized in early seminal papers such as [9].

Yet, the importance of the quality of the solution returned by the planners has been noticed, in particular, from the point of view of incremental sampling-based motion planning. In [13], Urmson and Simmons have proposed heuristics to bias the tree growth in the RRT towards those regions that result in low-cost solutions. They have also shown experimental results

The authors are with the Laboratory for Information and Decision Systems, Massachusetts Institute of Technology, Cambridge, MA.

evaluating the performance of different heuristics in terms of the quality of the solution returned. In [14], Ferguson and Stentz have considered running the RRT algorithm multiple times in order to progressively improve the quality of the solution. They showed that each run of the algorithm results in a path with smaller cost, even though the procedure is not guaranteed to converge to an optimal solution.

To the best of the authors' knowledge, this paper provides the first thorough analysis of optimality properties of incremental sampling-based motion planning algorithms. In particular, it is shown that the probability that the RRT converges to an optimal solution, as the number of samples approaches infinity, is zero under some reasonable technical assumptions. In fact, the RRT algorithm almost always converges to a non-optimal solution. Second, it is shown that the probability of the same event for the RRG algorithm is one. That is, the RRG algorithm is asymptotically optimal, in the sense that it converges to an optimal solution almost surely as the number of samples approaches infinity. Third, a novel variant of the RRG algorithm is introduced, called RRT*, which inherits the asymptotic optimality of the RRG algorithm while maintaining a tree structure. To do so, the RRT* algorithm essentially "rewires" the tree as it discovers new lower-cost paths reaching the nodes that are already in the tree. Finally, it is shown that the asymptotic computational complexity of the RRG and RRT* algorithms is essentially the same as that of RRTs.

To the authors' knowledge, the algorithms considered in this paper are the first computationally efficient incremental sampling-based motion planning algorithms with asymptotic optimality guarantees. Indeed, the results in this paper imply that these algorithms are optimal also from an asymptotic computational complexity point of view, since they closely match lower bounds for computing nearest neighbors. The key insight is that connections between vertices in the graph should be sought within balls whose radius vanishes with a certain rate as the size of the graph increases, and is based on new connections between motion planning and the theory of random geometric graphs [15], [16].

The paper is organized as follows. Section II lays the ground in terms of notation and problem formulation. Section III is devoted to the introduction of the RRT and RRG algorithms. In Section IV, these algorithms are analyzed in terms of probabilistic completeness, asymptotic optimality, and computational complexity. The RRT* algorithm is presented in Section V, where it is shown that RRT* inherits the theoretical guarantees of the RRG algorithm. Experimental results are presented and discussed in Section VI. Concluding remarks and directions for future work are given in Section VII.

Due to space limitations, results are stated without formal proofs. An extended version of this paper, including proofs of the major results, technical discussions, and extensive experimental results, is available [17]. An implementation of the RRT* algorithm in the C language is available at `http://ares.lids.mit.edu/software`.

The focus of this paper is on the basic problem of navigating through a connected bounded subset of a d-dimensional Euclidean space. However, the proposed algorithms also extend to systems with differential constraints, as shown in [18].

II. Preliminary Material

A. Notation

A sequence on a set A, denoted as $\{a_i\}_{i\in\mathbb{N}}$, is a mapping from $\mathbb{N}$ to A with $i \mapsto a_i$. Given $a, b \in \mathbb{R}$, the closed interval between a and b is denoted by $[a, b]$. The Euclidean norm is denoted by $\|\cdot\|$. Given a set $X \subset \mathbb{R}^d$, the closure of X is denoted by $\mathrm{cl}(X)$, the Lebesgue measure of X, i.e., its volume, is denoted by $\mu(X)$. The closed ball of radius $r > 0$ centered at $x \in \mathbb{R}^d$ is defined as $\mathcal{B}_{x,r} := \{y \in \mathbb{R}^d \mid \|y - x\| \leq r\}$. The volume of the unit ball in $\mathbb{R}^d$ is denoted by ζ_d.

Given a set $X \subset \mathbb{R}^d$, and a scalar $s \geq 0$, a path in X is a continuous function $\sigma : [0, s] \to X$, where s is the length of the path defined in the usual way. Given two paths in X, $\sigma_1 : [0, s_1] \to X$, and $\sigma_2 : [0, s_2] \to X$, with $\sigma_1(s_1) = \sigma_2(0)$, their concatenation is denoted by $\sigma_1|\sigma_2$, i.e., $\sigma = \sigma_1|\sigma_2 : [0, s_1 + s_2] \to X$ with $\sigma(s) = \sigma_1(s)$ for all $s \in [0, s_1]$, and $\sigma(s) = \sigma_2(s - s_1)$ for all $s \in [s_1, s_1 + s_2]$. The set of all paths in X with nonzero length is denoted by Σ_X. The straight continuous path between $x_1, x_2 \in \mathbb{R}^d$ is denoted by $\mathtt{Line}(x_1, x_2)$.

Let $(\Omega, \mathcal{F}, \mathbb{P})$ be a probability space. A random variable is a measurable function from Ω to $\mathbb{R}$; an extended random variable can also take the values $\pm\infty$. A sequence $\{\mathcal{Y}_i\}_{i\in\mathbb{N}}$ of random variables is said to converge surely to a random variable $\mathcal{Y}$ if $\lim_{i\to\infty} \mathcal{Y}_i(\omega) = \mathcal{Y}(\omega)$ for all $\omega \in \Omega$; the sequence is said to converge almost-surely if $\mathbb{P}(\{\lim_{i\to\infty} \mathcal{Y}_i = \mathcal{Y}\}) = 1$.

B. Problem Formulation

In this section, two variants of the path planning problem are presented. First, the feasibility problem in path planning is formalized, then the optimality problem is introduced.

Let X be a bounded connected open subset of $\mathbb{R}^d$, where $d \in \mathbb{N}$, $d \geq 2$. Let X_{obs} and X_{goal}, called the *obstacle region* and the *goal region*, respectively, be open subsets of X. Let us denote the *obstacle-free space*, i.e., $X \setminus X_{\mathrm{obs}}$, as X_{free}. Let the *initial state*, x_{init}, be an element of X_{free}. In the sequel, a path in X_{free} is said to be a *collision-free path*. A collision-free path that starts at x_{init} and ends in the goal region is said to be a *feasible path*, i.e., a collision-free path $\sigma : [0, s] \to X_{\mathrm{free}}$ is feasible if and only if $\sigma(0) = x_{\mathrm{init}}$ and $\sigma(s) \in X_{\mathrm{goal}}$.

The *feasibility problem* of path planning is to find a feasible path, if one exists, and report failure otherwise.

Problem 1 (Feasible planning) *Given a bounded connected open set $X \subset \mathbb{R}^d$, an obstacle space $X_{\mathrm{obs}} \subset X$, an initial state $x_{\mathrm{init}} \in X_{\mathrm{free}}$, and a goal region $X_{\mathrm{goal}} \subset X_{\mathrm{free}}$, find a path $\sigma : [0, s] \to X_{\mathrm{free}}$ such that $\sigma(0) = x_{\mathrm{init}}$ and $\sigma(s) \in X_{\mathrm{goal}}$, if one exists. If no such path exists, then report failure.*

Let $c : \Sigma_{X_{\mathrm{free}}} \to \mathrm{R}_{>0}$ be a function, called the *cost function*, which assigns a non-negative cost to all nontrivial collision-free paths. The *optimality problem* of path planning asks for finding a feasible path with minimal cost.

Problem 2 (Optimal planning) *Given a bounded connected open set X, an obstacle space X_{obs}, an initial state x_{init}, and a goal region X_{goal}, find a path $\sigma^* : [0, s] \to \mathrm{cl}(X_{\mathrm{free}})$ such that (i) $\sigma^*(0) = x_{\mathrm{init}}$ and $\sigma^*(s) \in X_{\mathrm{goal}}$, and (ii) $c(\sigma^*) = \min_{\sigma\in\Sigma_{\mathrm{cl}(X_{\mathrm{free}})}} c(\sigma)$. If no such path exists, then report failure.*

III. ALGORITHMS

In this section, two incremental sampling-based motion planning algorithms, namely the RRT and the RRG algorithms, are introduced. Before formalizing the algorithms, let us note the primitive procedures that they rely on.

Sampling: The function $\texttt{Sample} : \mathbb{N} \to X_{\text{free}}$ returns independent identically distributed (i.i.d.) samples from X_{free}.

Steering: Given two points $x, y \in X$, the function $\texttt{Steer} : (x, y) \mapsto z$ returns a point $z \in \mathbb{R}^d$ such that z is "closer" to y than x is. Throughout the paper, the point z returned by the function Steer will be such that z minimizes $\|z - y\|$ while at the same time maintaining $\|z - x\| \leq \eta$, for a prespecified $\eta > 0$, i.e., $\texttt{Steer}(x, y) = \text{argmin}_{z \in \mathbb{R}^d, \|z-x\| \leq \eta} \|z - y\|$.

Nearest Neighbor: Given a graph $G = (V, E)$ and a point $x \in X_{\text{free}}$, the function $\texttt{Nearest} : (G, x) \mapsto v$ returns a vertex $v \in V$ that is "closest" to x in terms of a given distance function. In this paper, we will use Euclidean distance (see, e.g., [9] for alternative choices), i.e., $\texttt{Nearest}(G = (V, E), x) = \text{argmin}_{v \in V} \|x - v\|$.

Near Vertices: Given a graph $G = (V, E)$, a point $x \in X_{\text{free}}$, and a number $n \in \mathbb{N}$, the function $\texttt{Near} : (G, x, n) \mapsto V'$ returns a set V' of vertices such that $V' \subseteq V$. The Near procedure can be thought of as a generalization of the nearest neighbor procedure in the sense that the former returns a collection of vertices that are close to x, whereas the latter returns only one such vertex that is the closest. Just like the Nearest procedure, there are many ways to define the Near procedure, each of which leads to different algorithmic properties. For technical reasons to become clear later, we define $\texttt{Near}(G, x, n)$ to be the set of all vertices within the closed ball of radius r_n centered at x, where $r_n = \min \left\{ \left(\frac{\gamma}{\zeta_d} \frac{\log n}{n} \right)^{1/d}, \eta \right\}$, and γ is a constant. Hence, the volume of this ball is $\min\{\gamma \frac{\log n}{n}, \zeta_d \eta^d\}$.

Collision Test: Given two points $x, x' \in X_{\text{free}}$, the Boolean function $\texttt{ObstacleFree}(x, x')$ returns True iff the line segment between x and x' lies in X_{free}, i.e., $[x, x'] \subset X_{\text{free}}$.

Both the RRT and the RRG algorithms are similar to most other incremental sampling-based planning algorithms (see Algorithm 1). Initially, the algorithms start with the graph that includes the initial state as its single vertex and no edges; then, they incrementally grow a graph on X_{free} by sampling a state x_{rand} from X_{free} at random and extending the graph towards x_{rand}. In the sequel, every such step of sampling followed by extensions (Lines 2-5 of Algorithm 1) is called a single *iteration* of the incremental sampling-based algorithm.

