Wide-Area Egomotion from Omnidirectional Video and Coarse 3D Structure

by

Olivier Koch

Submitted to the Department of Electrical Engineering and Computer Science
in partial fulfillment of the requirements for the degree of

Master of Science

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

February 2007

Author .....
Department of Electrical Engineering and Computer Science
February 2, 2007

Certified by .....
Seth Teller
Associate Professor of Computer Science and Engineering
Thesis Supervisor

Accepted by .....
Arthur C. Smith
Chairman, Department Committee on Graduate Students

Wide-Area Egomotion from Omnidirectional Video and Coarse 3D Structure

by

Olivier Koch

Submitted to the Department of Electrical Engineering and Computer Science
on February 2, 2007, in partial fulfillment of the
requirements for the degree of
Master of Science

Abstract

This thesis describes a method for real-time vision-based localization in human-made environments. Given a coarse model of the structure (walls, floors, ceilings, doors and windows) and a video sequence, the system computes the camera pose (translation and rotation) in model coordinates with an accuracy of a few centimeters in translation and a few degrees in rotation. The system has several novel aspects: it performs 6-DOF localization; it handles visually cluttered and dynamic environments; it scales well over regions extending through several buildings; and it runs over several hours without losing lock.

We demonstrate that the localization problem can be split into two distinct problems: an initialization phase and a maintenance phase. In the initialization phase, the system determines the camera pose with no other information than a search region provided by the user (building, floor, area, room). This step is computationally intensive and is run only once, at startup. We present a probabilistic method to address the initialization problem using a RANSAC framework. In the maintenance phase, the system keeps track of the camera pose from frame to frame without any user interaction. This phase is computationally light-weight to allow a high processing frame rate and is coupled with a feedback loop that helps reacquire "lock" when lock has been lost. We demonstrate a simple, robust geometric tracking algorithm based on correspondences between 3D model lines and 2D image edges.

We present navigation results on several real datasets across the MIT campus with cluttered, dynamic environments. The first dataset consists of a five-minute robotic exploration across the Robotics, Vision and Sensor Network Lab. The second dataset consists of a two-minute hand-held, 3D motion in the same lab space. The third dataset consists of a 26-minute exploration across MIT buildings 26 and 36. We also present a detailed analysis of the system performance along with several failure modes and ideas to address them.

Thesis Supervisor: Seth Teller
Title: Associate Professor of Computer Science and Engineering

Acknowledgments

I would like to thank all those who contributed to this work: Matthew Antone for his multiple comments and suggestions, Yoni Battat for his 3D models, Bryt Bradley for her tips and tricks, David Diel for his ideas at the early stage of the project, Albert Huang for his skills on the Ladybug, and Ed Olson for his robot. In general, I would like to thank all the members of the Robotics, Vision and Sensor Networks Lab for making the lab such a great workplace.

Also, I wish to express my most sincere gratitude to my family: Anne-Christine, Arnaud, Aurélie, Florence and my parents, and to my friends, who made my life in Cambridge and Boston so enjoyable: Léonide Saad, Anya Obizhaeva, Gleb Chuvpilo, Tilke Judd, Dan Roy, Cathy Bolliet, Benjamin Bruet, Blaise and Valérie Gassend, and many more.

Above everything else, I would like to thank my advisor Professor Seth Teller for having been such a wonderful advisor. His high availability, very strong support and consistent input of ideas are behind all my achievements in these two years.

I dedicate this work to my parents.

Contents

1 Introduction **19**

1.1 Vision-based Egomotion 19

1.2 Contributions 21

2 Related Work **23**

2.1 The Pose Estimation Problem 24

2.2 The Tracking Problem 26

2.3 Structure from Motion 26

2.4 Omnidirectional Video 28

3 Preliminaries **29**

3.1 Notation 29

3.2 Homogeneous and Non-homogeneous Coordinates 30

3.3 The PointGrey Ladybug Camera 31

3.4 Coordinate Frames 31

3.5 Pixel Rectification 32

3.6 Image Edges 33

3.7 Edge Overlap 35

3.8 Topology on the Image Sphere 36

4 Estimating Camera Pose From Line Correspondences **39**

4.1 Absolute Orientation 39

4.2 Localization by Error Function Minimization 39

4.3 Pose Estimation from Three Line Matches 40
4.4 Degenerate Configurations 43

5 Initialization **45**
5.1 Probabilistic Line-Edge Matching 46
5.2 Rotation Voting 47
5.3 System Initialization 49

6 Maintenance **51**
6.1 Hue-based Edge Matching Constraint 51
6.2 A Random Sample Algorithm 52
6.3 Correspondence Lifetimes 52

7 Offline Visibility Computation for Wide-Area Scaling **55**
7.1 Model Coordinate Subdivision 55
7.2 Visible Model Faces 56
7.3 Visible Model Edges 57

8 Experimental Results **59**
8.1 Pose Estimation from Three Line Matches 60
8.2 Offline Visibility Computation 60
8.3 Initialization 62
8.3.1 Probabilistic Line-Edge Matching 62
8.3.2 Rotation Voting 62
8.3.3 Corner Matching 64
8.3.4 Vanishing Points 65
8.4 Initialization Example 66
8.5 Maintenance 66
8.5.1 Hue-Based Edge Matching Constraint 66
8.5.2 Correspondence Lifetimes 67
8.6 Real sequences 69
8.6.1 The LAB Dataset 69

8.6.2 The HAND-HELD Dataset 70
8.6.3 The CORRIDOR Dataset 71
8.7 An Application to 3D Model Texture Painting 75
8.8 Accuracy Analysis 77
8.9 System Performance 77
8.10 Discussion 77

9 Conclusion and Future Work **83**
9.1 System Characteristics 83
9.1.1 Successes 83
9.1.2 Limitations 84
9.2 Future Directions 84

A A Note on 3D Models **91**
A.1 Model Assumptions 91
A.2 Model Representation 91
A.3 Model Generation 92

B Representing Rotations **93**

C Alternative Initialization Methods **95**
C.1 Initialization from Vanishing Point Alignment 95
C.2 Initialization from Corner Alignment 96

D Ladybug Calibration Parameters **99**
D.1 Intrinsic Parameters 99
D.2 Extrinsic Parameters 100

E Development Instructions **103**
E.1 Location and Organization of Source 103
E.2 Build Instructions 104
E.3 Dataset Import 105
E.4 Execution Instructions 105

F Datasets **109**

F.0.1 Image Datasets 109

F.0.2 3D Models 110

List of Figures

1-1 Egomotion reconstruction from coarse 3D model (top) and omnidirectional video sequence (bottom). The camera motion is shown in blue (1,800 frames - path length $\sim 120\ m$ at the speed of $0.4\ m/s$). 20

3-1 The Ladybug camera unit (right, courtesy of Point Grey Research) and its schematic structure (left). Five sensors are set on a horizontal rig. The sixth sensor points vertically. Note that the sensors do not have identical optical centers; they are approximately 2 cm apart. 31

3-2 The multi-sensor coordinate frame (left) and the generic camera coordinate frame (right). A 3D point M in the world maps onto a point m on the re-projection sphere. 32

3-3 Left: the geometry of an edge ab in the spherical model. The center is noted as o, the two end points as a and b and the normal vector as n. Right: the geometry of two edges a_1b_1 and a_2b_2. α represents the dihedral angle. The poles are noted as p_1 and p_2. 34

3-4 The maximum polar angle between the edge (o, a_1, a_2) and the edge (o, b_1, b_2) is defined as the maximum angle between the poles p_1 and p_2 and the edge end points a_1, a_2, b_1 and b_2. In this case, it is the angle between p_1 and a_2. 35

3-5 Left: the two edges in grey have the same dihedral angle with the edge in green. However, one of them has a significantly higher overlap with the green edge than the other. Right: projection view on the grey edge plane. 36

3-6 Sphere tessellation by icosahedron subdivision. Edges are sorted according to the index of the facet pointed to by the normal vector to the edge (in red). 37

4-1 Angle between image edge e_i and model line l_j seen by camera (R, T). The angle α is equal to the inverse cosine of the inner product between the two normal vectors to the planes generated by the camera center, the 2D edge e_i and the 3D line $l_j, respectively$. 41

4-2 Aligning three image edges (e_1, e_2, e_3) onto three model lines (l_1, l_2, l_3). The first step is solved in closed-form. The second step is solved by minimizing the error function δ over α and β. 41

5-1 Probabilistic line-edge matching based on dihedral angle constraints. When a triplet of lines $\{l_i, l_j, l_k\}$ is compatible with a triplet of edges $\{e_r, e_s, e_t\}$, the scoring table is updated accordingly. 47

6-1 Correspondence update mechanism: the angular constraint and the color constraint determine the most likely image match e_j^{t+1} in frame $t+1$ given a correspondence between the 2D image edge e_j^t and the 3D model line l_i at frame t. 52

6-2 Correspondence state machine 53

6-3 Number of sample rounds needed vs. clutter percentage (b) and number of correspondences ($|\bar{\mathcal{S}}_t|$). 54

7-1 Computation of the set of visible faces from a given camera position using pixel color information. 56

7-2 Computation of visible lines from Z-buffer and feedback buffer comparison. Visible points are shown in green; occluded points are shown in red. 57

8-1 Graph of δ with respect to α and β in the pose estimation problem from three line matches. Local minima of the function correspond to the solutions for α and β. 61

8-2 Off-line computation of the visibility set. The grid is displayed in blue. The lines visible from the node shown in red are displayed in green. Note the relatively small size of the visible set with respect to the size of the model. 61

8-3 An example output of the fitness-based line matching algorithm. A 3D model line is selected on the left. The three best image edge matches for this line are displayed on the right. The system found the correct match (in blue on the left-most tile). 62

8-4 Output of the probabilistic line matching algorithm. The probability of match between each model line and each image edge is displayed as a surface function. A peak in the table corresponds to a high match probability between the corresponding model line and image edge. Note that the data is noisier in the real case than in the synthetic case. . . 63

8-5 Output of the rotation voting algorithm. The rotation angles are aggregated in a histogram. Peaks correspond to the most likely camera rotation angles (red stars). The true rotation angle is marked with a dashed line. The algorithm detected the true camera rotation angle. . 63

8-6 Top: Corner detection. Middle: 3D map. Bottom: Re-projection of the structure after alignment. The correct alignment is obtained after aligning the two model corners shown in green in the model with the two image corners highlighted in green on the image. 64

8-7 Pose initialization from vanishing point (VP) alignment. Top: image VPs are displayed in red; expected VPs computed from the model are displayed in blue. Bottom: the structure is reprojected in green after aligning the image VPs with the model VPs. Note that even though the solution is not exact, the alignment of VPs gives a very good estimate of the camera rotation. 65

8-8 Omnidirectional image and re-projected 3D model (in green). Note the accuracy of 6-DOF localization despite clutter in the scene. 66

8-9 The average and standard deviation of the hue along the edge is used to track edges robustly from frame to frame. The tracked edge is displayed in red. The candidates are shown in yellow. 67

8-10 Correspondence state of two model lines over time (sequence HAND-HELD dataset). Left: a model line which is consistently occluded by a couch. The correspondence spent most of its lifetime in Pending or Unknown state. Right: a model line which is partially occluded from frame 245 to frame 250. 68

8-11 Video sequence for the two correspondence in Figure 8-10. The model edge shown in green is visible at frames 200 and 400 but disappears at frame 300. The base of a partition shown in red is consistently occluded by the couch (detail view). 68

8-12 Recovered 3D motion for the LAB dataset (1,500 frames, 120 m at 0.4 m/s). The camera was attached to a robot and followed a flat 2D motion. (a) Overview of the excursion; (b) and (d) Detail view of the camera motion; (c) Area of high occlusion and large visibility change. 70

8-13 LAB dataset; 3D model re-projected on the video. 71

8-14 Recovered egomotion for the HAND-HELD dataset (1,900 frames, 5 m at 0.04 m/s). The camera followed an arbitrary 3D motion. (a) and (b) Overview of the camera motion; (c) and (d) Detail view of the camera path. 72

8-15 HAND-HELD dataset; 3D model re-projected on the video. 73

8-16 Recovered egomotion for the CORRIDOR dataset (7,900 frames, 300 m at 0.20 m/s). The camera was attached to a robot and followed a flat 2D motion. (a) Overview of the excursion; (b) Camera entering two classrooms; (c) Detail view of a classroom entrance; (d) Camera turning at a corner in building 26. 74

8-17 CORRIDOR dataset; 3D model re-projected on the video. 75

8-18 Face subdivision scheme for the texture painting algorithm. Left: a quadrilateral $S_1S_2S_3S_4$; each edge is divided into two equal segments defining four new quadrilaterals $S_1S_{12}S_0S_{34}$, ... Right: a triangle $T_1T_2T_3$ is divided into three quadrilaterals in a similar fashion. 76

8-19 3D model texture painting for various levels of detail. Note the accuracy of details at level 5. Un-modeled 3D objects and localization inaccuracy generate artifacts in the results. 79

8-20 Localization accuracy with respect to the number of correspondences. The data was simulated using a Gaussian noise model for image edges. Accuracy plateaus at about 40 correspondences. 80

8-21 Left: re-projection error distribution for each dataset; plus signs indicate the presence of outliers in the data. Right: correspondence state distribution for the LAB dataset. The pool of *accepted* correspondences remains steady despite large changes in visibility across the sequence. 80

8-22 The hue constraint helps the system avoid mismatches. However, incorrect correspondences may occur when clutter predominates (bottom). Image edges are color-coded by omnidirectional image tile. Model structure is overlaid in green. 81

A-1 Model data structure. (a) Faces point to vertices; (b) edges point to vertices and adjacent faces; (c) vertices point to adjacent edges. . . . 92

E-1 Screenshots of the software application. (a) after opening a database; (b) after opening the model (the camera motion is shown in blue); (c) in *Sphere* mode after running edge detection; (d) 3D map with grid view enabled. The closest node is shown in red; the visible model edges are shown in green; (e) re-projection of the structure after initialization; (f) example of a correspondence after initialization. 106

List of Tables

4.1 The Three-line Pose Estimation Algorithm for Rotation 42

5.1 The $INIT_{LINE-EDGE}$ Algorithm 47

5.2 The $INIT_{R-VOTING}$ Algorithm 48

5.3 The Initialization Algorithm 49

6.1 The Maintenance Algorithm 54

8.1 Test datasets 59

8.2 Parameter values 60

C.1 The Vanishing Point Algorithm for Initialization 96

C.2 The Corner Matching Algorithm for Initialization 97

D.1 Ladybug 5040012 Intrinsic Calibration Parameters 99

D.2 Ladybug 5040012 Extrinsic Calibration Parameters 101

E.1 Source code directories 104

Chapter 1

Introduction

1.1 Vision-based Egomotion

Robust, wide-area egomotion estimation within general environments is a longstanding goal of computer vision. Existing methods typically handle short-duration, short-excursion sequences within visually uncluttered environments. We wish to develop an ego-motion estimation capability suitable for long-duration, long-excursion use, to recover the precise 6-DOF rigid-body pose of a camera (attached to a user's head, body, or hand-held device) as it is moved within a spatially extended, visually cluttered environment.

Our design target, formulated to support precise augmented-reality style overlay of CAD information onto building walls, is to estimate egomotion with an accuracy of 2 centimeters and 0.2 degrees, over several hours of walking-speed motion within a building containing scores or hundreds of rooms. The method described in this thesis falls short of this design target by a factor of about ten; it achieves accuracies of a few centimeters in translation and a few degrees in rotation in our (unprepared) test environments. Some of this error is due to tracking errors; some of it is due to inaccuracies in the available 3D models and in the camera calibration.

We formulate ego-motion estimation as an online 6-DOF localization task alternating between two operating phases. The *Initialization* phase determines a valid current camera pose estimate when the camera pose is known poorly or not at all,

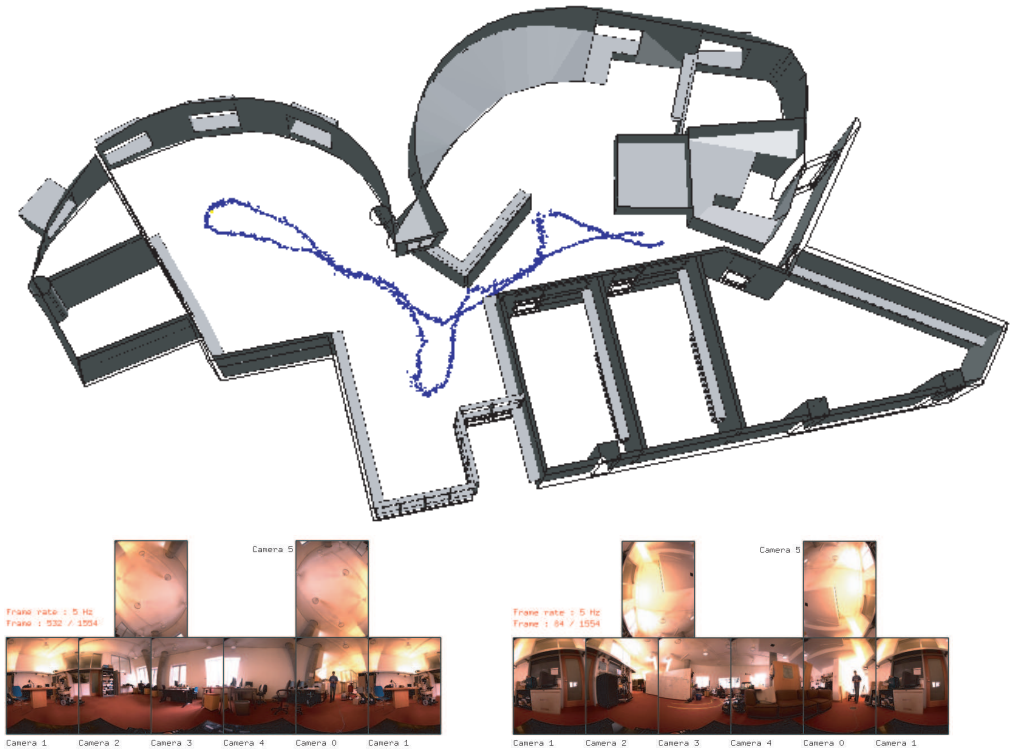

Figure 1-1: Egomotion reconstruction from coarse 3D model (top) and omnidirectional video sequence (bottom). The camera motion is shown in blue (1,800 frames - path length $\sim 120\ m$ at the speed of $0.4\ m/s$).

e.g., at the start of processing, or after loss of lock. After Initialization, i.e., assuming an accurate camera pose estimate for one or more prior frames, the *Maintenance* phase updates the camera pose using the current frame. We show how both the Initialization and Maintenance phases can be dramatically accelerated through *Visibility Analysis* of the environment model performed once before the start of localization.

Our system makes three significant assumptions. First, we assume that a coarse polyhedral model of the environment – which we define as including, at a minimum, walls, floors, ceilings, door jambs and windows frames – is supplied as input. Such a model could be provided by the building's architects, or produced independently by a post-construction modeling method. Second, we assume that the camera is intrinsically calibrated. Third, we assume that the camera motion is smooth, i.e. that the camera never exceeds a sensor-dependant linear or rotational velocity, nor does it move through opaque (i.e. impenetrable) surfaces.

We emphasize that we do *not* make assumptions about built structure that form the foundation of many other vision-based localization systems. For example, we do not assume the presence of vertical and horizontal model edges [11], vanishing points [8], or right angles [4]. We do not assume knowledge of surface color or reflectance attributes in the environment, or indeed of any "appearance" information other than knowledge of the geometric model itself. Finally, though we do assume that the portions of the environment represented by the input model are static, we do not assume a static world. In particular, our method handles time-varying lighting, time-varying visual clutter (caused e.g. by furniture), and transient image motion (caused e.g. by passers-by).

1.2 Contributions

Our method is distinct from other approaches in several respects. First, it uses omnidirectional images in order to support full motion freedom (e.g., close proximity to environment surfaces), and to prevent pointing constraints from burdening the camera operator. Second, the method includes an automated initialization capability. Third, the method is robust to significant clutter and transient motion. Finally, the method scales to large, real-world environments.

Chapter 2

Related Work

There is an extensive literature on the field of vision-based localization. Although theoretical work dates to Ancient Greece, Horn [20] presents the localization problem to the computer science community. In his seminal book, many major computer vision problems are presented and addressed in both a theoretical and a practical way. It is worth noting that some of these problems remain unsolved today. Another reference book by Faugeras et al. [14] brings a significant contribution toward a better understanding of the fundamental aspects of computer vision, with a strong focus on the geometry of one or several images. Finally, Hartley and Zisserman [19] present algorithms for solving many computer vision problems.

Apart from these monographs, a series of original papers have focused on various specific problems. The pose estimation problem consists of computing the camera position and orientation from a set of feature correspondences between an image and the real world. Both closed-form and iterative solutions have been proposed for this problem. In addition, the pose estimation problem may be extended to the tracking problem where the pose of the camera is to be estimated over a sequence of images. The tracking problem therefore incorporates a data association problem since the system needs to update its correspondences. When the surrounding structure is unknown, the problem is to recover both the structure and the camera motion. This is commonly known as the structure from motion problem or, in the robotics community, as "vision-based SLAM". Finally, we review some work specific to omnidirectional

camera systems.

2.1 The Pose Estimation Problem

Several closed-form solutions have been proposed to the pose estimation problem. Horn presents a closed-form solution to the absolute orientation problem in the case of three image-image point correspondences for a calibrated camera [21]. Using the unit quaternion representation for rotations, Horn shows how to compute the rotation by solving a fourth degree polynomial and a linear system of four equations. Dhome [12] presents an algorithm for recovering the orientation of a 3D object given three correspondences between image lines and model lines. The rotation and translation are computed separately. The rotation is determined by solving an eighth degree polynomial. Once the rotation is known, the translation is recovered trivially. The authors also suggest a set of simple rules to reduce the number of solutions.

Several linear solutions are suggested, such as the one of Quan and Lan [32]. Their method is specific to the cases of four points and five points and gives a unique solution to the problem except in degenerate configurations. In addition, their algorithms handle coplanar situations successfully. In a similar fashion, Ansar and Daniilidis [2] present a linear solution to the absolute orientation problem for both point and line correspondences. The algorithm involves computing the SVD decomposition of matrices built from feature coordinates. A noise analysis and experiments on real datasets are presented.

More recently, Nister [29] presented a five-point algorithm for recovering the relative camera motion between two frames. The algorithm expresses the correspondences as a set of linear constraints which can be solved by computing the roots of a tenth-degree polynomial. Robustness is improved by combining the algorithm with random sample consensus [15]. Also, Nister presents an algorithm for upgrading a projective reconstruction into a metric reconstruction in the case of multiple views [30]. A signature function is introduced that allows twisted pairs of camera matrices to be distinguished. Once twisted pairs have been removed, the algorithm proceeds iter-

atively to find the set of camera matrices that best fits a set of hypotheses on the camera intrinsic parameters.

However, a number of iterative methods have also been suggested, since they usually handle noise and over-constrained systems more effectively. Haralick [18] reviews the early work of the German mathematician Grunert (1841) on the absolute orientation problem and presents an analysis of various state-of-the-art algorithms. Experiments show that the accuracy of the results depend greatly on the analytical technique used in the algorithm. Kumar and Hanson [23] propose algorithms to solve for pose from a set of at least five line correspondences. Their approach consists of selecting a set of camera position estimates and minimizing the reprojection error using an iterative method. A technique is presented which handles outliers, i.e. bad correspondences. Similarly, another approach is presented by Phong et al. [31]. They demonstrate an algorithm for estimating the camera position from point or line correspondences. They reduce the problem to a linear least-square minimization problem in the case of points and to a non-linear least-square problem in the case of lines. A new minimization method called "trust-region optimization" is presented which allows efficient solution of the line problem.

Iterative methods are particularly useful once a good estimate of the camera pose is available. Coorg and Teller [11], for example, demonstrate two algorithms for refining camera poses from a large set of images. In the first algorithm, an "incidence counting" mechanism allows extraction of consistent matches between images. In the second algorithm, an iterative intersection-resection method for refining both 3D points and camera poses simultaneously is presented. In a similar manner, Chou [10] presents a framework for the accurate reconstruction of large scale environments from a set of calibrated images. Correspondence hypotheses are generated and scored based on the agreement between observations. Reconstructed surfaces are then taken into account to incorporate occlusion and remove inconsistent correspondences.

2.2 The Tracking Problem

In the tracking problem, the challenge becomes one of identifying and using correspondences over a sequence, rather than a pair, of images. Drummond and Cipolla [13] demonstrated a real-time algorithm for tracking an object given its 3D CAD model. The method involves an edge tracker and a binary-search tree renderer to determine which lines of the object are expected to be visible. An application to online camera calibration is presented and tested on a real data set. Vacchetti [40] pushes the problem one step further by using the concept of keyframes. His algorithm combines short-term point feature matching between image pairs and long-term registration using keyframes. The system requires user input to initialize the system with one or several keyframes and a 3D model of the object to track. More recently, Rosten and Drummond [33] presented a novel approach based on combining point features and line features. Their algorithm enables robust camera tracking from a video sequence and a 3D model using the complementary failure modes of points and lines. In a first step, an estimate of the camera position is computed using both modes. Then, the most probable mode is used as an initial guess, with further refinement based on an edge tracker. Finally, correspondences are updated to prepare for the next frame. In practice, the algorithm performs as fast as 500 Hz thanks to an optimized feature tracker, and handles severe camera shaking.