Hence, the body of both algorithms, given in Algorithm 1, is the same. However, RRGs and RRTs differ in the choice of the vertices to be extended. The Extend procedures for the RRT and the RRG algorithms are provided in Algorithms 2 and 3, respectively. Informally speaking, the RRT algorithm extends the nearest vertex towards the sample. The RRG algorithm first extends the nearest vertex, and if such extension is successful, it also extends all the vertices returned by the Near procedure, producing a graph in general. In both cases, all the extensions resulting in collision-free trajectories are added to the graph as edges, and their terminal points as new vertices.

Algorithm 1: Body of RRT and RRG Algorithms

1 $V \leftarrow \{x_{\text{init}}\}$; $E \leftarrow \emptyset$; $i \leftarrow 0$;
2 **while** $i < N$ **do**
3 $\quad G \leftarrow (V, E)$;
4 $\quad x_{\text{rand}} \leftarrow \texttt{Sample}(i)$; $i \leftarrow i + 1$;
5 $\quad (V, E) \leftarrow \texttt{Extend}(G, x_{\text{rand}})$;

Algorithm 2: $\texttt{Extend}_{RRT}(G, x)$

1 $V' \leftarrow V$; $E' \leftarrow E$;
2 $x_{\text{nearest}} \leftarrow \texttt{Nearest}(G, x)$;
3 $x_{\text{new}} \leftarrow \texttt{Steer}(x_{\text{nearest}}, x)$;
4 **if** $\texttt{ObstacleFree}(x_{\text{nearest}}, x_{\text{new}})$ **then**
5 $\quad V' \leftarrow V' \cup \{x_{\text{new}}\}$;
6 $\quad E' \leftarrow E' \cup \{(x_{\text{nearest}}, x_{\text{new}})\}$;
7 **return** $G' = (V', E')$

IV. ANALYSIS

A. Convergence to a Feasible Solution

In this section, the feasibility problem is considered. It is proven that the RRG algorithm inherits the probabilistic completeness as well as the exponential decay of the probability of failure (as the number of samples increase) from the RRT. These results imply that the RRT and RRG algorithms have the same performance in producing a solution to the feasibility problem as the number of samples increase.

Sets of vertices and edges of the graphs maintained by the RRT and the RRG algorithms can be defined as functions from the sample space Ω to appropriate sets. More precisely, let $\{\mathcal{V}_i^{\text{RRT}}\}_{i \in \mathbb{N}}$ and $\{\mathcal{V}_i^{\text{RRG}}\}_{i \in \mathbb{N}}$, sequences of functions defined from Ω into finite subsets of X_{free}, be the sets of vertices in the RRT and the RRG, respectively, at the end of iteration i. By convention, we define $\mathcal{V}_0^{\text{RRT}} = \mathcal{V}_0^{\text{RRG}} = \{x_{\text{init}}\}$. Similarly, let $\mathcal{E}_i^{\text{RRT}}$ and $\mathcal{E}_i^{\text{RRG}}$, defined for all $i \in \mathbb{N}$, denote the set of edges in the RRT and the RRG, respectively, at the end of iteration i. Clearly, $\mathcal{E}_0^{\text{RRT}} = \mathcal{E}_0^{\text{RRG}} = \emptyset$.

An important lemma used for proving the equivalency between the RRT and the RRG algorithms is the following.

Lemma 3 *For all $i \in \mathbb{N}$ and all $\omega \in \Omega$, $\mathcal{V}_i^{\text{RRT}}(\omega) = \mathcal{V}_i^{\text{RRG}}(\omega)$ and $\mathcal{E}_i^{\text{RRT}}(\omega) \subseteq \mathcal{E}_i^{\text{RRG}}(\omega)$.*

Lemma 3 implies that the paths discovered by the RRT

Algorithm 3: $\texttt{Extend}_{RRG}(G, x)$

1 $V' \leftarrow V$; $E' \leftarrow E$;
2 $x_{\text{nearest}} \leftarrow \texttt{Nearest}(G, x)$;
3 $x_{\text{new}} \leftarrow \texttt{Steer}(x_{\text{nearest}}, x)$;
4 **if** $\texttt{ObstacleFree}(\texttt{x}_{\text{nearest}}, \texttt{x}_{\text{new}})$ **then**
5 $\quad V' \leftarrow V' \cup \{x_{\text{new}}\}$;
6 $\quad E' \leftarrow E' \cup \{(x_{\text{nearest}}, x_{\text{new}}), (x_{\text{new}}, x_{\text{nearest}})\}$;
7 $\quad X_{\text{near}} \leftarrow \texttt{Near}(G, x_{\text{new}}, |V|)$;
8 $\quad$ **for** *all* $x_{\text{near}} \in X_{\text{near}}$ **do**
9 $\quad\quad$ **if** $\texttt{ObstacleFree}(x_{\text{new}}, x_{\text{near}})$ **then**
10 $\quad\quad\quad E' \leftarrow E' \cup \{(x_{\text{near}}, x_{\text{new}}), (x_{\text{new}}, x_{\text{near}})\}$;
11 **return** $G' = (V', E')$

algorithm by the end of iteration i is, essentially, a subset of those discovered by the RRG by the end of the same iteration.

An algorithm addressing Problem 1 is said to be *probabilistically complete* if it finds a feasible path with probability approaching one as the number of iterations approaches infinity. Note that there exists a collision-free path starting from x_{init} to any vertex in the tree maintained by the RRT, since the RRT maintains a connected graph on X_{free} that necessarily includes x_{init}. Using this fact, the probabilistic completeness property of the RRT is stated alternatively as follows.

Theorem 4 (see [9]) *If there exists a feasible solution to Problem 1, then* $\lim_{i\to\infty} \mathbb{P}\left(\left\{\mathcal{V}_i^{\text{RRT}} \cap X_{\text{goal}} \neq \emptyset\right\}\right) = 1$.

An *attraction sequence* [9] is defined as a finite sequence $\mathcal{A} = \{A_1, A_2, \ldots, A_k\}$ of sets as follows: (i) $A_0 = \{x_{\text{init}}\}$, and (ii) for each set A_i, there exists a set B_i, called the *basin* such that for any $x \in A_{i-1}$, $y \in A_i$, and $z \in X \setminus B_i$, there holds $\|x - y\| \leq \|x - z\|$ Given an attraction sequence $\mathcal{A}$ of length k, let p_k denote $\min_{i\in\{1,2,\ldots,k\}} \left(\frac{\mu(A_i)}{\mu(X_{\text{free}})}\right)$.

The following theorem states that the probability that the RRT algorithm fails to return a solution, when one exists, decays to zero exponentially fast.

Theorem 5 (see [9]) *If there exists an attraction sequence* $\mathcal{A}$ *of length* k*, then* $\mathbb{P}\left(\left\{\mathcal{V}_i^{\text{RRT}} \cap X_{\text{goal}} = \emptyset\right\}\right) \leq e^{-\frac{1}{2}(i\,p_k - 2k)}$.

With Lemma 3 and Theorems 4 and 5, the following theorem is immediate.

Theorem 6 *If there exists a feasible solution to Problem 1, then* $\lim_{i\to\infty} \mathbb{P}\left(\left\{\mathcal{V}_i^{\text{RRG}} \cap X_{\text{goal}} \neq \emptyset\right\}\right) = 1$. *Moreover, if an attraction sequence* $\mathcal{A}$ *of length* k *exists, then* $\mathbb{P}\left(\left\{\mathcal{V}_i^{\text{RRG}} \cap X_{\text{goal}} = \emptyset\right\}\right) \leq e^{-\frac{1}{2}(i\,p_k - 2\,k)}$.

B. Asymptotic Optimality

This section is devoted to the investigation of optimality properties of the RRT and the RRG algorithms. First, under some mild technical assumptions, it is shown that the probability that the RRT converges to an optimal solution is zero. However, the convergence of this random variable is guaranteed, which implies that the RRT converges to a *non*-optimal solution with probability one. On the contrary, it is subsequently shown that the RRG algorithm converges to an optimal solution almost-surely.

Let $\{\mathcal{Y}_i^{\text{RRT}}\}_{i\in\mathbb{N}}$ be a sequence of extended random variables that denote the cost of a minimum-cost path contained within the tree maintained by the RRT algorithm at the end of iteration i. The extended random variable $\mathcal{Y}_i^{\text{RRG}}$ is defined similarly. Let c^* denote the cost of a minimum-cost path in $\text{cl}(X_{\text{free}})$, i.e., the cost of a path that solves Problem 2.

Let us note that the limits of these two extended random variable sequences as i approaches infinity exist. More formally, notice that $\mathcal{Y}_{i+1}^{\text{RRT}}(\omega) \leq \mathcal{Y}_i^{\text{RRT}}(\omega)$ holds for all $i \in \mathbb{N}$ and all $\omega \in \Omega$. Moreover, $\mathcal{Y}_i^{\text{RRT}}(\omega) \geq c^*$ for all $i \in \mathbb{N}$ and all $\omega \in \Omega$, by optimality of c^*. Hence, $\{\mathcal{Y}_i^{\text{RRT}}\}_{i\in\mathbb{N}}$ is a surely non-increasing sequence of random variables that is surely lower-bounded by c^*. Thus, for all $\omega \in \Omega$, the limit $\lim_{i\to\infty} \mathcal{Y}_i^{\text{RRT}}(\omega)$ exists. The same argument also holds for the sequence $\{\mathcal{Y}_i^{\text{RRG}}\}_{i\in\mathbb{N}}$.

1) Almost Sure Nonoptimality of the RRT: Let Σ^* denote the set of all optimal paths, i.e., the set of all paths that solve Problem 2, and X_{opt} denote the set of states that an optimal path in Σ^* passes through, i.e., $X_{\text{opt}} = \cup_{\sigma^*\in\Sigma^*} \cup_{\tau\in[0,s^*]} \{\sigma^*(\tau)\}$. Consider the following assumptions.

Assumption 7 (Zero-measure Optimal Paths) *The set of all points in the state-space that an optimal trajectory passes through has measure zero, i.e.,* $\mu\left(X_{\text{opt}}\right) = 0$.

Assumption 8 (Sampling Procedure) *The sampling procedure is such that the samples* $\{\texttt{Sample}(i)\}_{i\in\mathbb{N}}$ *are drawn from an absolutely continuous distribution with a continuous density function* $f(x)$ *bounded away from zero on* X_{free}.

Assumption 9 (Monotonicity of the Cost Function) *For all* $\sigma_1, \sigma_2 \in \Sigma_{X_{\text{free}}}$*, the cost function* c *satisfies the following:* $c(\sigma_1) \leq c(\sigma_1|\sigma_2)$.

Assumption 7 rules out trivial cases, in which the RRT algorithm can sample exactly an optimal path with non-zero probability. Assumption 8 also ensures that the sampling procedure can not be tuned to construct the optimal path exactly. Finally, Assumption 9 merely states that extending a path to produce a longer path can not decrease its cost.