2.3 Structure from Motion

The "structure from motion" problem consists of recovering both the camera motion and the 3D structure through which the motion occurs. Taylor and Kriegman [39] present an algorithm for reconstructing the 3D structure of a scene from an image sequence. The method is based on minimizing an objective function that measures the reprojection error of the structure lines onto the camera plane. A set of random initial estimates are provided to the system in order to achieve global minimization. Beardsley et al. [7] present a structure from motion algorithm based on corner tracking.

The output of the system is a projective reconstruction of the scene with little skew from the true metric reconstruction with no prior camera calibration. An application to robot navigation in the affine space is demonstrated.

More complex features may also be used for tracking. Se et al. [35], for example, present a method for ego-motion estimation based on SIFT [24] feature tracking on a trinocular system. At each frame, features are matched between each of the three cameras and reconstructed in 3D. The camera pose is then computed by minimizing the reprojection error of these features onto each camera image plane. Features are tracked from frame to frame using odometry to optimize the search space on the image.

Bartoli and Sturm [5] use Plücker coordinates to formulate an efficient least square optimization method. This parametrization is then used in the bundle adjustment problem to remove superfluous degrees of freedom in the system, allowing the use of unconstrained optimization algorithms. Successful reconstruction is demonstrated on several real data sets.

In order to account for noise, clutter and occlusions, Nister [28] presents a "preemptive" approach to RANSAC [15], dramatically increasing the efficiency of random sample consensus. Synthetic and real data results are demonstrated showing ego-motion estimation in real time. He then incorporates the five-point algorithm [29] into his system to obtain "visual odometry" [27].

Zhao et al. [43] present a framework for automatic registration of video with respect to 3D model data. The pipeline is composed of three components: first, a camera pose estimation algorithm that determines the relative motion of the camera from feature tracking; second, a motion stereo algorithm that registers pairs of frames and reconstructs depth of scene points; and third, a 3D-to-3D registration algorithm that aligns the reconstructed point features with 3D data collected from a LIDAR.

Hsu et al. [22] present a "correspondence-less" approach which aligns a video sequence onto an untextured 3D model of the scene by minimizing an energy function. Experimental results are demonstrated on a long aerial sequence, along with an application to augmented reality. However, their method does not incorporate inter-frame

constraints and does not solve the initialization problem.

2.4 Omnidirectional Video

Recently, omnidirectional video, or "omnivideo" systems have become more popular for their natural advantages in terms of reduced occlusion and wide field of view. Yagi et al. [42] for example, present an algorithm for generating a map and navigating using a conic omnivideo system. Vertical edges of the environment are used to track the robot motion and compute the free space into which the robot can move.

In addition, Gluckman [16] maps the visual field of an omnivideo system onto a sphere and presents a generalization of several traditional ego-motion estimation algorithms to the spherical case. His work allows recovery of camera motion from omnidirectional optical flow.

Finally, Bosse et al. [9] demonstrate a method based on vanishing points and 3D lines to solve for structure from motion. The camera motion is modelled using an Extended Kalman Filter and dynamic programming is used to optimize the tracking of 3D lines. The system is demonstrated on both indoor and outdoor sequences.

Chapter 3

Preliminaries

Optical systems are traditionally classified into two categories: dioptric and catadioptric. Dioptric systems involve optical refraction through a single camera lens while catadioptric systems involve reflection in a mirror prior to refraction through a lens. Each of these systems has advantages and drawbacks depending on the application. Dioptric systems are usually preferred for their somewhat compact viewpoint locus but do not achieve the same resolution as catadioptric systems. Recently, a new type of omnivision system involving multiple rigidly attached cameras has appeared providing both compactness and high-resolution images.

Our system has been developed on the Point Grey Research Ladybug camera rig. However, our camera model is generic and a wide range of multi-camera rigs could be used instead of the Ladybug.

3.1 Notation

We denote 3D points and lines as upper case italics (*e.g.* X), 2D points and edges as lower case italics (*e.g.* x), and matrices in bold (*e.g.* $\mathbf{K}$). We denote indices with subscripts (*e.g.* $X = (X_1, X_2, X_3)^\top$). We represent each projective camera as a 3×4 matrix $\mathbf{P}$ as in [19]:

$$\mathbf{P} = \mathbf{K} \left[\begin{array}{cc} \mathbf{R} & -\mathbf{R}T \end{array} \right] \tag{3.1}$$

where $\mathbf{R}$ is a 3×3 rotation matrix, T a 3×1 translation vector, and $\mathbf{K}$ the 3×3 camera calibration matrix

$$\mathbf{K} = \begin{bmatrix} \alpha_x & 0 & x_0 \\ 0 & \alpha_y & y_0 \\ 0 & 0 & 1 \end{bmatrix} \tag{3.2}$$

with α_x and α_y the focal length in pixels, x_0 and y_0 the principal point in pixels, and a skew factor of zero.

3.2 Homogeneous and Non-homogeneous Coordinates

In *homogeneous coordinates*, a 3D point X is represented by four coordinates $X = (X_1, X_2, X_3, 1)^\top$. Scaling is unimportant, so the point $(X_1, X_2, X_3, 1)^\top$ is the same as $(\alpha X_1, \alpha X_2, \alpha X_3, \alpha)^\top$ for any nonzero α. Similarly, a 2D point x is represented by three coordinates $x = (x_1, x_2, 1)$.

In homogeneous coordinates, any affine transformation can be represented by a matrix. In 3D, the matrix is of size 4x4.

With this model, any 3D point X expressed in the homogeneous coordinates can be projected onto a camera $\mathbf{P}$ using the formula $x = \mathbf{P}X$, where x is expressed in the homogeneous coordinates as well ($x = (x_1, x_2, 1)^\top$). If an inverse projection matrix $\mathbf{P}^{-1}$ is known, the image point can be back-projected into a 3D ray $X = \mathbf{P}^{-1}x$. Note that an inverse projection matrix is a 4x3 matrix satisfying the relation $\mathbf{P} \cdot \mathbf{P}^{-1} = \mathbf{I}_3$ where $\mathbf{I}_3$ is the 3×3 identity matrix.

Equation 3.7 shows the sequence of operations for projecting a 3D point X expressed in the non-homogeneous coordinate frame on camera in order to obtain a 2D point x expressed in non-homogeneous coordinates.

$$X = \quad (X_1, X_2, X_3) \quad (Cartesian) \tag{3.3}$$

$$X = \quad (X_1, X_2, X_3, 1) \quad (homogeneous) \tag{3.4}$$

$$x = \mathbf{P}X \tag{3.5}$$

$$x = (x_1, x_2, x_3) \quad (homogeneous) \tag{3.6}$$

$$x = (x_1/x_3, x_2/x_3) \quad (Cartesian) \tag{3.7}$$

Note that in equation 3.7, $x_3 = 0$ never occurs if the 3D point X lies in front of the camera ($x_3 > 0$).

3.3 The PointGrey Ladybug Camera

The Ladybug camera (Figure 3-1) is composed of six color Charge-Coupled Device (CCD) sensors each collecting 1024×768 images at a maximum frame rate of 30 Hz. Each lens has a focal length of approximately 2.5 mm. The sensors are mounted on a rigid and compact head unit, with an effective field of view of more than 75 % of the full sphere.

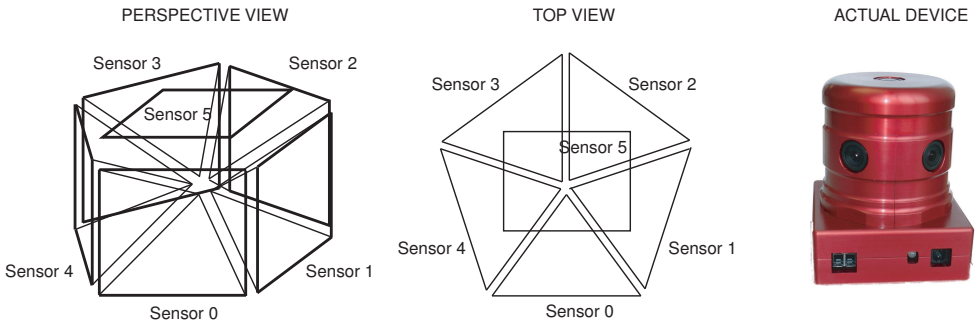

Figure 3-1: The Ladybug camera unit (right, courtesy of Point Grey Research) and its schematic structure (left). Five sensors are set on a horizontal rig. The sixth sensor points vertically. Note that the sensors do not have identical optical centers; they are approximately 2 cm apart.

3.4 Coordinate Frames

Each of the six sensors is numbered from zero to five. Although each sensor has its own coordinate frame (*sensor coordinate frame*), the rigid-body transformation

between each sensor is precisely known, which allows us to conceive a virtual *camera coordinate frame* centered on the common viewpoint locus of all sensors. The *camera coordinate frame* is defined by its Z-axis pointing toward the vertical direction, its X-axis pointing toward the Z-axis of sensor 0 and its Y-axis determined using the right-hand rule.

Figure 3-2 illustrate the *camera coordinate frame*. In the spherical re-projection model, points in the 3D world are projected onto a unit projection sphere representing the camera coordinate frame.

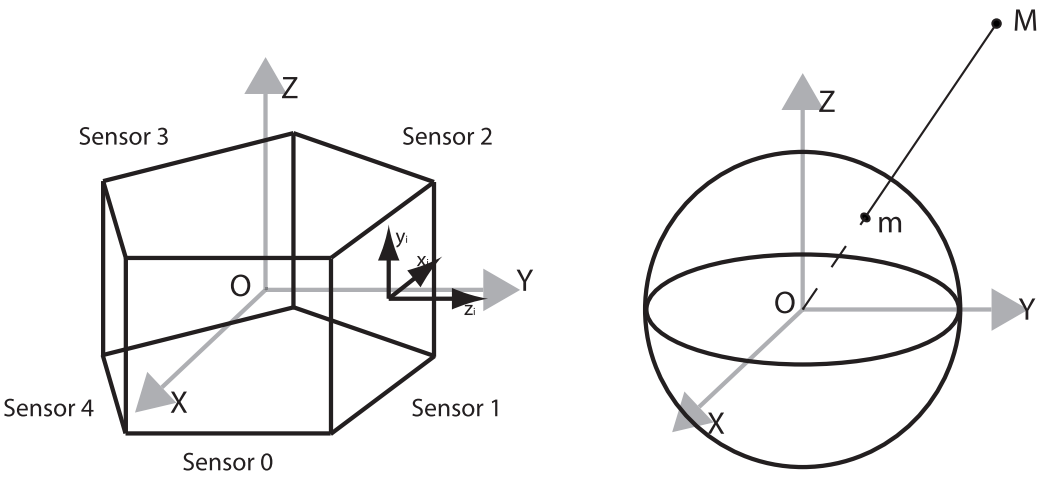

Figure 3-2: The multi-sensor coordinate frame (left) and the generic camera coordinate frame (right). A 3D point M in the world maps onto a point m on the re-projection sphere.

3.5 Pixel Rectification

Since each Ladybug sensor has a wide field of view, the resulting images are highly distorted. Accounting for this distortion is required in order to perform further geometric computation.

There exists an extensive literature on the topic of image un-distortion [38, 26]. The most common approach is based on a polynomial approximation of the distortion transformation. However, polynomial approximations can perform poorly on the regions of the image where the distortion is extreme (namely, the corners). Therefore,

straight lines in the 3D world are not transformed into straight lines on the image in these regions.

We use a single-parameter spherical distortion model proposed by Antone [3]. In this model, the relationship between a distorted image point p_d and the corresponding rectified image point p_r expressed in the sensor normalized coordinate frame is defined as:

$$p_r = \frac{\alpha}{\sqrt{1 - \alpha^2 r_d^2}} \cdot (p_d - c) \tag{3.8}$$

where c is the center of projection of the sensor, r_d is the distance from p_d to c and α is a parameter recovered through calibration. All points in this equation are expressed in the sensor normalized coordinate frame.

The inverse distortion function is defined as :

$$p_d = c + \frac{1}{\alpha\sqrt{1 + r_r^2}} \cdot p_r \tag{3.9}$$

where r_r is the distance from p_r to the center of projection. Note that unlike most rectification models, this model offers a closed-form solution to both un-distortion and distortion operations.

3.6 Image Edges

We present here some geometric notions relative to the spherical model of the camera. After distortion rectification, a 3D straight line projects onto the spherical virtual camera as a great circle arc (see Figure 3-3). We refer to this arc as an *image edge*. Image edges are noted in lowercase italics (e) throughout this document. In the virtual *camera coordinate frame*, an edge $e = \{\text{o,a,b,n}\}$ is defined by the sensor center o (which lies almost at the origin) and the two unit vectors a and b corresponding to projection of the end points on the re-projection sphere. These three points define a plane in $\Re^3$ called the *edge plane*, whose normal vector n is referred to as the *normal vector* of the edge. Note that this vector is defined up to a reflection through the

plane *oab*, *i.e.* up to an arbitrary sign.

The *subtended angle* (or *length*) of an edge is defined as the angle subtended by its two unit vectors, a and b.

In the case of two edges, the *dihedral angle* between them is defined as the angle between the two normal vectors. Since the normal vectors are defined up to a reflection, the dihedral angle is defined modulo π. The dihedral angle can be uniquely defined by taking the smallest of the angles obtained by reflecting either normal vector. Note that the dihedral angle also corresponds to the angle between the two edge planes.

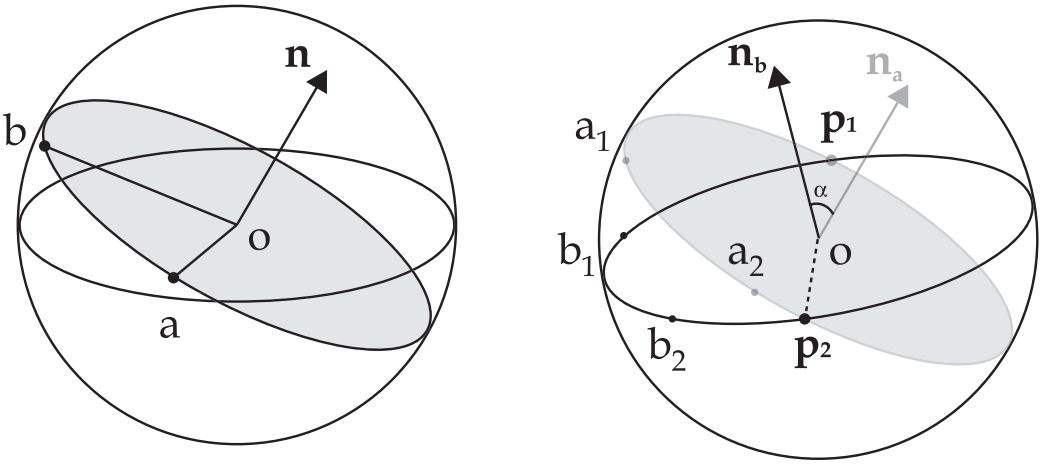

Figure 3-3: Left: the geometry of an edge ab in the spherical model. The center is noted as o, the two end points as a and b and the normal vector as n. Right: the geometry of two edges a_1b_1 and a_2b_2. α represents the dihedral angle. The poles are noted as p_1 and p_2.

Two image edges are *parallel* on the sphere if and only if their normal vectors are parallel. Note that two 3D lines in the environment generate two parallel edges if and only if they are parallel and define a common normal with the camera center.

The *intersections* (or *poles*) of two edges are defined as the cross products of the two normal vectors to the edges (see Figure 3-3). This definition is valid as long as the two edges are not parallel. The intersections are symmetric to each other with respect to the camera center.

The *maximum polar angle* of two edges is defined as the maximum angle subtended by the poles of the edges and the edge end points (see Figure 3-4). The *maximum*

polar angle is between 0 and π and gives an intuitive measure of the area spanned by two edges over the re-projection sphere. The dihedral angle and maximum polar angle provide a meaningful description of the geometric attitude of two edges in the virtual camera coordinate frame.

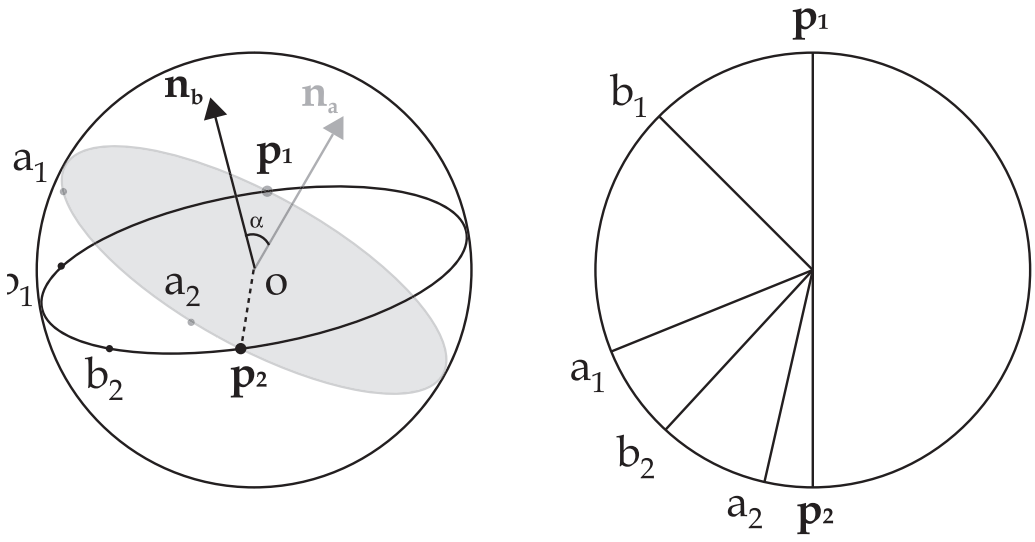

Figure 3-4: The maximum polar angle between the edge (o, a_1, a_2) and the edge (o, b_1, b_2) is defined as the maximum angle between the poles p_1 and p_2 and the edge end points a_1, a_2, b_1 and b_2. In this case, it is the angle between p_1 and a_2.

3.7 Edge Overlap

The *overlap* between two edges measures how much the two edges overlap on the sphere (Figure 3-5). This measure is particularly useful to compare an image edge with the interpretation edge generated by a model line.

Consider two edges $e_1 = \{o, a_1, b_1, n_1\}$ and $e_2 = \{o, a_2, b_2, n_2\}$. First, a_2 and b_2 are replaced by their normalized projection onto the plane of e_1. Define a local oriented coordinate frame by putting the x-axis along the unit vector a, the z-axis along the cross product of a and b and the y-axis defined by the right-hand rule. Figure 3-5 illustrates the corresponding configuration.

In this coordinate frame, let us define α_{a_1,b_1} as the angle between a_1 and b_1, α_{a_1,a_2} the angle between a_1 and a_2, α_{a_1,b_2} the angle between a_1 and b_2 and α_{a_2,b_2} the angle

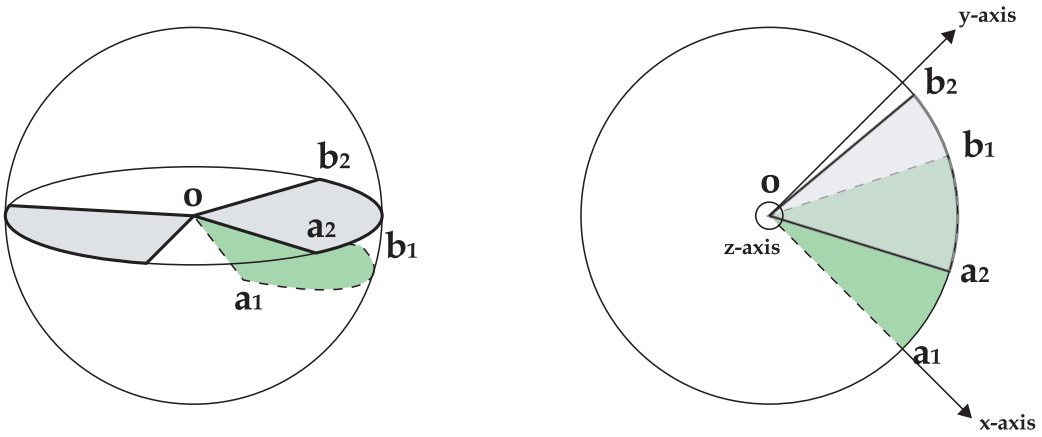

Figure 3-5: Left: the two edges in grey have the same dihedral angle with the edge in green. However, one of them has a significantly higher overlap with the green edge than the other. Right: projection view on the grey edge plane.

between a_2 and b_2. Then the overlap $g_{1,2}$ between e_1 and e_2 is defined as :

$$g_{1,2} = \begin{cases} 0 & \text{if } \alpha_{a_1,b_2} > \alpha_{a_1,b_1} \text{ and } \alpha_{a_1,a_2} > \alpha_{a_1,b_1} \\ 1 & \text{if if } \alpha_{a_1,b_2} < \alpha_{a_1,b_1} \text{ and } \alpha_{a_1,a_2} < \alpha_{a_1,b_1} \\ \frac{\alpha_{a_1,a_2}^2}{\alpha_{a_1,b_1} \cdot \alpha_{a_2,b_2}} & \text{if } \alpha_{a_1,b_2} - \alpha_{a_1,a_2} > \pi \\ \frac{(\alpha_{a_1,b_1} - \alpha_{a_1,a_2})^2}{\alpha_{a_1,b_1} \cdot \alpha_{a_2,b_2}} & \text{if } \alpha_{a_1,b_2} - \alpha_{a_1,a_2} < \pi \end{cases} \tag{3.10}$$

The first case corresponds to the case where the two edges do not overlap at all. The second case corresponds to the case where the second edge lies entirely into the first edge. The third and fourth cases correspond to the hybrid situation (Figure 3-5) and assume for simplicity that $\alpha_{a_1,a_2} < \alpha_{a_1,b_1}$. The overlap is defined as the ratio between the overlap region (α_{a_1,a_2}) and the squared sum of the sizes of the two edges (α_{a_1,b_1} and α_{a_2,b_2}). In the case where $\alpha_{a_1,a_2} > \alpha_{a_1,b_1}$, then α_{a_1,a_2} should be replaced by α_{a_1,b_2}. Note that the overlap function is not commutative ($g_{1,2} \neq g_{2,1}$ in general).

3.8 Topology on the Image Sphere

A number of algorithms described in this thesis rely on identifying a set of edges whose dihedral angle with a reference edge is smaller than some threshold value. In order to optimize this search, the sphere is tessellated using a icosahedral subdivision.

Each edge is classified according to the two faces of the icosahedron pointed to by its (symmetric) normal vectors. Hence, edges belonging to neighbor subdivisions are close to each other in terms of dihedral angle.

An icosahedron is a 20-faced uniform polyhedron. The first subdivision of a icosahedron is the icosahedron itself with 20 equilateral faces, 30 edges and 12 vertices. Each subdivision is obtained from the previous one by subdividing the equilateral triangle of each face into four equilateral triangles. The first subdivision has 80 triangles, the second 320 triangles, the third 1280 triangles and so on. Figure 3-6 illustrates the icosahedron subdivision.

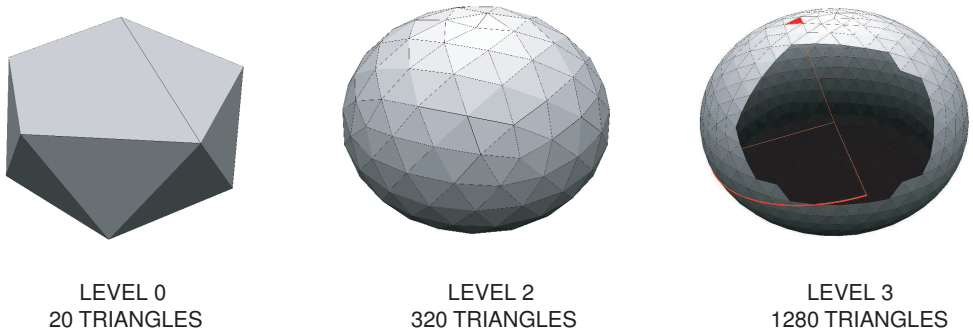

Figure 3-6: Sphere tessellation by icosahedron subdivision. Edges are sorted according to the index of the facet pointed to by the normal vector to the edge (in red).

After edge detection, edges are "linked" by mean of pointers to the two subdivision faces pointed by their normal vectors. Hence, given a subdivision face, it is possible to access the list of edges which normal vectors point toward this face in constant time. A table records the adjacency map of the subdivision, so that the "neighborhood" of a given edge (in the sense of dihedral angle) is also accessible in constant time. This data structure enables fast search of edge matches on the image given a projected 3D model line segment.

Chapter 4

Estimating Camera Pose From Line Correspondences

4.1 Absolute Orientation

The relative orientation problem, as described by Horn [20], consists of recovering the relation between two coordinate frames given a set of correspondences between observations in the first coordinate frame and 3D objects at known positions in the second coordinate frame.