Recall that d denotes the dimensionality of the state space. The negative result of this section is formalized as follows.

Theorem 10 *Let Assumptions 7, 8, and 9 hold. Then, the probability that the cost of the minimum-cost path in the RRT converges to the optimal cost is zero, i.e.,*

$$\mathbb{P}\left(\left\{\lim_{i\to\infty} \mathcal{Y}_i^{\text{RRT}} = c^*\right\}\right) = 0,$$

whenever $d \geq 2$.

The key idea in proving this result is that the probability of extending a node on an optimal path (e.g., the root node) goes to zero very quickly, in such a way that any such node will only have a finite number of children, almost surely. Because of Assumptions 7 and 8, this implies the result.

As noted before, the limit $\lim_{i\to\infty} \mathcal{Y}_i^{\text{RRT}}(\omega)$ exists and is a random variable. However, Theorem 10 directly implies that this limit is strictly greater than c^* with probability one, i.e., $\mathbb{P}\left(\{\lim_{i\to\infty} \mathcal{Y}_i^{\text{RRT}} > c^*\}\right) = 1$. In other words, it is established, as a corollary, that the RRT algorithm converges to a nonoptimal solution with probability one.

It is interesting to note that, since the cost of the best path returned by the RRT algorithm converges to a random variable, say $\mathcal{Y}_\infty^{\text{RRT}}$, Theorem 10 provides new insight explaining the effectiveness of approaches as in [14]. In fact, running multiple instances of the RRT algorithm amounts to drawing multiple samples of $\mathcal{Y}_\infty^{RRT}$.

2) Almost Sure Optimality of the RRG: Consider the following set of assumptions, which will be required to show the asymptotic optimality of the RRG.

Assumption 11 (Additivity of the Cost Function) *For all* $\sigma_1, \sigma_2 \in \Sigma_{X_{\text{free}}}$*, the cost function* c *satisfies the following:* $c(\sigma_1|\sigma_2) = c(\sigma_1) + c(\sigma_2)$.

Assumption 12 (Continuity of the Cost Function) *The cost function c is Lipschitz continuous in the following sense: there exists some constant κ such that for any two paths $\sigma_1 : [0, s_1] \to X_{\text{free}}$ and $\sigma_2 : [0, s_2] \to X_{\text{free}}$, $|c(\sigma_1) - c(\sigma_2)| \leq \kappa \sup_{\tau \in [0,1]} \|\sigma_1(\tau\, s_1) - \sigma_2(\tau\, s_2)\|$.*

Assumption 13 (Obstacle Spacing) *There exists a constant $\delta \in \mathbb{R}_+$ such that for any point $x \in X_{\text{free}}$, there exists $x' \in X_{\text{free}}$, such that (i) the δ-ball centered at x' lies inside X_{free}, i.e., $\mathcal{B}_{x',\delta} \subset X_{\text{free}}$, and (ii) x lies inside the δ-ball centered at x', i.e., $x \in \mathcal{B}_{x',\delta}$.*

Assumption 12 ensures that two paths that are very close to each other have similar costs. Let us note that several cost functions of practical interest satisfy Assumptions 11 and 12. Assumption 13 is a rather technical assumption, which ensures existence of some free space around the optimal trajectories to allow convergence. For simplicity, it is assumed that the sampling is uniform, although the results can be directly extended to more general sampling procedures.

Recall that d is the dimensionality of the state-space X, and γ is the constant defined in the `Near` procedure. The positive result that states the asymptotic optimality of the RRG algorithm can be formalized as follows.

Theorem 14 *Let Assumptions 11, 12, and 13 hold, and assume that Problem 1 admits a feasible solution. Then, the cost of the minimum-cost path in the RRG converges to the optimal cost almost-surely, i.e.,*

$$\mathbb{P}\left(\left\{\lim_{i\to\infty} \mathcal{Y}_i^{\text{RRG}} = c^*\right\}\right) = 1,$$

whenever $d \geq 2$ and $\gamma > \gamma_L := 2^d(1 + 1/d)\mu(X_{\text{free}})$.

This result relies on the fact that a random geometric graph with n vertices formed by connecting each vertex with vertices within a distance of $d_n = \gamma' (\log n \,/\, n)^{1/d}$ will result in a connected graph almost surely as $n \to \infty$, whenever γ' is larger than a certain lower bound γ_1 [19]. In fact, the bound on γ' is a *tight threshold* in the sense that there exists an upper bound $\gamma_2 < \gamma_1$ such that, if $\gamma' < \gamma_2$, then the resulting graph will be disconnected almost surely [19]. This result strongly suggests that shrinking the ball in the `Near` procedure faster than the rate proposed will not yield an asymptotically optimal algorithm. The authors have experienced this fact in simulation studies: setting γ to around one third of γ_L does not seem to provide the asymptotic optimality property. On the other hand, as it will be shown in the next section, if the size of the same ball is reduced slower than the proposed rate, then the asymptotic complexity of the resulting algorithm will not be the same as the RRT. Hence, scaling r_n with $(\log n \,/\, n)^{1/d}$ in the `Near` procedure, surprisingly, achieves the perfect balance between asymptotic optimality and low computational complexity, since relevant results in the theory of random geometric graphs and lower bounds on nearest neighbor computation strongly suggest that a different rate will lose either the former or the latter while failing to provide an extra benefit in any of the two.

C. Computational Complexity

The objective of this section is to compare the computational complexity of RRTs and RRGs. It is shown that these algorithms share essentially the same asymptotic computational complexity in terms of the number of calls to *simple operations* such as comparisons, additions, and multiplications.

Consider first the computational complexity of the RRT and the RRG algorithms in terms of the number of calls to the primitive procedures introduced in Section III. Notice that, in every iteration, the number of calls to `Sample`, `Steer`, and `Nearest` procedures are the same in both algorithms. However, number of calls to `Near` and `ObstacleFree` procedures differ: the former is never called by the RRT and is called at most once by the RRG, whereas the latter is called exactly once by the RRT and at least once by the RRG.

Let $\mathcal{O}_i^{\text{RRG}}$ be a random variable that denotes the number of calls to the `ObstacleFree` procedure by the RRG algorithm in iteration i. Notice that, as an immediate corollary of Lemma 3, the number of vertices in the RRT and RRG algorithms is the same at any given iteration. Let $\mathcal{N}_i$ be the number of vertices in these algorithms at the end of iteration i. The following theorem establishes that the expected number of calls to the `ObstacleFree` procedure in iteration i by the RRG algorithm scales logarithmically with the number of vertices in the graph as i approaches infinity.

Lemma 15 *In the limit as i approaches infinity, the random variable $\mathcal{O}_i^{\text{RRG}} / \log(\mathcal{N}_i)$ is no more than a constant in expectation, i.e., $\limsup_{i\to\infty} \mathbb{E}\left[\frac{\mathcal{O}_i^{\text{RRG}}}{\log(\mathcal{N}_i)}\right] \leq \phi$, where $\phi \in \mathbb{R}_{>0}$ is a constant that depends only on the problem instance.*

However, some primitive procedures clearly take more computation time to execute than others. For a meaningful comparison, one should also evaluate the time required to execute each primitive procedure in terms of the number of simple operations (also called steps) that they perform. This analysis shows that the expected number of simple operations performed by the RRG is asymptotically within a constant factor of that performed by the RRT, which establishes that the RRT and the RRG algorithms have the same asymptotic computational complexity in terms of the number of steps that they perform.

First, notice that `Sample`, `Steer`, and `ObstacleFree` procedures can be performed in a constant number of steps, i.e., independent of the number of vertices in the graph.

Second, consider the computational complexity of the `Nearest` procedure. The problem of finding a nearest neighbor is widely studied, e.g., in the computer graphics literature. Even though algorithms that achieve sub-linear time complexity are known [20], lower bounds suggest that nearest neighbor computation requires at least logarithmic time [21]. In fact, assuming that the `Nearest` procedure computes an approximate nearest neighbor (see, e.g., [21] for a formal definition) using the algorithm given in [21], which is optimal in fixed dimensions from a computational complexity point of view closely matching a lower bound for tree-based algorithms, the `Nearest` algorithm has to run in $\Omega(\log n)$ time as formalized in the following lemma.

Let $\mathcal{M}_i^{\mathrm{RRT}}$ be the random variable that denotes the number of steps executed by the RRT algorithm in iteration i.

Lemma 16 *Assuming that* `Nearest` *is implemented using the algorithm given in [21], which is computationally optimal in fixed dimensions, the number of steps executed by the RRT algorithm at each iteration is at least order* $\log(\mathcal{N}_i)$ *in expectation in the limit, i.e., there exists a constant* $\phi_{RRT} \in \mathbb{R}_{>0}$ *such that* $\liminf_{i\to\infty} \mathbb{E}\left[\frac{\mathcal{M}_i^{\mathrm{RRT}}}{\log(\mathcal{N}_i)}\right] \geq \phi_{RRT}$.

Likewise, problems similar to that solved by the `Near` procedure are also widely studied in the literature, generally under the name of *range search problems*, as they have many applications in, for instance, computer graphics [20].

Similar to the nearest neighbor search, computing approximate solutions to the range search problem is computationally easier. A range search algorithm is said to be ε-approximate if it returns all vertices that reside in the ball of size r_n and no vertices outside a ball of radius $(1+\varepsilon)\, r_n$, but may or may not return the vertices that lie outside the former ball and inside the latter ball. In fixed dimensions, computing ε-approximate solutions can, in fact, be done in logarithmic time using polynomial space, in the worst case [22].

Note that the `Near` procedure can be implemented as an approximate range search while maintaining the asymptotic optimality guarantee. Notice that the expected number of vertices returned by the `Near` procedure also does not change, except by a constant factor. Hence, the `Near` procedure can be implemented to run in order $\log n$ expected time in the limit and linear space in fixed dimensions.

Let $\mathcal{M}_i^{\mathrm{RRG}}$ denote the number of steps performed by the RRG algorithm in iteration i. Then, together with Lemma 15, the discussion above implies the following lemma.

Lemma 17 *Assuming that the* `Near` *procedure is implemented using the algorithm given in [22], the number of steps executed by the RRG algorithm at each iteration is at most order* $\log(\mathcal{N}_i)$ *in expectation in the limit, i.e., there exists a constant* $\phi_{RRG} \in \mathbb{R}_{>0}$ *such that* $\limsup_{i\to\infty} \mathbb{E}\left[\frac{\mathcal{M}_i^{\mathrm{RRG}}}{\log(\mathcal{N}_i)}\right] \leq \phi_{RRG}$.

Finally, by Lemmas 16 and 17, we conclude that the RRT and the RRG algorithms have the same asymptotic computational complexity as stated in the following theorem.