It is well known that from three point matches there are at most four solutions [19] and from three line matches there are at most eight solutions [32]. When at least four points or four lines are provided, the system is over-constrained and a unique solution can be found using an iterative method. There exists an extensive collection of algorithms dealing with either points, lines, or both [12, 21, 18, 23, 25, 2].

4.2 Localization by Error Function Minimization

This thesis presents a hybrid method for recovering a unique solution from three line correspondences based on the additional constraint that each 3D line should lie in front of the camera observing it (frontality test).

Our method consists of matching detected (2D) image edges to known (3D) model

lines, without performing structure-from-motion. In this thesis, image edges are simply referred to as *edges* whereas model edges are referred to as *lines*. We chose to base our ego-motion estimation on line tracking, rather than point tracking, both because this approach seemed relatively unexplored in the vision literature, and because intuitively we expected long model lines to be robustly detectable and precisely localizable even in the presence of severe clutter. This intuition was only partially borne out by our experiments (see section 8).

Given a set of $n \geq 3$ correspondences between model lines and image edges, we recover the camera pose by minimizing an error function ξ, defined as the normalized square sum of angular disparities for each correspondence between image edge and (reprojected) model line (Figure 4-1):

$$\xi(R,T) = \frac{1}{n} \cdot \sum_{i=1}^{n} \alpha(e_i, R, T, l_j)^2 \tag{4.1}$$

where n is the number of correspondences, R and T are the rotational and translation components of the camera's rigid-body pose expressed with respect to the model coordinate origin, and α is the angle between the two planes spanned by the camera center and the observed image edge e_i and model line l_j respectively.

4.3 Pose Estimation from Three Line Matches

Given three model lines $\{l_1, l_2, l_3\}$ expressed in world coordinates and three observed edges $\{e_1, e_2, e_3\}$ expressed as planes in spherical camera coordinates, our algorithm determines the unique transformation that brings $\{e_1, e_2, e_3\}$ into alignment with $\{l_1, l_2, l_3\}$, with the additional constraint that each line l_i must lie in front of the camera that observed it. Our algorithm solves for the aligning rotation first, then for the translation.

We first set the camera pose to the origin and apply a rotation that brings e_1 into alignment with l_1. The axis of this rotation is defined by the normalized cross product of l_1 and e_1. At this point, any rotation along l_1 or along the normal to the

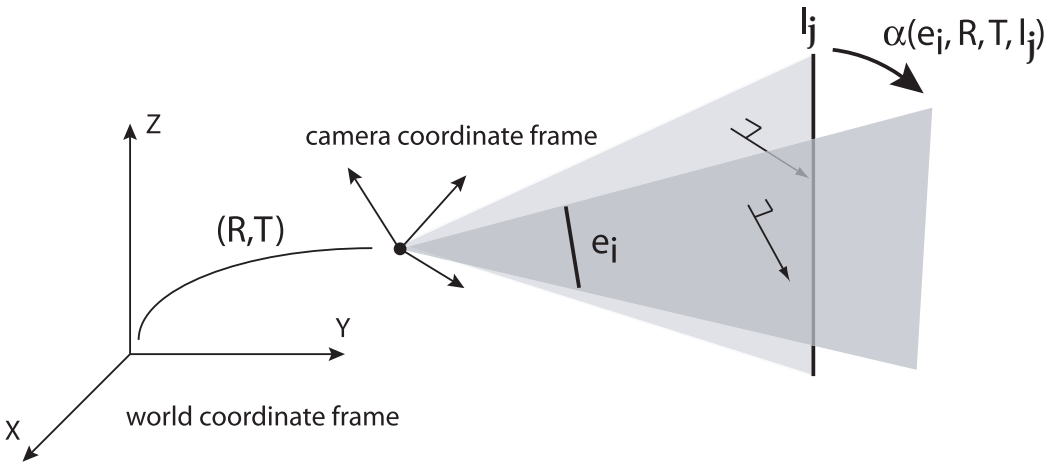

Figure 4-1: Angle between image edge e_i and model line l_j seen by camera (R, T). The angle α is equal to the inverse cosine of the inner product between the two normal vectors to the planes generated by the camera center, the 2D edge e_i and the 3D line $l_j, respectively$.

plane defined by the camera center and the two end points of l_1 does not modify the alignment between e_1 and l_1. Figure 4-2 illustrates this configuration.

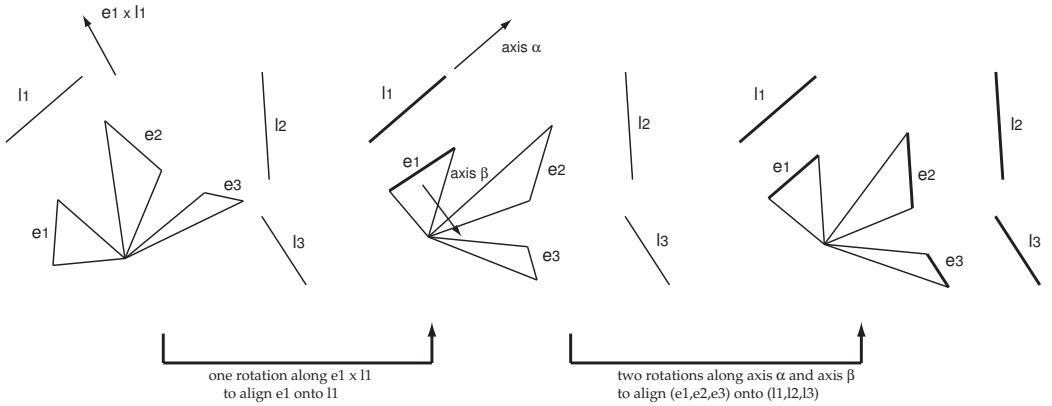

Figure 4-2: Aligning three image edges (e_1, e_2, e_3) onto three model lines (l_1, l_2, l_3). The first step is solved in closed-form. The second step is solved by minimizing the error function δ over α and β.

Let us define $R_1(\alpha)$ a rotation along l_1 of angle α and $R_2(\beta)$ a rotation along the normal to the plane defined by the camera center and l_1 of angle β. R_2 followed by R_1 does not modify the alignment between e_1 and l_1[1]. Since the axes of R_1 and R_2

[1]But note that R_1 followed by R_2 does modify this aligment!

are orthogonal, these rotations can bring any vector $p \in \Re^3$ into alignment with any vector $q \in \Re^3$. Therefore, one needs only to find the right α and β to bring e_2 and e_3 in alignment with l_2 and l_3. Since no simple closed-form solution to this problem is available, we rely on a simplex minimization algorithm to find the multiple (α, β) pairs of solutions. The function to minimize is defined as :

$$\delta = \sum_{i=1}^{i=3} \mid R_1 R_2(n_i) \cdot u_i \mid \tag{4.2}$$

where n_i is the normal vector to the plane defined by the camera center and the edge e_i, and u_i is a unit vector along line l_i. When the three alignments are satisfied, each dot product is equal to zero.

Each local minimum of the surface corresponds to a solution for (α, β). The different solutions can be found by initializing the simplex algorithm with various values of α and β. We use $\alpha, \beta \in \{k\pi/16\}, 0 \leq k \leq 15$.

Once all solutions have been found, we determine which one, if any, satisfies the frontality constraint. In most cases, only one solution, or no solution, passes the test successfully. When more than one solution is accepted, the algorithm simply reports failure since this outcome generally corresponds to a degenerate configuration.

Table 4.1 summarizes the algorithm. The whole process runs in less than 10 ms on a modern computer.

1: Given three lines $\{l_1, l_2, l_3\}$ and three edges $\{e_1, e_2, e_3\}$
2: Set the camera pose to the origin.
3: Align e_1 with l_1.
4: **for** each $\alpha, \beta \in \{k\pi/16\}, 0 \leq k \leq 15$ **do**
5: Initialize the simplex algorithm with the values of α and β.
6: Minimize δ to align e_2 and e_3 onto l_2 and l_3.
7: Run the frontality test and stores the solution if test is OK.
8: If exactly one solution remains, return solution (α, β) and SUCCESS.
9: Otherwise, return FAILURE.

Table 4.1: The Three-line Pose Estimation Algorithm for Rotation

Once the camera rotation has been recovered, solving for the camera translation is straightforward using translations along the axis normal to the planes defined by

the lines and the camera center.

4.4 Degenerate Configurations

In some cases, the three-line orientation problem may have an infinity of solutions, for instance when all three model lines are mutually parallel. Although we do not provide any rigorous proof of it, we assume that such configurations occur only when the three model lines are pair-wise skew. Two lines in $\Re^3$ are *skew* if they do not lie in the same plane. Therefore, the Three-line Pose Estimation Algorithm runs only on triplets of model lines which are pair-wise skew.

Chapter 5

Initialization

Initialization, in our system, requires determination of an initial set of valid correspondences between image edges and model lines. Rather than incur the geometric complexity of performing SFM from two or more images, then matching recovered 3D structure to known 3D model, we initialize from a single omnidirectional image. This simplifies the geometry involved to simple projection of model lines through our (known) camera model.

The core idea of the initialization algorithm is to find the camera pose (R_0, T_0), within a specified 6-DOF search region, that minimizes ξ. Clearly, neither exhaustive search of a 6-DOF space, nor naive RANSAC (with most model features occluded) [15] are tractable approaches. Instead, we present an initialization algorithm that takes as input a 3-DOF (x, y, z) volume (center $\tilde{T}$, diameter δ) known to contain the camera position, and returns the 6-DOF camera pose that minimizes ξ within the region. We use the notation $\tilde{T}$ to refer to an *estimate* of the true translation T to be determined. In case of multiple solutions, the algorithm returns all of them and prompts the user for the correct answer.

The 3D model is subdivided into a set of cylindrical search regions defined by a center $\tilde{T}$ (*node*), a diameter δ and the minimum and maximum height of the model (we assume that there is a model for each floor in case of multiple floors). The initialization algorithm runs on a search region specified by the user, or on several regions if the uncertainty of the camera position is larger than δ.

5.1 Probabilistic Line-Edge Matching

The core of the initialization algorithm is a method for generating putative correspondences between elements of two sets, n model lines $\mathcal{L}$ and m image edges $\mathcal{E}$, and using a selected subset of those correspondences to recover camera pose. The method is based on the following observation: that a *pair* of image edges forms a compatible match with a *pair* of model lines within a search region only if the dihedral angles formed by the two associated plane pairs differ by less than some bound determined by the region diameter.

Using this observation, we define a function that takes as inputs a triplet of model lines and a triplet of image edges and returns a fitness estimate that the triplets match. The fitness estimate is defined as the normalized product of overlaps between the dihedral angle ranges taken over the region interior. Given a set of model lines and a set of image edges, the algorithm computes the match fitness for each triplet of lines and edges and aggregates them within an $m \times n$ table.

$$p_{i,j}^{q,r} = \frac{min(\alpha_{ij,max}, \beta_{qr,max}) - max(\alpha_{ij,min}, \beta_{qr,min})}{(\alpha_{ij,max} - \alpha_{ij,min}) \cdot (\beta_{qr,max} - \beta_{qr,min})} \tag{5.1}$$

The table records the k best-scoring candidate matches for each model line (we use $k = 3$). The initialization method next performs a series of random samplings, each composed of a set of model lines, each line paired with one of its best-matching image edge. From each sample match set, the camera pose and resulting ξ value are computed. The algorithm returns the camera pose with the lowest ξ value. Figure 5-1 and table 5.1 illustrate the algorithm.

We could push this model one step further by defining the probability $p_{i,j}^{q,r}$ as a product of a Gaussian function for the edge dihedral angle range and a probability density function over the space of camera positions for the line dihedral angle range. However, experiments show that doing so significantly increases the computation time while not significantly improving the results.

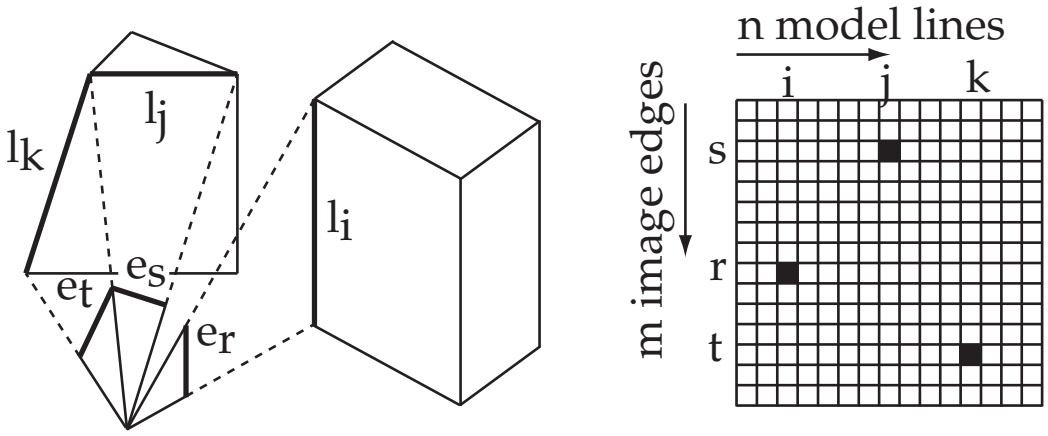

Figure 5-1: Probabilistic line-edge matching based on dihedral angle constraints. When a triplet of lines $\{l_i, l_j, l_k\}$ is compatible with a triplet of edges $\{e_r, e_s, e_t\}$, the scoring table is updated accordingly.

1: Given $(\tilde{T}, \mathcal{E}, \mathcal{L}, \delta)$
2: Initialize a $m \times n$ fitness table : $A[r][i] = 0, 0 \leq r \leq m, 0 \leq i \leq n$
3: **for** each triplet $\{l_i, l_j, l_k\} \in \mathcal{L}^3$ $(0 \leq i < j < k < n)$ and each triplet $\{e_r, e_s, e_t\} \in \mathcal{E}^3$ $(0 \leq r < s < t < m)$ **do**
4: Compute min, max dihedral angles for $\{l_i, l_j, l_k\}$ and dihedral angles for $\{e_r, e_s, e_t\}$
5: **if** dihedral angles match **then**
6: Update the table: A[r][i]++; A[s][j]++; A[t][k]++;
7: **for** each row A^j, $0 < j < n$ **do**
8: Determine the top k elements $\{A^j_{i_1}, \cdots, A^j_{i_k}\}$
9: Generate multi-hypothesis correspondence $c_j = \{l_j \mid e_{i_1}, \cdots, e_{i_k}\}$
10: Draw random correspondence match set:
11: **for** each sample $\{c_{j_1}, \cdots, c_{j_p}\}$ **do**
12: Select a random edge match for each c_{j_k}.
13: Minimize the ξ function over the sample.
14: Return the solution with lowest ξ value : $(R_0, T_0, \mathcal{S}_0)$.

Table 5.1: The $INIT_{LINE-EDGE}$ Algorithm

5.2 Rotation Voting

This algorithm is similar to the line-edge algorithm in the sense that it also compares pairs of image edges with pairs of model lines. For each pair of image edges $\{e_1, e_2\}$ and each pair of model lines $\{l_1, l_2\}$, the rotation voting algorithm computes the rotation that would bring the two edges in alignment with the two model lines. This

rotation is uniquely defined if the camera is not allowed to translate. However, the rotation does not exist if the dihedral angles between the two lines and the two edges (respectively) differ. In practice, a small error between these two angles is allowed to account for noise in the detected edge and error in the model.

Ideally, all rotations generated during this process should be the same and equal to the camera rotation itself. In reality, artifacts such as clutter and occlusion add noise to the results. However, we still expect an aggregation of *votes* around the true camera rotation. The algorithm therefore looks for the rotation angles having the most votes, then determines the rotation axis corresponding to each angle. For each rotation, the algorithm then looks for the optimal camera translation using an exhaustive search around the initial camera position estimate. Each pose is then scored as described in section 4 and the best pose is retained. Table 5.2 summarizes the algorithm.

1: Given $(\tilde{T}, \mathcal{E}, \mathcal{L}, \delta)$
2: **for** each pair $\{l_i, l_j\} \in \mathcal{S}^2$ and each pair $\{e_r, e_s\} \in \mathcal{E}^2$ **do**
3: Compute min, max dihedral angle for $\{l_i, l_j\}$ and dihedral angles for $\{e_r, e_s\}$
4: **if** dihedral angles match **then**
5: Compute the rotation R_{ijrs} which brings $\{e_r, e_s\}$ into alignment with $\{l_i, l_j\}$
6: Store R_{ijrs} in memory.
7: Histogram rotation angles; retain peak angles $\{\alpha_1, \cdots, \alpha_k\}$ $(k = 3)$
8: **for** each α_j **do**
9: Keep all rotations differing in magnitude from α_j by less than a given threshold.
10: Compute the number of neighbors for each remaining rotation.
11: Keep the rotation R_j with highest number of neighbors.
12: Keep the rotation $R \in \{R_0, \cdots, R_k\}$ with highest number of neighbors.
13: Set the camera rotation $R_0 \leftarrow R$.
14: Sweep the cylinder $(\tilde{T}, \delta)$ using a $10x10x10$ grid of 3D points to find the camera translation T with the lowest $\xi(R, T)$ value.
15: Populate the set $\mathcal{S}_0$ with all model lines having acceptably close matches on the current image.

Table 5.2: The $INIT_{R-VOTING}$ Algorithm

5.3 System Initialization

In practice, the system runs both algorithms $INIT_{LINE-EDGE}$ and $INIT_{R-VOTING}$ and returns the solution with the lowest ξ value. For increased robustness, each algorithm is followed by a standard simplex minimization of ξ. Table 5.3 summarizes the Initialization Algorithm.

1: Given $(\tilde{T}, \delta)$
2: Detect edges on the first frame $\rightarrow \mathcal{E}$
3: Determine the set of visible model lines $\mathcal{L} = VIS(T)$
4: Run $INIT_{LINE-EDGE}$ and $INIT_{R-VOTING}$ on $(\tilde{T}, \mathcal{E}, \mathcal{L}, \delta)$.
5: Keep the solution with the lowest $\xi(R, T)$ value.
6: Return the corresponding solution $(R_0, T_0, \mathcal{S}_0)$.

Table 5.3: The Initialization Algorithm

Chapter 6

Maintenance

This section describes the algorithm's maintenance component. Given a set of edge-line correspondences $\mathcal{S}_t$ at frame t, the *maintenance* problem is to identify a set of correspondences $\mathcal{S}_{t+1}$ at frame $t+1$ and to compute the new camera pose (R_{t+1}, T_{t+1}). To account for clutter, we use a multi-hypothesis approach combined with a hue-based inter-frame constraint.

6.1 Hue-based Edge Matching Constraint

Each image edge is associated with a hue mean and variance for two five-pixel wide regions, one on each side of the edge. The color is encoded in Hue-Saturation-Value format. Given a correspondence between a model line and an image edge in frame t, the algorithm looks for matching edges in frame $t+1$ by considering all edges with a dihedral angle smaller than a given threshold and a hue distance smaller than a given threshold (with respect to the edge at frame t). We mark a correspondence *observed* if it satisfies these two criteria. Figure 6-1 illustrates the correspondence update mechanism.

The color distance δ between an color strip with HSV mean and variance (μ_1, σ_1) and a color strip with HSV mean and variance (μ_2, σ_2) is defined as :

$$\delta = \frac{1}{2} \cdot (\mid \mu_1 - \mu_2 \mid + \mid \sigma_1 - \sigma_2 \mid) \tag{6.1}$$

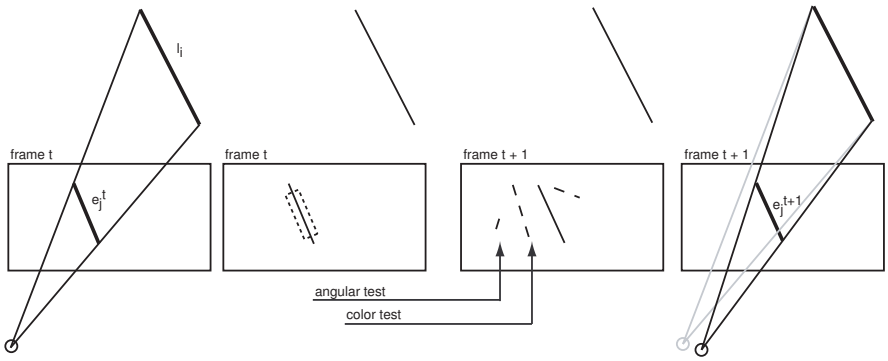

Figure 6-1: Correspondence update mechanism: the angular constraint and the color constraint determine the most likely image match e_j^{t+1} in frame $t+1$ given a correspondence between the 2D image edge e_j^t and the 3D model line l_i at frame t.

where μ_1, μ_2, σ_1 and σ_2 are expressed in the normalized HSV space. The question of weighting mean and variance differently has not been studied in this thesis, as good tracking results have been obtained as is.

6.2 A Random Sample Algorithm

Once each correspondence has been updated, the system draws a series of random samples from the set of new correspondences, computes the new camera pose from each sample and calculates the ξ function over the set of remaining correspondences (those which were not in the sample). The pose with minimum ξ value is retained.

6.3 Correspondence Lifetimes

In order to further improve the robustness of the matching process, we implement a simple finite-state machine for each edge-line correspondence. The machine has three states: an entry state *unknown*, and two other states *pending* and *accepted*. A particular correspondence's state moves to *pending* after it is observed once, and to *accepted* after it is consistently observed over k consecutive frames (we use $k = 4$).

A correspondence status degrades from *accepted* to *pending* if it has been unobserved for at least 1 but no more than $k-1$ frames, after which it either upgrades back to *accepted* or (when the edge has been unobserved in k consecutive frames) degrades to *unknown*.

The hue signature of an edge is retained as long as it is *accepted* or *pending*. However, only correspondences with *accepted* status are used for pose recovery in the current frame. Figure 6.1 summarizes the Maintenance Algorithm.

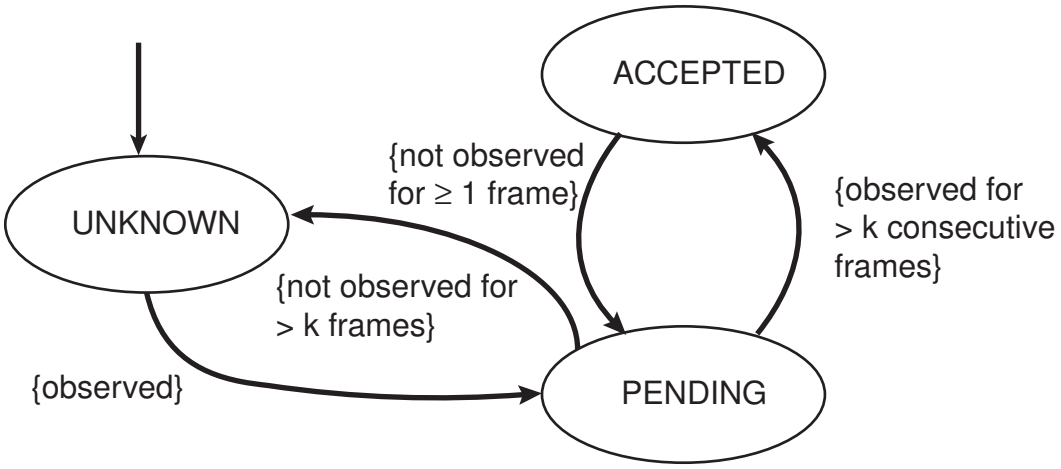

Figure 6-2: Correspondence state machine

The number of correspondences per sample (λ) is defined by the typical number of correct correspondences required to accurately determine the camera pose. We use $\lambda = 10$ (see § 8.8). The number of samples s_t is defined as the minimum number of draws of p out of q elements required to achieve 95% odds of success assuming the set has b% outliers, *i.e.*:

$$s_t = p \mid (1 - \binom{(1-\frac{b}{100})q}{p} / \binom{q}{p})^n \leq 5\% \tag{6.2}$$

Table 6-3 evaluates s_t for different values of b and $|\bar{s_t}|$. The number of sample rounds needed increases as the noise increases or the total number of correspondences decreases. Thus we conclude that the computational load of maintenance depends on the visibility of the 3D model. In the case of little clutter, a few rounds are sufficient to run the maintenance successfully. On the other hand, in the case of high clutter,

1: Given $(R_t, T_t, \mathcal{S}_t)$
2: Detect edges at frame $t+1 \rightarrow \mathcal{E}$
3: **for** each correspondence $\{l_i, e_j\}$ in $\mathcal{S}_t$ **do**
4: Search for match in $\mathcal{E}$ satisfying color consistency with e_j and acceptable angular error with l_i.
5: If a match is found, update the correspondence.
6: From the set $\bar{\mathcal{S}}_t$ of correspondences with *accepted* status, draw s_t samples of λ correspondences ($\lambda \ll |\bar{\mathcal{S}}_t|$).
7: **for** each sample **do**
8: Compute the camera pose by minimizing ξ over the λ correspondences using a simplex method.
9: Score the sample by computing ξ over the remaining correspondences in $\bar{\mathcal{S}}_t$.
10: Keep the sample with the lowest score and update the camera pose at frame $t+1$.
11: Update each correspondence in $\mathcal{S}_t$ according to the state machine $\rightarrow \mathcal{S}_{t+1}$.
12: Query the new visibility set $\mathcal{L}_{t+1} = VIS(T_{t+1})$
13: If $\mathcal{L}_{t+1} \neq \mathcal{L}_t$, remove demoted model lines and insert new model lines with status *unknown* in $\mathcal{S}_{t+1}$.