Theorem 18 *Under the assumptions of Lemmas 16 and 17, there exists a constant* $\phi \in \mathbb{R}_{>0}$ *such that*

$$\limsup_{i\to\infty} \mathbb{E}\left[\frac{\mathcal{M}_i^{\mathrm{RRG}}}{\mathcal{M}_i^{\mathrm{RRT}}}\right] \leq \phi.$$

V. A Tree Version of the RRG Algorithm

Maintaining a tree structure rather than a graph may be advantageous in some applications, due to, for instance, relatively easy extensions to motion planning problems with differential constraints, or to cope with modeling errors. The RRG algorithm can also be slightly modified to maintain a tree structure, while preserving the asymptotic optimality properties as well the computational efficiency. In this section a tree version of the RRG algorithm, called RRT*, is introduced and analyzed.

A. The RRT* Algorithm

Given two points $x, x' \in X_{\mathrm{free}}$, recall that $\texttt{Line}(x, x') : [0, s] \to X_{\mathrm{free}}$ denotes the path defined by $\sigma(\tau) = \tau x + (s - \tau)x'$ for all $\tau \in [0, s]$, where $s = \|x' - x\|$. Given a tree $G = (V, E)$ and a vertex $v \in V$, let `Parent` be a function that maps v to the unique vertex $v' \in V$ such that $(v', v) \in E$.

The RRT* algorithm differs from the RRT and the RRG algorithms only in the way that it handles the `Extend` procedure. The body of the RRT* algorithm is presented in Algorithm 1 and the `Extend` procedure for the RRT* is given in Algorithm 4. In the description of the RRT* algorithm, the cost of the unique path from x_{init} to a vertex $v \in V$ is denoted by $\texttt{Cost}(v)$. Initially, $\texttt{Cost}(x_{\mathrm{init}})$ is set to zero.

Algorithm 4: $\texttt{Extend}_{RRT^*}(G, x)$

```
1  V' ← V; E' ← E;
2  x_nearest ← Nearest(G, x);
3  x_new ← Steer(x_nearest, x);
4  if ObstacleFree(x_nearest, x_new) then
5      V' ← V' ∪ {x_new};
6      x_min ← x_nearest;
7      X_near ← Near(G, x_new, |V|);
8      for all x_near ∈ X_near do
9          if ObstacleFree(x_near, x_new) then
10             c' ← Cost(x_near) + c(Line(x_near, x_new));
11             if c' < Cost(x_new) then
12                 x_min ← x_near;
13     E' ← E' ∪ {(x_min, x_new)};
14     for all x_near ∈ X_near \ {x_min} do
15         if ObstacleFree(x_new, x_near) and
           Cost(x_near) > Cost(x_new) + c(Line(x_new, x_near))
           then
16             x_parent ← Parent(x_near);
17             E' ← E' \ {(x_parent, x_near)};
               E' ← E' ∪ {(x_new, x_near)};
18 return G' = (V', E')
```

Similar to the RRT and RRG, the RRT* algorithm first extends the nearest neighbor towards the sample (Lines 2-3). However, it connects the new vertex, x_{new}, to the vertex that incurs the minimum accumulated cost up until x_{new} and lies within the set X_{near} of vertices returned by the `Near` procedure (Lines 6-13). RRT* also extends the new vertex to the vertices in X_{near} in order to "rewire" the vertices that can be accessed through x_{new} with smaller cost (Lines 14-17).

B. Convergence to a Feasible Solution

For all $i \in \mathbb{N}$, let $\mathcal{V}_i^{\mathrm{RRT}^*}$ and $\mathcal{E}_i^{\mathrm{RRT}^*}$ denote the set of vertices and the set of edges of the graph maintained by the RRT* algorithm, at the end of iteration i. The following lemma is the equivalent of Lemma 3.

Lemma 19 *For all* $i \in \mathbb{N}$ *and all* $\omega \in \Omega$, $\mathcal{V}_i^{\mathrm{RRT}^*}(\omega) = \mathcal{V}_i^{\mathrm{RRG}}(\omega)$, *and* $\mathcal{E}_i^{\mathrm{RRT}^*}(\omega) \subseteq \mathcal{E}_i^{\mathrm{RRG}}(\omega)$.

From Lemma 19 and Theorem 6, the following theorem, which asserts the probabilistic completeness and the exponential decay of failure probability of the RRT* algorithm, is immediate.

Theorem 20 *If there exists a feasible solution to Problem 1, then* $\lim_{i\to\infty} \mathbb{P}\left(\left\{\mathcal{V}_i^{\mathrm{RRT}^*} \cap X_{\mathrm{goal}} \neq \emptyset\right\}\right) = 1$. *Moreover, if an attraction sequence* $\mathcal{A}$ *of length* k *exists, then* $\mathbb{P}\left(\left\{\mathcal{V}_i^{\mathrm{RRT}^*} \cap X_{\mathrm{goal}} = \emptyset\right\}\right) \leq e^{-\frac{1}{2}(i\,p_k - 2\,k)}$.

C. Asymptotic Optimality

Let $\mathcal{Y}_i^{\mathrm{RRT}^*}$ be a random variable that denotes the cost of a minimum cost path in the tree maintained by the RRT* algorithm, at the end of iteration i. The following theorem ensures the asymptotic optimality of the RRT* algorithm.

Theorem 21 *Let Assumptions 11, 12, and 13 hold. Then, the cost of the minimum cost path in the RRT* converges to c^* almost surely, i.e.,* $\mathbb{P}\left(\left\{\lim_{i\to\infty} \mathcal{Y}_i^{\mathrm{RRT}^*} = c^*\right\}\right) = 1$.

D. Computational Complexity

Let $\mathcal{M}_i^{\mathrm{RRT}^*}$ be the number of steps performed by the RRT* algorithm in iteration i. The following theorem follows from Lemma 19 and Theorem 18.

Theorem 22 *Under the assumptions of Theorem 18, there exists a constant ϕ such that* $\limsup_{i\to\infty} \mathbb{E}\left[\frac{\mathcal{M}_i^{\mathrm{RRT}^*}}{\mathcal{M}_i^{\mathrm{RRT}}}\right] \leq \phi$.

VI. Simulations

This section presents simulation examples. A thorough simulation study of the algorithms can be found in [17].

The RRT and the RRT* algorithms are run in a square environment with obstacles and the cost function is set to the Euclidean path length. The trees maintained by the algorithms at different stages are shown in Figure 1. The figure illustrates that the RRT algorithm does not considerably improve the solution, whereas the RRT* algorithm converges towards an optimal solution by finding a feasible solution of the homotopy class that the optimal path lies in. An important difference between the RRT and the RRT* algorithms is the ability of the latter to efficiently consider different homotopy classes. Thus, in an environment cluttered with obstacles, the cost of first feasible solution found by the RRT or the RRT* algorithms can be drastically higher than the optimal cost. Although the RRT* algorithm efficiently improves the solution over time, the RRT algorithm tends to get stuck with the first solution found. In fact, Monte-Carlo runs of both algorithms, as shown in Figure 2.(a)-(b), illustrate that generally the RRT does not improve the first solution found, whereas the RRT* algorithm improves the solution significantly within the first few thousand iterations, for this particular scenario. Moreover, the cost of the best path in RRT seems to have high variance, while after a few thousand iterations the costs of the best path in the RRT* is almost the same in all runs, as expected from the theoretical results presented in the previous sections. Finally, the relative complexity of the two algorithms is demonstrated in Monte-Carlo runs in Figure 2.(c). Notice that the ratio of the running time of the algorithms up until a certain iteration converges to a constant as the number of iterations increases. Note that the convergence to this constant is achieved when the free space is "fully explored", i.e., almost uniformly filled with the nodes of the trees. However, before then the complexity of the RRT* is much lower than the complexity in the limit value. In fact, the average amount of time that the RRT* algorithm takes for finding a feasible solution was found to be no more than five times that of the RRT, in this particular scenario. Moreover, the first solution found by the RRT* generally costs considerably less than that found by the RRT.

VII. Conclusions and Future Work

This paper presented the results of a thorough analysis of the RRT and RRG algorithms for optimal motion planning. It is shown that, as the number of samples increases, the RRT algorithm converges to a sub-optimal solution almost surely. On the other hand, it is proven that the RRG algorithm has the asymptotic optimality property, i.e., almost sure convergence to an optimal solution, which the RRT algorithm lacked. The paper also proposed a novel algorithm called the RRT*, which inherits the asymptotic optimality property of the RRG, while maintaining a tree structure rather than a graph. The RRG and the RRT* were shown to have no significant overhead when compared to the RRT algorithm in terms of asymptotic computational complexity. Experimental evidence demonstrating the effectiveness of the proposed algorithms and supporting the theoretical claims was also provided.

The results reported in this paper can be extended in a number of directions, and applied to other sampling-based algorithms other than RRT. First of all, the proposed approach, building on the theory of random graphs to adjust the length of new connections can enhance the computational efficiency of PRM-based algorithms. Second, the algorithms and the analysis should be modified to address motion planning problems in the presence of differential constraints, also known as kino-dynamic planning problems. A third direction is the optimal planning problem in the presence of temporal/logic constraints on the trajectories, e.g., expressed using formal specification languages such as Linear Temporal Logic, or the μ-calculus. Such constraints correspond to, e.g., rules of the road constraints for autonomous ground vehicles, mission specifications for autonomous robots, and rules of engagement in military applications. Ultimately, incremental sampling-based algorithms with asymptotic optimality properties may provide the basic elements for the on-line solution of differential games, as those arising when planning in the presence of dynamic obstacles.

Finally, it is noted that the proposed algorithms may have applications outside of the robotic motion planning domain. In fact, the class of incremental sampling algorithm described in this paper can be readily extended to deal with problems described by partial differential equations, such as the eikonal equation and the Hamilton-Jacobi-Bellman equation.

Acknowledgments

The authors are grateful to Professors M.S. Branicky and G.J. Gordon for their insightful comments on a draft version of this paper. This research was supported in part by the Michigan/AFRL Collaborative Center on Control Sciences, AFOSR grant no. FA 8650-07-2-3744.

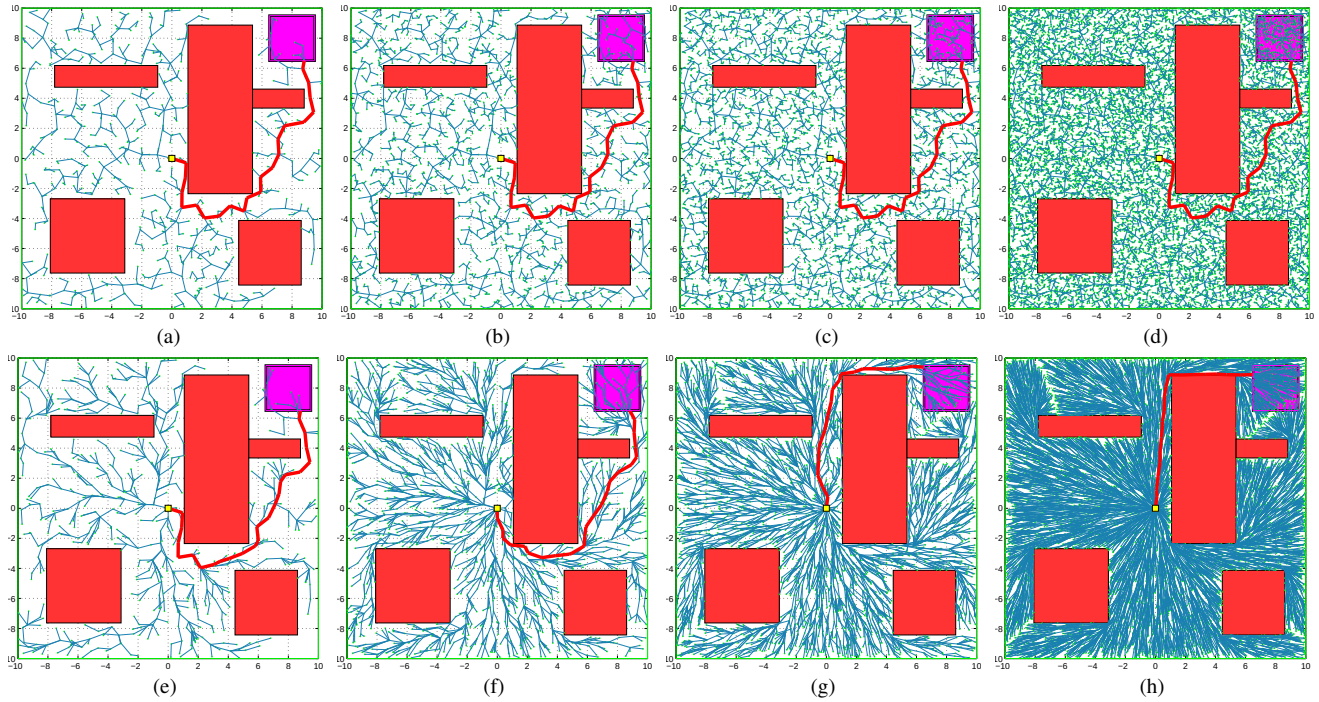


Fig. 1. A Comparison of the RRT* and RRT algorithms on a simulation example. The tree maintained by the RRT algorithm is shown in (a)-(d) in different stages, whereas that maintained by the RRT* algorithm is shown in (e)-(h). The tree snapshots (a), (e) are at 1000 iterations , (b), (f) at 2500 iterations, (c), (g) at 5000 iterations, and (d), (h) at 15,000 iterations. The goal regions are shown in magenta. The best paths that reach the target are highlighted with red.

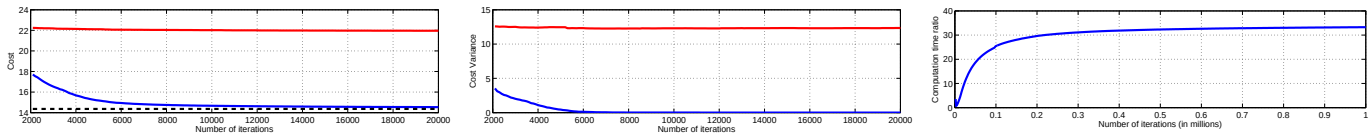


Fig. 2. The cost of the best paths in the RRT (shown in red) and the RRT* (shown in blue) plotted against iterations averaged over 500 trials in (a). The optimal cost is shown in black. The variance of the trials is shown in (b). A comparison of the running time of the RRT* and the RRT algorithms averaged over 50 trials is shown in (c); the ratio of the running time of the RRT* over that of the RRT up until each iteration is plotted versus the number of iterations.

REFERENCES

[1] J. Latombe. Motion planning: A journey of robots, molecules, digital actors, and other artifacts. *International Journal of Robotics Research*, 18(11):1119–1128, 1999.

[2] A. Bhatia and E. Frazzoli. Incremental search methods for reachability analysis of continuous and hybrid systems. In R. Alur and G.J. Pappas, editors, *Hybrid Systems: Computation and Control*, number 2993 in Lecture Notes in Computer Science, pages 142–156. Springer-Verlag, Philadelphia, PA, March 2004.

[3] M. S. Branicky, M. M. Curtis, J. Levine, and S. Morgan. Sampling-based planning, control, and verification of hybrid systems. *IEEE Proc. Control Theory and Applications*, 153(5):575–590, Sept. 2006.

[4] J. Cortes, L. Jailet, and T. Simeon. Molecular disassembly with RRT-like algorithms. In *IEEE International Conference on Robotics and Automation (ICRA)*, 2007.

[5] Y. Liu and N.I. Badler. Real-time reach planning for animated characters using hardware acceleration. In *IEEE International Conference on Computer Animation and Social Characters*, pages 86–93, 2003.

[6] J. T. Schwartz and M. Sharir. On the 'piano movers' problem: II. general techniques for computing topological properties of real algebraic manifolds. *Advances in Applied Mathematics*, 4:298–351, 1983.

[7] J.H. Reif. Complexity of the mover's problem and generalizations. In *Proceedings of the IEEE Symposium on Foundations of Computer Science*, 1979.

[8] L.E. Kavraki, P. Svestka, J Latombe, and M.H. Overmars. Probabilistic roadmaps for path planning in high-dimensional configuration spaces. *IEEE Transactions on Robotics and Automation*, 12(4):566–580, 1996.

[9] S. M. LaValle and J. J. Kuffner. Randomized kinodynamic planning. *International Journal of Robotics Research*, 20(5):378–400, May 2001.

[10] D. Hsu, R. Kindel, J. Latombe, and S. Rock. Randomized kinodynamic motion planning with moving obstacles. *International Journal of Robotics Research*, 21(3):233–255, 2002.

[11] Y. Kuwata, J. Teo, G. Fiore, S. Karaman, E. Frazzoli, and J.P. How. Real-time motion planning with applications to autonomous urban driving. *IEEE Transactions on Control Systems*, 17(5):1105–1118, 2009.

[12] S. Karaman and E. Frazzoli. Sampling-based motion planning with deterministic μ-calculus specifications. In *IEEE Conference on Decision and Control (CDC)*, 2009.

[13] C. Urmson and R. Simmons. Approaches for heuristically biasing RRT growth. In *Proceedings of the IEEE/RSJ International Conference on Robotics and Systems (IROS)*, 2003.

[14] D. Ferguson and A. Stentz. Anytime RRTs. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2006.

[15] M. Penrose. *Random Geometric Graphs*. Oxford University Press, 2003.

[16] J. Dall and M. Christensen. Random geometric graphs. *Physical Review E*, 66(1):016121, Jul 2002.

[17] S. Karaman and E. Frazzoli. Incremental sampling-based algorithms for optimal motion planning. http://arxiv.org/abs/1005.0416.

[18] S. Karaman and E. Frazzoli. Optimal kinodynamic motion planning using incremental sampling-based methods. In *Proceedings of the IEEE Conference on Decision and Control (CDC)*, 2010. Submitted.

[19] S. Muthukrishnan and G. Pandurangan. The bin-covering technique for thresholding random geometric graph properties. In *Proceedings of the sixteenth annual ACM-SIAM symposium on discrete algorithms*, 2005.

[20] H. Samet. *Design and Analysis of Spatial Data Structures*. Addison-Wesley, 1989.

[21] S. Arya, D. M. Mount, R. Silverman, and A. Y. Wu. An optimal algorithm for approximate nearest neighbor search in fixed dimensions. *Journal of the ACM*, 45(6):891–923, November 1999.

[22] S. Arya and D. M. Mount. Approximate range searching. *Computational Geometry: Theory and Applications*, 17:135–163, 2000.

*Anytime Motion Planning using the RRT**

Anytime Motion Planning using the RRT*

Sertac Karaman Matthew R. Walter Alejandro Perez Emilio Frazzoli Seth Teller

***Abstract*— The Rapidly-exploring Random Tree (RRT) algorithm, based on incremental sampling, efficiently computes motion plans. Although the RRT algorithm quickly produces candidate feasible solutions, it tends to converge to a solution that is far from optimal. Practical applications favor "anytime" algorithms that quickly identify an initial feasible plan, then, given more computation time available during plan execution, improve the plan toward an optimal solution. This paper describes an anytime algorithm based on the RRT* which (like the RRT) finds an initial feasible solution quickly, but (unlike the RRT) almost surely converges to an optimal solution. We present two key extensions to the RRT*, committed trajectories and branch-and-bound tree adaptation, that together enable the algorithm to make more efficient use of computation time online, resulting in an anytime algorithm for real-time implementation. We evaluate the method using a series of Monte Carlo runs in a high-fidelity simulation environment, and compare the operation of the RRT and RRT* methods. We also demonstrate experimental results for an outdoor wheeled robotic vehicle.**

I. Introduction

The motion planning problem is to find a dynamically feasible trajectory that takes the robot from an initial state to a goal state while avoiding collision with obstacles. Motion planning is of fundamental importance not only for robotics [1], but also in many applications outside the robotics domain [1]–[4].

From a computational complexity point of view, even a simple form of the motion planning problem is PSPACE-hard [5], which suggests that any *complete algorithm*, i.e., one that returns a solution if one exists and returns failure otherwise, is doomed to be computationally intractable.

In order to achieve computational efficiency, practical motion planning methods generally relax the completeness requirements. Sampling-based approaches, including algorithms such as the Probabilistic RoadMap (PRM) [6] and the RRT [7], form a relatively recent line of research in this direction. Most sampling-based algorithms are *probabilistically complete*, i.e., the probability that the algorithm finds a solution, if one exists, converges to one as the number of samples approaches infinity.

Sampling-based algorithms have the advantage that they are able to find a feasible motion plan relatively quickly (when a feasible plan exists), even in high-dimensional state spaces. Furthermore, the RRT, in particular, effectively handles systems with differential constraints. These characteristics make the RRT a practical algorithm for motion planning on state-of-the-art robotic platforms [8].

Any robotic motion planning algorithm intended for practical use must operate within limited real-time computational resources and incomplete and imperfect knowledge of the environment. Such settings favor "anytime" algorithms that quickly find some feasible but not necessarily optimal motion plan, then incrementally improve it over time toward optimality. An anytime motion planning algorithm should exhibit two properties: a form completeness guarantees and asymptotic optimality. A system based on anytime planning overlaps two functions in time: *execution* of (some initial portion of) its current plan, and *computation* to replace (any pending portion of) the current plan with an improved plan.

The RRT algorithm exhibits the first property, efficiently finding an initial feasible solution. Until recently, the RRT's ability to improve this solution as the number of samples increases was an open research question. Karaman and Frazzoli [9] proved that the probability of the RRT algorithm converging to an optimal solution is actually zero. In the same paper, they proposed an alternative method, RRT*, a sampling-based algorithm with the *asymptotic optimality* property, i.e., almost-sure convergence to an optimal solution, along with probabilistic completeness guarantees. The RRT* algorithm achieves the asymptotic optimality absent from the RRT without incurring substantial computational overhead.