Table 6.1: The Maintenance Algorithm

as many as several thousands rounds might be needed. We found that $5,000$ rounds worked well in our environment.

	$\lvert\bar{\mathcal{S}}_t\rvert = 30$	40	50	60
$b = 10\%$	10	9	9	8
$b = 30\%$	254	193	167	152
$b = 50\%$	29971	13743	9413	7516

Figure 6-3: Number of sample rounds needed vs. clutter percentage (b) and number of correspondences ($|\bar{\mathcal{S}}_t|$).

Chapter 7

Offline Visibility Computation for Wide-Area Scaling

One of the novel aspects of this project is the off-line computation of the visible model lines and faces from a set of predefined positions in the model (referred to as *nodes*). This pre-computation enables the method to bound the number of model lines to take into account when the camera position is known or approximately known and therefore allows the system to scale to large environments.

7.1 Model Coordinate Subdivision

The initialization algorithm uses a visibility data structure. This data structure consists of a discretization of the 3-DOF model space into *nodes* at a spacing of about one meter. Each node is associated with a volumetric *cell* containing all points closer to that node than to any other node (each cell is indeed the Voronoi region of one node, but the cell boundary is known by construction, rather than computed from the arrangement of node positions). Our implementation uses 2D subdivision, but extension to 3D is straightforward. The initialization algorithm is invoked with a specified search region. It identifies which cells intersect this region, and searches these cells for the camera pose with lowest ξ score.

Although the offline computation runs fairly quickly (about half a second per node

for a lab space environment), storing and searching the visibility set in a Look-Up Table (LUT) is faster. The LUT is composed of an *index file* which contains, for each node, a pointer to a *data file* containing the set of visible model lines from the *node* position. Each model line and face is represented by a unique integer.

7.2 Visible Model Faces

Our line visibility method first identifies the set of visible model faces from a given position. We present here an algorithm for computing this set. Testing the occlusion of every model face against all the other faces becomes quickly untractable as the number of faces increases.

Instead, we rely on the efficiency of modern GPUs and use a framebuffer-based algorithm defined as follows (Figure 7-1). From a given position in the model, we render the 3D model in openGL, assigning a unique color to each face[1]. Then, we examine each pixel of the image and declare as visible the face corresponding to the color at that pixel position. This algorithm is extremely robust, as it leaves the responsibility of complex geometric computations to the GPU and scales well with the number of faces.

In our system, we use this algorithm as an input to the line visibility computation, since only those lines that are part of a visible face may themselves be visible. This makes the line visibility computation faster and more robust against false positives which may occur due to Z-buffer inaccuracy.

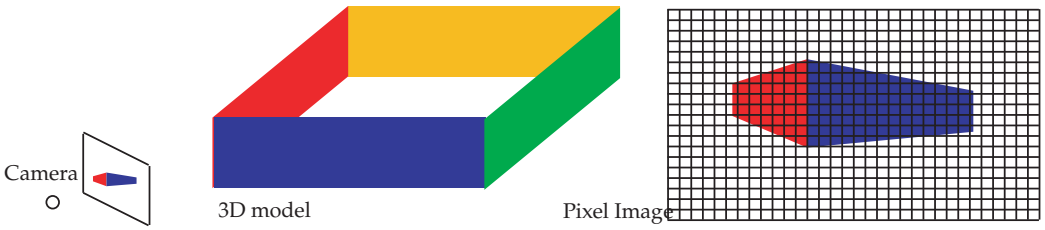

Figure 7-1: Computation of the set of visible faces from a given camera position using pixel color information.

[1]Colors are distributed uniformly over the 24 bit RGB color space containing over 16 million elements.

7.3 Visible Model Edges

We now describe an algorithm for computing the set of visible edges from a given position in the model. Again, we rely on an OpenGL feature referred to as the *feedback buffer* [41]. The OpenGL *feedback buffer* is a mechanism through which OpenGL reports the set of objects to be displayed without actually displaying them on the screen. In our case, the objects are actually 3D points. Each object in the *feedback buffer* is assigned an (x, y) position on the image as well as a *z-value* which corresponds to the depth of the object if it were to be actually displayed on the image. This *feedback buffer z-value* can then be compared to the Z-buffer value at position (x, y) on the image where the entire model would be displayed. If it is larger (*i.e.* at a greater depth), it means that the object is occluded. Otherwise, the object is visible. To combat aliasing effects, we use a window of 3x3 pixels to compare the buffer z-values.

In practice, we subdivide each model edge into a set of 3D points with one-inch spacing. We test the visibility of each 3D point along the model edge. If a significant ratio of the points are visible (*e.g.* 20%), the model edge is declared visible. Doing so for each model edge from a given *node* position gives a list of the visible model lines which are stored in the LUT. Figure 7-2 illustrates our algorithm.

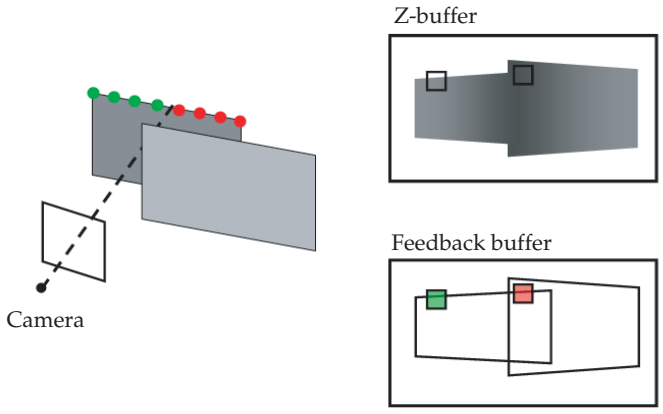

Figure 7-2: Computation of visible lines from Z-buffer and feedback buffer comparison. Visible points are shown in green; occluded points are shown in red.

Chapter 8

Experimental Results

We demonstrate our system on one synthetic and three real datasets:

- SYNTHETIC: 6-DOF motion within a simulated lab space;
- LAB: rolling 3-DOF (x, y, θ) motion within a real lab space;
- CORRIDOR: rolling 3-DOF exploration through adjoining buildings; and
- HAND-HELD: hand-held 6-DOF motion within a real lab space.

	SYNTH	LAB	CORRIDOR	HAND-HELD
MIT building number	36, 26	32-33x	36, 26	32-33x
Number of frames	100	1,500	7,800	1,900
Motion type	4-DOF	4-DOF	4-DOF	6-DOF
Excursion duration (min)	.33	5	26	2
Excursion length (m)	10	120	300	5
Walking speed (m/s)	0.5	0.4	0.20	0.04
# 3D faces in model	1900	675	1900	675
# 3D edges in model	7,400	3,000	7,400	3,000
Model surface area (m^2)	7,000	450	7,000	450
LUT size (average # faces/per node)	30	120	30	120
LUT size (average # edges/per node)	80	140	80	140

Table 8.1: Test datasets

The following results were obtained by setting the parameters described in table 8.2.

Parameter name	value
Edge Detection	
Minimum edge length (pixels)	20
Minimum threshold for contour detection	20.0
Maximum deviant for contour to line conversion	1.0
Initialization	
Search region diameter (m)	2.5
Number of edge match candidates in $INIT_{VOTING}$	3
Number of top rotations in $INIT_{R-VOTING}$	5
Maximum angular error for correspondence generation (deg.)	10.0
Maximum number of model corner per image corner (corner alignment method)	15
Minimum subtended angle of expected line (deg.)	10.0
Maintenance	
Minimum number of frames before promotion to Accepted	4
Maximum HSV distance for correspondence update	0.03
Maximum angle change for correspondence update (deg.)	15.0
Minimum overlap between image edge and model line	10%
Scoring function	
Minimum overlap to accept correspondence	80%
Maximum angular error to accept correspondence (deg.)	30.0

Table 8.2: Parameter values

8.1 Pose Estimation from Three Line Matches

Figure 8-1 shows a graph of δ with respect to α and β. The local minima of the function correspond to the solutions of the pose estimation problem from three line matches (see section 4.3). The solutions are found using a standard minimization method with various seed values for α and β.

8.2 Offline Visibility Computation

Figure 8-2 demonstrates the off-line visibility computation on a typical lab space environment. The nodes in the grid are separated by 50 inches. A visibility set is attached to each node and can be queried in real time during both the Initialization and the Maintenance. The visibility set of a standard space (450 m^2) takes about five minutes to compute on a standard desktop and requires 10 MB of disk space.

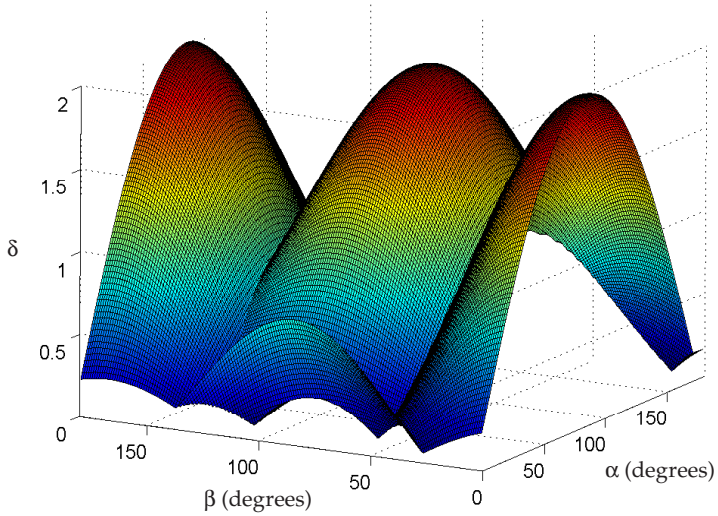

Figure 8-1: Graph of δ with respect to α and β in the pose estimation problem from three line matches. Local minima of the function correspond to the solutions for α and β.

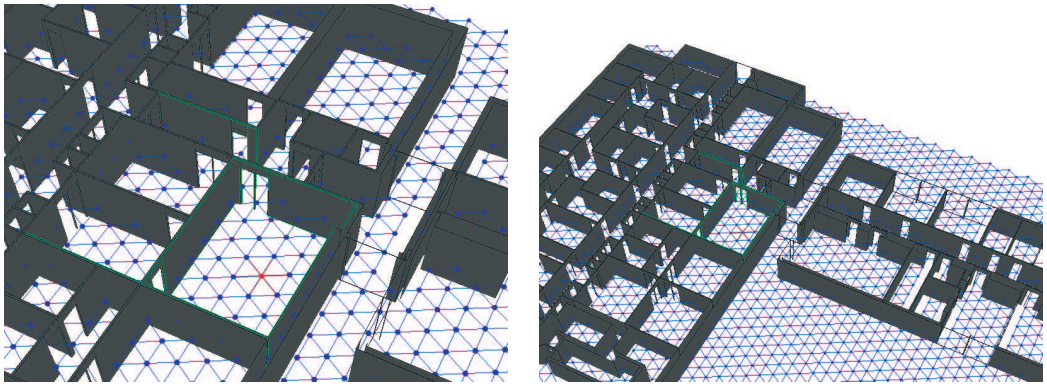

Figure 8-2: Off-line computation of the visibility set. The grid is displayed in blue. The lines visible from the node shown in red are displayed in green. Note the relatively small size of the visible set with respect to the size of the model.

8.3 Initialization

8.3.1 Probabilistic Line-Edge Matching

Figure 8-3 shows an example output of the $INIT_{P-LINE}$ algorithm. For a given 3D model line, the image shows the three best candidate image edge matches. Note that one of the candidates is the correct match in each case. Figure 8-4 shows several matching tables generated by the algorithm. Each table is of size $M \times N$ where M is the number of 2D image edges and N is the number of 3D model lines. In the synthetic case, the synthetic images have been generated so that $M = N$ (each visible model line is projected on the synthetic image in the same order). Therefore, the peaks are expected to accumulate on the diagonal of the matching table. In the real case, such ordering could not be enforced. However, the success ratio appears in practice to be about 50% over a large set of initializations.

Figure 8-3: An example output of the fitness-based line matching algorithm. A 3D model line is selected on the left. The three best image edge matches for this line are displayed on the right. The system found the correct match (in blue on the left-most tile).