Hence, RRT* provides substantial benefits, especially for real-time applications. Like the RRT, it quickly finds a feasible motion plan. Moreover, it improves the plan toward the optimal solution in the time remaining before plan execution is complete. This refinement property is advantageous, as most robotic systems take significantly more time to execute trajectories than to plan them. For example, robotic cars [10] spend no more than a few seconds to plan a path before driving toward the goal, which may take several minutes. In such settings, asymptotic optimality is particularly useful, since the available computation time as the robot is moving along its trajectory can be used to improve the quality of the remaining portion of the planned path.

In this paper, we leverage the anytime asymptotic optimality property of the RRT* algorithm to improve the online convergence of the plan during execution. Our experimental results show that these proposed extensions to RRT* substantially improve trajectory quality. We analyze the algorithm, compare its performance to that of RRT in a realistic simulation environment, and demonstrate its effectiveness on a wheeled robotic vehicle [11]–[13].

Sertac Karaman and Emilio Frazzoli are with the Laboratory for Information and Decision Systems, Massachusetts Institute of Technology, Cambridge, MA, USA {sertac, frazzoli}@mit.edu

Matthew R. Walter and Seth Teller are with the Computer Science and Artificial Intelligence Laboratory, Massachusetts Institute of Technology, Cambridge, MA, USA {mwalter, teller}@csail.mit.edu

Alejandro Perez is with the Polytechnic University of Puerto Rico, San Juan, PR, USA aperez@ieee.org

II. The RRT* Algorithm

This section formally states the motion planning problem and describes the RRT* algorithm. Consider a system with dynamics of the following form: $\dot{x}(t) = f(x(t), u(t))$, where $x(t) \in X$ and $u(t) \in U$, where $X \subset \mathbb{R}^d$ and $U \subset \mathbb{R}^m$ denote the state space and the input space, respectively. Let X_{obs} denote the *obstacle region*, and $X_{\text{free}} = X \setminus X_{\text{obs}}$ define the *obstacle-free space*. Finally, let $X_{\text{goal}} \subset X$ denote the *goal region*. The motion planning problem is to find a control input $u : [0, T] \to U$ that yields a *feasible path* $x(t) \in X_{\text{free}}$ for $t \in [0, T]$ from an initial state $x(0) = x_{\text{init}}$ to the goal region $x(T) \in X_{\text{goal}}$ that obeys the system dynamics.

The *optimal* motion planning problem imposes the additional requirement that the resulting feasible path minimize a given cost function, $c(x)$, mapping each non-trivial admissible trajectory $x : [0, T] \to X$ to a positive real number.

In solving the optimal motion planning problem, the RRT* algorithm builds and maintains a tree $\mathcal{T} = (V, E)$ comprised of a vertex set V of states from X_{free} connected by directed edges $E \subseteq V \times V$. The manner in which the RRT* generates this tree closely resembles that of the standard RRT, with the addition of a few key steps that achieve optimality. The RRT* algorithm uses a set of basic procedures, which we describe in the context of kinodynamic motion planning [14].

Sampling: The `Sample` function randomly samples a state $z_{\text{rand}} \in X_{\text{free}}$ from the obstacle-free region of the state space.

Distance: $\texttt{Dist} : X \times X \to \mathbb{R}_{\geq 0}$ returns the cost of the optimal trajectory between two states, assuming no obstacles. Without differential constraints, it is the Euclidean distance.

Nearest Neighbor: Given a state $z \in X$ and the tree $\mathcal{T} = (V, E)$, the $v = \texttt{Nearest}(T, z)$ function returns the nearest node in the tree in terms of the distance function.

Near-by Vertices: Given a state $z \in X$, tree $\mathcal{T} = (V, E)$, and a number n, the $Z_{\text{nearby}} = \texttt{Near}(\mathcal{T}, z, n)$ function returns the vertices in V that are near z. More precisely, define $\texttt{Reach}(z, l) = \{z' \in X \,|\, \texttt{Dist}(z, z') \leq l \text{ or } \texttt{Dist}(z, z') \leq l\}$, and choose $l(n)$ such that $\texttt{Reach}(z, l(n))$ contains a ball of volume $\gamma\,((\log n)/n)^d$, where γ is a fixed number [14].

Collision Check: The $\texttt{ObstacleFree}(x)$ function checks whether a path $x : [0, T] \to X$ lies within the obstacle-free region of state space, i.e., $x(t) \in X_{\text{free}}$ for all $t \in [0, T]$.

Steering: The $(x, u, T) = \texttt{Steer}(z_1, z_2)$ function solves for the control input $u : [0, T]$ that drives the system from $x(0) = z_1$ to $x(T) = z_2$ along the path $x : [0, T] \to X$.

Node Insertion: Given the current tree $\mathcal{T} = (V, E)$, an existing state $z_{\text{current}} \in V$, and a new state z_{new}, the $\texttt{InsertNode}(z_{\text{current}}, z_{\text{new}}, \mathcal{T})$ procedure adds z_{new} to V and creates an edge to z_{current} as its parent, which it adds to E. It assigns a $\texttt{Cost}(z_{\text{new}})$ to z_{new} equal to that of its parent, plus the cost $c(x)$ of the trajectory associated with the new edge.

Using these functions, the RRT* exhibits the general structure outlined in Alg. 1. With the exception of the process of extending an existing node in the tree toward a new node (lines 8–11), the RRT* essentially behaves identically to the RRT. The RRT* starts with an empty tree and adds a single node corresponding to the initial state. It then builds and refines the tree through a set of N iterations (lines 3–11). Like the RRT, the RRT* incrementally builds the tree by sampling a random state z_{rand} from the obstacle-free space (line 4) and solving for a trajectory x_{new} that extends the closest node in the tree z_{nearest} toward the sample (lines 5–6). If this trajectory does not collide with obstacles (line 7), the standard RRT inserts the new node z_{new} into the tree with z_{nearest} as its parent and continues with the next iteration.

It is here that the operation of the RRT* differs. Rather than choosing the nearest node as the parent, the RRT* considers all nodes in a neighborhood of z_{new} (line 8) and evaluates the cost of choosing each as the parent. This process (Alg. 2) evaluates the total cost as the additive combination of the cost associated with reaching the potential parent node and the cost of the trajectory to z_{new}. The node that yields the lowest cost becomes the parent as the new node is added to the tree (Alg. 1, line 10). The `ReWire` procedure described in Alg. 3 then checks each node z_{near} in the vicinity of z_{new} to see whether reaching z_{near} via z_{new} would achieve lower cost than doing so view its current parent (Alg. 3, line 3). When this connection reduces the total cost associated with z_{near}, the algorithm modifies ("rewires") the tree to make z_{new} the parent of z_{near} (line 4). The RRT* then continues with the next iteration.

Algorithm 1: $\mathcal{T} = (V, E) \leftarrow \texttt{RRT}^\star(z_{\text{init}})$

1 $\mathcal{T} \leftarrow \texttt{InitializeTree}()$;
2 $\mathcal{T} \leftarrow \texttt{InsertNode}(\emptyset, z_{\text{init}}, \mathcal{T})$;
3 **for** $i = 1$ *to* $i = N$ **do**
4 $\quad z_{\text{rand}} \leftarrow \texttt{Sample}(i)$;
5 $\quad z_{\text{nearest}} \leftarrow \texttt{Nearest}(\mathcal{T}, z_{\text{rand}})$;
6 $\quad (x_{\text{new}}, u_{\text{new}}, T_{\text{new}}) \leftarrow \texttt{Steer}(z_{\text{nearest}}, z_{\text{rand}})$;
7 $\quad$ **if** $\texttt{ObstacleFree}(x_{new})$ **then**
8 $\quad\quad Z_{\text{near}} \leftarrow \texttt{Near}(\mathcal{T}, z_{\text{new}}, |V|)$;
9 $\quad\quad z_{\text{min}} \leftarrow \texttt{ChooseParent}(Z_{\text{near}}, z_{\text{nearest}}, z_{\text{new}}, x_{\text{new}})$;
10 $\quad\quad \mathcal{T} \leftarrow \texttt{InsertNode}(z_{\text{min}}, z_{\text{new}}, \mathcal{T})$;
11 $\quad\quad \mathcal{T} \leftarrow \texttt{ReWire}(\mathcal{T}, Z_{\text{near}}, z_{\text{min}}, z_{\text{new}})$;
12 **return** $\mathcal{T}$

Algorithm 2: $z_{\text{min}} \leftarrow \texttt{ChooseParent}(Z_{\text{near}}, z_{\text{nearest}}, x_{\text{new}})$

1 $z_{\text{min}} \leftarrow z_{\text{nearest}}$;
2 $c_{\text{min}} \leftarrow \texttt{Cost}(z_{\text{nearest}}) + c(x_{\text{new}})$;
3 **for** $z_{near} \in Z_{near}$ **do**
4 $\quad (x', u', T') \leftarrow \texttt{Steer}(z_{\text{near}}, z_{\text{new}})$;
5 $\quad$ **if** $\texttt{ObstacleFree}(x')$ *and* $x'(T') = z_{new}$ **then**
6 $\quad\quad c' = \texttt{Cost}(z_{\text{near}}) + c(x')$;
7 $\quad\quad$ **if** $c' < \texttt{Cost}(z_{new})$ *and* $c' < c_{min}$ **then**
8 $\quad\quad\quad z_{\text{min}} \leftarrow z_{\text{near}}$;
9 $\quad\quad\quad c_{\text{min}} \leftarrow c'$;
10 **return** z_{min}

Algorithm 3: $\mathcal{T} \leftarrow \texttt{ReWire}(\mathcal{T}, Z_{\text{near}}, z_{\text{min}}, z_{\text{new}})$

1 **for** $z_{near} \in Z_{near} \setminus \{z_{min}\}$ **do**
2 $\quad (x', u', T') \leftarrow \texttt{Steer}(z_{\text{new}}, z_{\text{near}})$;
3 $\quad$ **if** $\texttt{ObstacleFree}(x')$ *and* $x'(T') = z_{near}$ *and* $\texttt{Cost}(z_{new}) + c(x') < \texttt{Cost}(z_{near})$ **then**
4 $\quad\quad \mathcal{T} \leftarrow \texttt{ReConnect}(z_{\text{new}}, z_{\text{near}}, \mathcal{T})$;
5 **return** $\mathcal{T}$

III. Extensions for Anytime Motion Planning

This section describes how to exploit the anytime nature of the RRT* algorithm to achieve an online motion planning algorithm that significantly improves path quality during path execution, i.e. as the robot is moving toward its goal. These extensions are inspired by techniques for real-time kinodynamic planning [8].

A. Committed Trajectory

Upon receiving the goal region, the online planning algorithm starts an *initial planning phase*, in which the RRT* runs until the robot must start moving toward its goal. The amount of time devoted to this initial phase is domain-dependent. In the example presented in this paper involving a full-size robotic forklift, this time is on the order of a few seconds, which is the time required to put the vehicle in gear.

Once the initial planning phase is completed, the online algorithm goes into an *iterative planning phase*, in which the robot starts to execute the initial portion of the best trajectory in the tree maintained by the RRT* algorithm. Meanwhile, the RRT* algorithm focuses on improving the remaining part of the trajectory. Once the robot reaches the end of the portion that it is executing, the iterative phase is restarted by picking the current best path in the tree and executing its initial portion.

More precisely, the iterative planning phase occurs as follows. Given a motion plan $x : [0, T] \rightarrow X_{\text{free}}$ generated by the RRT* algorithm, the robot starts to execute an initial portion of $x : [0, t_{\text{com}}]$ until a given commit time t_{com}. We refer to this initial path as the *committed trajectory*. Once the robot starts executing the committed trajectory, the RRT* algorithm deletes each of its branches and declares the end of the committed trajectory $x(t_{\text{com}})$ to be the new tree root. This effectively shields the committed trajectory from any further modification. As the robot proceeds along the committed trajectory, the RRT* algorithm continues to improve the motion plan within the new (i.e., uncommitted) tree of trajectories. Once the robot reaches the end of the committed trajectory, the procedure restarts, using the initial portion of what is currently the best path in the RRT* tree to define a new committed trajectory. The iterative phase repeats until the robot reaches the goal region.

B. Branch-and-Bound

In addition to considering a committed trajectory, we also employ a branch-and-bound technique to more efficiently build the tree. Branch-and-bound is used within many domains in optimization and artificial intelligence. Most notably, the approach we present in this section shares certain aspects with the A* graph search algorithm and its variants, which are widely used in robotics applications [15].

1) Cost-to-go functions: Before providing the details of the branch-and-bound algorithm, let us first define a cost-to-go function as follows. For an arbitrary state $z \in X_{\text{free}}$, let c_z^* be the cost of the optimal path that starts at z and reaches the goal region, X_{goal}. A *cost-to-go function* $\texttt{CostToGo}(z)$ associates each $z \in X_{\text{free}}$ with a real number between 0 and c_z^*. Essentially, $\texttt{CostToGo}(z)$ provides a *lower-bound* on the optimal cost to reach the goal from z. The cost-to-go function described here is equivalent to the admissible heuristic employed by A* planning algorithms.

There are many ways to define a cost-to-go function, the most trivial being $\texttt{CostToGo}(z) = 0$ for all $z \in X_{\text{free}}$. Note that as the cost function more closely approximates the optimal cost-to-go c_z^*, the branch-and-bound algorithm becomes more effective.

In this paper, we use the Euclidean distance between z and X_{goal} (neglecting obstacles) divided by the maximum speed of the vehicle as a cost-to-go function.

2) Branch-and-bound algorithm: In the context of the RRT and RRT*, the branch-and-bound algorithm works as follows. Let $\mathcal{T} = (V, E)$ be a tree and $z \in V$ be a vertex in $\mathcal{T}$. Recall that $\texttt{Cost}(z)$ denotes the cost of the unique path that starts from the root node and reaches z through the edges of $\mathcal{T}$. Let $z_{\min}$ be the node that lies in the goal region and has the lowest-cost trajectory that reaches X_{goal} along the edges of $\mathcal{T}$. The cost of the unique trajectory that starts from the root and reaches $z_{\min}$ gives an upper bound on cost. Let V' denote the set of nodes z for which the cost to get to z, plus the lower-bound on the optimal cost-to-go, is more than the upper-bound c_u, i.e., $V' = \{z \in V \mid \texttt{Cost}(z) + \texttt{CostToGo}(z) \geq \texttt{Cost}(z_{\min})\}$. The branch-and-bound algorithm keeps track of all such nodes and periodically deletes them from the tree.

IV. System Dynamics and The Control Procedure

This section, outlines the aforementioned steering function and trajectory controller employed by the RRT*.

A. Dubins Curve Steering Function

The RRT* algorithm uses a steering function that assumes a Dubins vehicle model [16] to generate dynamically-feasible trajectories for curvature-constrained vehicles. Dubins vehicle dynamics have the general form:

$$\begin{aligned} \dot{x}_D &= v_D \cos(\theta_D) \\ \dot{y}_D &= v_D \sin(\theta_D) \\ \dot{\theta}_D &= u_D, \qquad |u_D| \leq \frac{v_D}{\rho}, \end{aligned}$$

where (x_D, y_D) and θ_D specify the position and orientation, u_D is the steering input, v_D is the velocity, and ρ is the minimum turning radius.

There are six types of paths that characterize the optimal trajectory between two states for a Dubins vehicle, each specified by a sequence of left, straight, or right steering inputs [16]. In this paper, we consider four path classes and choose the steering between two states that minimizes cost. Karaman and Frazzoli [14] describe the steering function in more detail.

B. Trajectory Tracking

The steering function returns a trajectory parametrized by a sequence of reference states (x_R, y_R, θ_R) and a reference velocity v_R. We employ a straightforward steering controller [13] to track this reference trajectory.

Let z_{n} be the robot's current state and $z_{\text{n+1}}$ be the next reference point. Define the cross-track error e_{ct} be the distance between z_n and z_{n+1} along a line perpendicular to the desired orientation θ_{n+1}. We steer the vehicle along the trajectory by controlling the steering angle δ via

$$\delta = K_{\text{str}} \arctan(K_{\text{ct}}\, e_{\text{ct}}) + K_{\text{str}}\, e_\theta,$$

where K_{str} and K_{ct} are gains. Meanwhile, we employ a PI controller to track the reference speed v_R,

$$u = K_{\text{p}}(v_R - v) + K_{\text{i}} \int_0^t (v_R - v(\tau))\, d\tau.$$

Using these controllers, the robot tracks the trajectory defined by the sequence of reference points.

V. Results

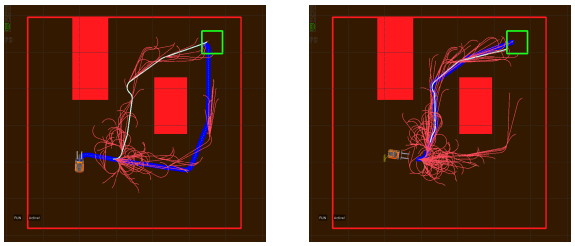
We implemented our algorithm in simulation as well as on an outdoor ground vehicle. In this section we discuss the performance of the RRT* in both domains and compare the results against those of a standard RRT. The simulations demonstrate the algorithm's ability to exploit computation available during the execution of the committed trajectory to improve the solution. In contrast, while RRT may improve the trajectory by chance through constant re-planning, such improvements are unlikely (probability zero convergence).

A. Performance Analysis

We first evaluate the implications of execution-time re-planning for the RRT* using a high-fidelity vehicle simulator. The vehicle dynamics correspond to those of a rear wheel-steered nonholonomic ground vehicle. Shown in Fig. 1, the environment consists of a bounded region with two polygonal obstacles. The planner must find a feasible trajectory from an initial pose in the lower left of the environment to the goal region indicated by the green box. We performed a total of 166 Monte Carlo simulation runs with the RRT* motion planner and 191 independent runs with the standard RRT. Both planners use branch-and-bound for tree expansion and maintain a committed trajectory. Both the RRT and RRT* were allowed to explore the state space throughout the execution period.

Figure 1 depicts the result of two independent runs of the RRT* in the simulation environment. In the first, the RRT* initially finds a trajectory that takes the vehicle along a relatively high cost path to the right of the obstacle (Fig. 1(a), in blue). As the vehicle begins to execute the plan, however, tree rewiring reveals a shorter, lower-cost route between the obstacles (Fig. 1(b)). Meanwhile, the second run demonstrates the benefit of branch-and-bound and online refinement as the algorithm improves the current path (Fig. 1(c)) into a more direct path to the goal (Fig. 1(d)).

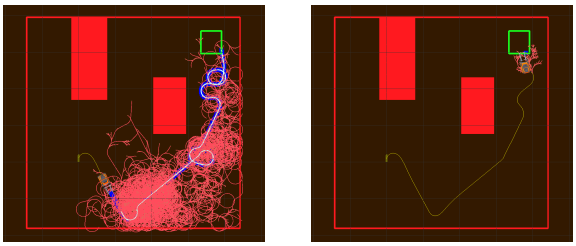
We compare the paths executed by the RRT* with those that result from a standard RRT-based planner. Figure 2 shows two different runs of the RRT at different points of execution. The re-planning together with branch-and-bound enable the RRT to refine an existing solution as demonstrated by the removal of unnecessary loops in the path. In contrast to the RRT* algorithm, however, these improvements tend to be local in nature and do not provide the significant modifications to the structure of the tree necessary to achieve lower cost solutions. Consequently, the free space bias of the RRT limits the extent to which the planner is able to refine



(a) RRT* run 1 (b) RRT* run 1

(c) RRT* run 2 (d) RRT* run 2

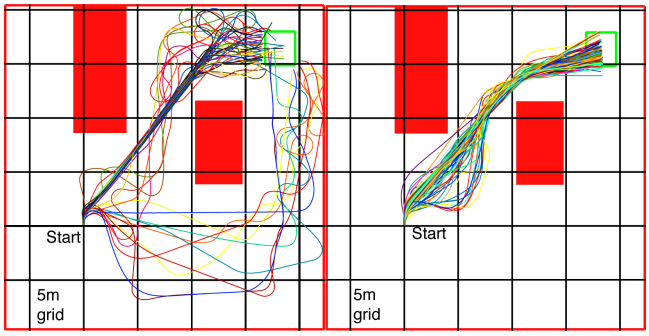
Fig. 1. The RRT* tree at two points during the execution of two different simulation runs. In the first run, (a) the planner initially finds the longer path to the right of the obstacle but, as a result of the online refinement, (b) the RRT* correctly chooses the lower cost path between the obstacles. The results of the second run demonstrate typical behavior of the RRT*, which refines (c) an initial path into (d) a more direct path to the goal.



(a) RRT run 1 (b) RRT run 1

(c) RRT run 2 (d) RRT run 2

Fig. 2. Two simulation runs with the RRT motion planner. (a,b) The first run demonstrates a common failure of the RRT, which effectively gets stuck after constructing a tree biased toward the longer route to the goal. While the RRT does refine the path (b), it converges to a high-cost solution. (c) During the initial period of the second run, the RRT identifies a feasible path to the goal that includes a loop maneuver. The planner continues to search for an improved trajectory and, with the assistance of branch-and-bound, (d) discovers a shorter loop-free path that the vehicle then executes.

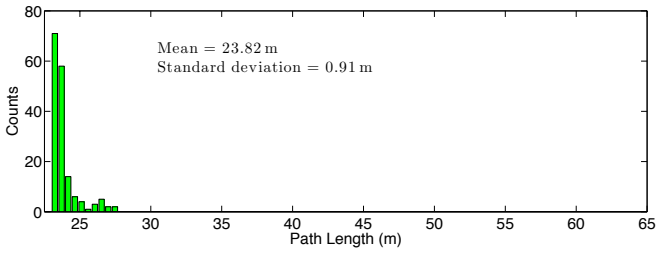


(a) RRT (b) RRT*

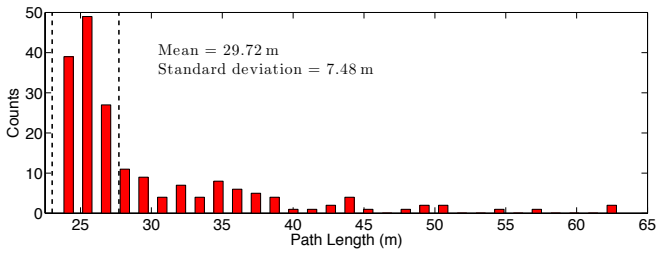
Fig. 3. Vehicle paths traversed for (a) 65 simulations of the RRT and (a) 140 simulations with our RRT* planner.

paths. This effect is evident in the result of the first run as the RRT gets "stuck" with a tree that favors longer paths to the right of the obstacle (Fig. 2(a)) and converges to a sub-optimal path (Fig. 2(b)). As is evident in Fig. 3(a), the RRT frequently produces trajectories that are unnecessarily long due either to the selection of over-long routes, or to oscillations in otherwise direct paths.

Figure 3(b) depicts the final paths for the RRT* simulations. In each case, the algorithm correctly identifies the route between the two obstacles as providing shorter paths to the goal. Occasionally, the RRT* yields an initial solution that steers the vehicle away from the goal. As the vehicle executes the path, the RRT* rewires the structure of the tree to discover a more direct path. This refinement continues while the vehicle executes the committed portion of the trajectory. The result is loop-free paths that tend to be more direct than those of the RRT.



(a) RRT*



(b) RRT

Fig. 4. Histogram plots of the executed path length for simulations of (a) the RRT* and (b) the RRT. The vertical dashed lines in (b) depict the range of path lengths that result from the RRT* planner.

The online formulation of the RRT* algorithm exploits the execution period to modify the tree structure as it converges to the optimal path. This convergence is evident in the distribution over the length of the executed simulation trajectories (Fig. 4(a)) that exhibits a mean length (cost) of 23.82 m and a standard deviation of 0.91 m for the set of 166 simulations. For comparison, Fig. 4(b) presents the corresponding distribution for the RRT planner. The mean path length for the 191 RRT simulations is 29.72 m while the standard deviation is 7.48 m. The significantly larger variance results from the RRT getting "stuck" refining a tree with sub-optimal structure. The anytime RRT*, on the other hand, opportunistically takes advantage of the available execution time to converge to a near-optimal path.

B. Motion Planning for a Robotic Forklift

In addition to the simulation experiments, we demonstrate the performance of the RRT* on a robotic ground vehicle. The platform (Fig. 5) is a rear wheel-steered robotic forklift designed to operate on uneven terrain alongside and in collaboration with humans [11].

We conducted a series of tests with both the RRT* anytime algorithm as well as the RRT-based planner. The vehicle operated in a 20 m by 20 m packed gravel environment consisting of five obstacles (Fig. 6). The task was to navigate from a starting position in one corner to a 1.6 m goal region in the opposite corner while avoiding the obstacles. We manually specified the location of the obstacles. In each experiment, planning started immediately prior to the controller tracking the committed trajectory.

Figure 6 presents the result of four different tests with the RRT* anytime motion planner. The plots depict the best trajectory as maintained by the RRT* at different points during the plan execution (false-colored by time). In the scenario represented in the upper left, the RRT* initially identifies a sub-optimal path that goes around an obstacle but, as the vehicle begins to execute the path, the planner correctly refines the solution to a shorter trajectory. As the vehicle proceeds along the committed trajectory, the planner continues to rewire the tree as evident in the improvements near the end of the execution when the paths more directly approach the goal.

Fig. 5. The robotic forklift used for experimental validation.

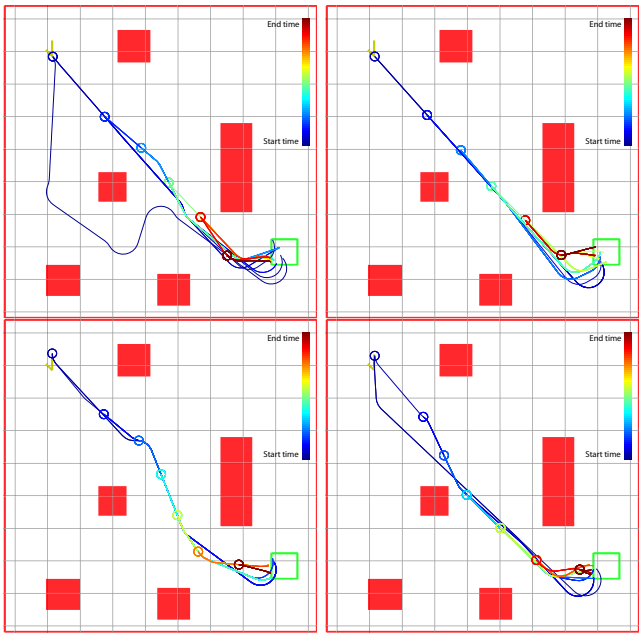


Fig. 6. Four runs of the anytime RRT* on the robotic forklift. Starting in the upper left, the forklift was tasked with driving to the goal region while avoiding obstacles. The trajectories indicate the optimal path as estimated by the RRT* at different points in time during the execution and are false-colored by time. Circles denote the initial position for each path.

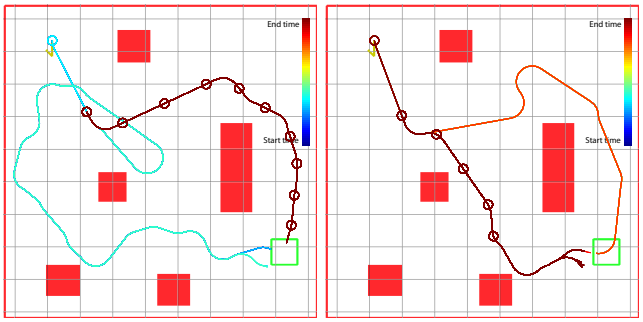


Fig. 7. Plans generated by the anytime planner using the standard RRT.

For comparison, Fig. 7 presents the resulting paths for the anytime planner utilizing the standard RRT. In the scenario depicted on the left, the RRT initially finds a looping trajectory that goes wide to the left but, after moving a few meters, discovers a shorter path that takes the vehicle wide to the right. At this point, the structure of the tree biases the RRT toward refinements that improve the trajectory only locally. In the second test, the RRT revises the initial trajectory that unnecessarily goes to the right of the obstacle and discovers a shorter, yet sub-optimal path to the goal.

VI. Conclusion

Incremental sampling-based motion planners have been used successfully to plan trajectories for vehicles with restricted dynamics operating in the presence of obstacles. The appeal of incremental planners such as the RRT stems, in part, from their efficiency at identifying feasible motion plans and their intuitive implementation. However, the feasible solutions produced by the RRT tend to be far from optimal.

This paper described an anytime motion planning algorithm that uses the RRT* to solve for and improve solutions to the motion planning problem in an online fashion. We described methods that enable the planner to asymptotically converge to the optimal solution online, during trajectory execution. We used Monte Carlo simulation to evaluate convergence of the anytime RRT* algorithm, and compared it to a standard RRT-based motion planner. We further demonstrated the algorithm's performance while planning trajectories for a large ground vehicle.

Acknowledgments

We gratefully acknowledge the support of the U.S. Army Logistics Innovation Agency, the U.S. Army Combined Arms Support Command, and the Department of the Air Force (Air Force Contract FA8721-05-C-0002).

References

[1] J. Latombe, "Motion planning: A journey of robots, molecules, digital actors, and other artifacts," *Int'l J. of Robotics Research*, vol. 18, no. 11, pp. 1119–1128, 1999.

[2] A. Bhatia and E. Frazzoli, "Incremental search methods for reachability analysis of continuous and hybrid systems," in *Hybrid Systems: Computation and Control*, ser. Lecture Notes in Computer Science, R. Alur and G. Pappas, Eds., Mar. 2004, no. 2993, pp. 451–471.

[3] M. S. Branicky, M. M. Curtis, J. Levine, and S. Morgan, "Sampling-based planning, control, and verification of hybrid systems," *IEEE Proc. Control Theory and Applications*, vol. 153, no. 5, pp. 575–590, Sept. 2006.

[4] Y. Liu and N. Badler, "Real-time reach planning for animated characters using hardware acceleration," in *IEEE Int'l Conf. on Computer Animation and Social Characters*, 2003, pp. 86–93.

[5] J. Reif, "Complexity of the mover's problem and generalizations," in *Proc. IEEE Symp. on Foundations of Computer Science*, 1979.

[6] L. Kavraki, P. Svestka, J. Latombe, and M. Overmars, "Probabilistic roadmaps for path planning in high-dimensional configuration spaces," *IEEE Trans. on Robotics and Automation*, vol. 12, no. 4, pp. 566–580, 1996.

[7] S. M. LaValle and J. J. Kuffner, "Randomized kinodynamic planning," *Int.'l J. of Robotics Research*, vol. 20, no. 5, pp. 378–400, May 2001.

[8] Y. Kuwata, J. Teo, G. Fiore, S. Karaman, E. Frazzoli, and J. How, "Real-time motion planning with applications to autonomous urban driving," *IEEE Trans. on Control Systems*, vol. 17, no. 5, pp. 1105–1118, 2009.

[9] S. Karaman and E. Frazzoli, "Incremental sampling-based algorithms for optimal motion planning," in *Proc. Robotics: Science and Systems (RSS)*, 2010.

[10] S. Thrun *et al.*, "Stanley: The robot that won the DARPA Grand Challenge," *J. of Field Robotics*, vol. 23, no. 9, pp. 661–692, Sept. 2006.

[11] S. Teller *et al.*, "A voice-commandable robotic forklift working alongside humans in minimally-prepared outdoor environments," in *Proc. IEEE Int'l Conf. on Robotics and Automation (ICRA)*, May 2010.

[12] A. Correa, M. R. Walter, L. Fletcher, J. Glass, S. Teller, and R. Davis, "Multimodal interaction with an autonomous forklift," in *Proc. ACM/IEEE Int'l Conf. on Human-Robot Interaction (HRI)*, Mar. 2010.

[13] M. R. Walter, S. Karaman, E. Frazzoli, and S. Teller, "Closed-loop pallet engagement in an unstructured environment," in *Proc. IEEE/RSJ Int'l Conf. on Intelligent Robots and Systems (IROS)*, Oct. 2010.

[14] S. Karaman and E. Frazzoli, "Optimal kinodynamic motion planning using incremental sampling-based methods," in *Proc. IEEE Conf. on Decision and Control (CDC)*, Dec. 2010.

[15] M. Likhachev, D. Ferguson, G. Gordon, A. Stentz, and S. Thrun, "Anytime search in dynamic graphs," *J. Artificial Intelligence*, vol. 172, pp. 1613–1643, Sept. 2008.

[16] L. Dubins, "On the curves of minimal length on average curvature, and with prescribed initial and terminal positions and tangents," *American J. of Mathematics*, vol. 79, no. 3, pp. 497–516, 1957.