8.3.2 Rotation Voting

Figure 8-5 shows an example rotation histogram generated by $INIT_{R-VOTING}$ on a real image (shown on figure 8-8). The true camera rotation is marked by a dashed line. The algorithm considered the top five rotations (marked with red stars) and recovered the correct camera pose.

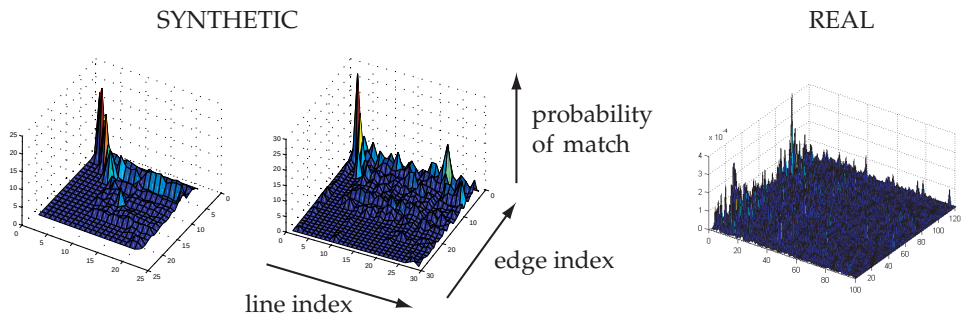

Figure 8-4: Output of the probabilistic line matching algorithm. The probability of match between each model line and each image edge is displayed as a surface function. A peak in the table corresponds to a high match probability between the corresponding model line and image edge. Note that the data is noisier in the real case than in the synthetic case.

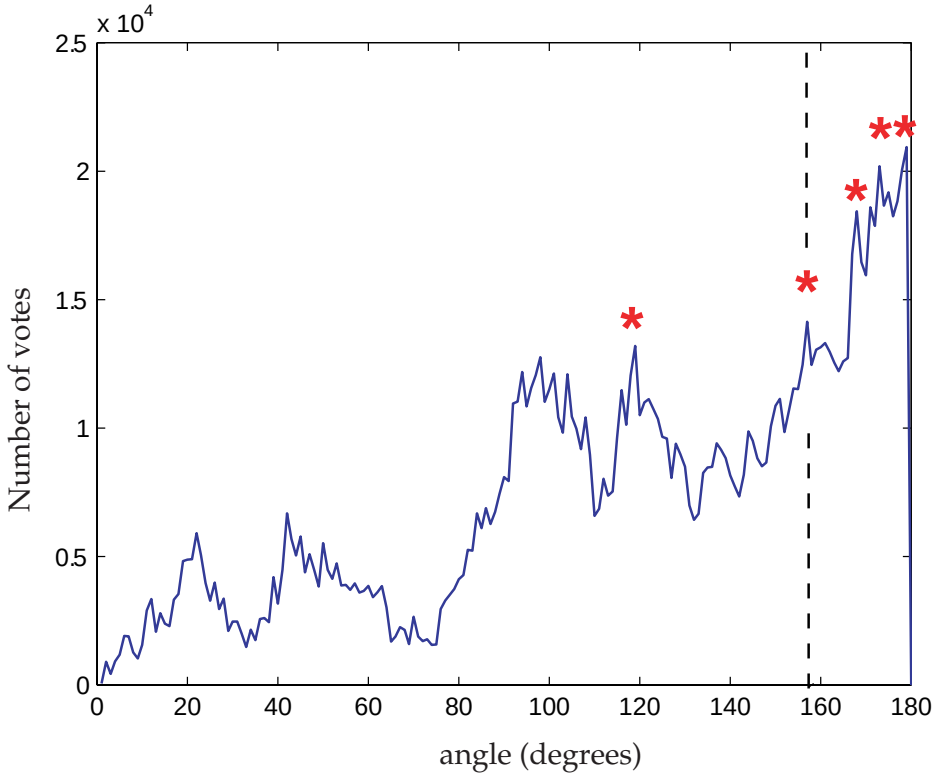

Figure 8-5: Output of the rotation voting algorithm. The rotation angles are aggregated in a histogram. Peaks correspond to the most likely camera rotation angles (red stars). The true rotation angle is marked with a dashed line. The algorithm detected the true camera rotation angle.

8.3.3 Corner Matching

Figure 8-6 illustrates the corner matching algorithm. Image corners are shown on the top. The system selects two corners on the image and two corners in the 3D model (along with the edges attached to them, shown in red on the figure) and computes the corresponding camera pose.

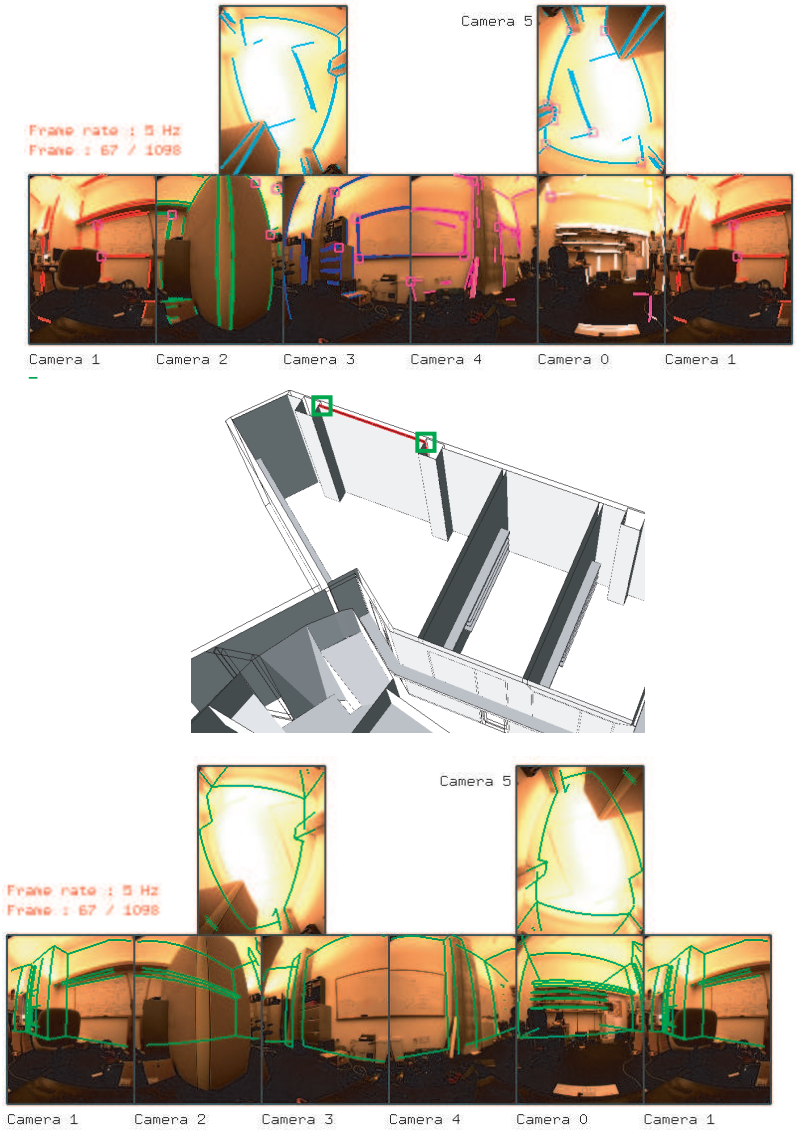

Figure 8-6: Top: Corner detection. Middle: 3D map. Bottom: Re-projection of the structure after alignment. The correct alignment is obtained after aligning the two model corners shown in green in the model with the two image corners highlighted in green on the image.

8.3.4 Vanishing Points

Figure 8-7 illustrates the vanishing point alignment method. The observed vanishing points (shown in red) and the expected vanishing points (shown in blue after alignment) are aligned in order to estimate the 3D camera rotation. Note that this method is particularly successful when the image exhibits strong vanishing points. This method complements the other initialization methods.

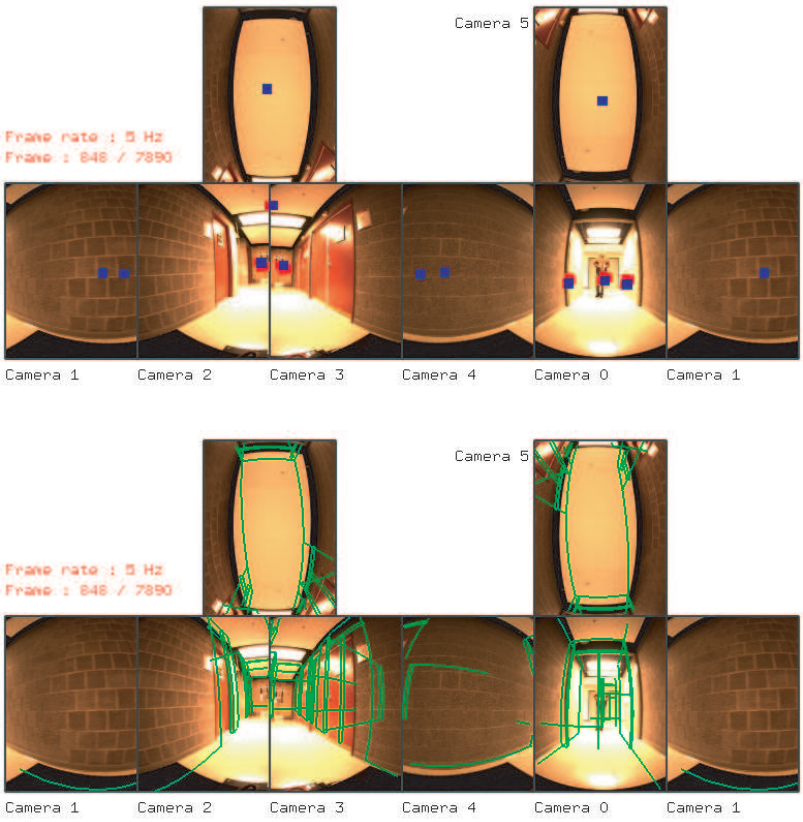

Figure 8-7: Pose initialization from vanishing point (VP) alignment. Top: image VPs are displayed in red; expected VPs computed from the model are displayed in blue. Bottom: the structure is reprojected in green after aligning the image VPs with the model VPs. Note that even though the solution is not exact, the alignment of VPs gives a very good estimate of the camera rotation.

8.4 Initialization Example

Figure 8-8 shows the result of Initialization on a real image. The search space was 2.5 meters wide. The corresponding camera pose was found by the $INIT_{P-LINE}$ algorithm.

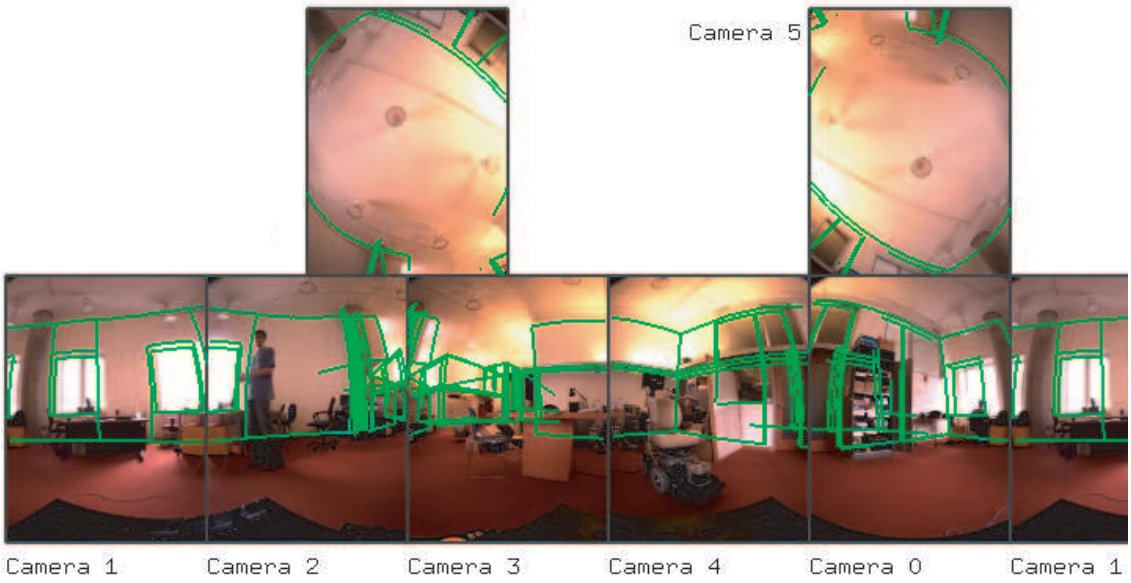

Figure 8-8: Omnidirectional image and re-projected 3D model (in green). Note the accuracy of 6-DOF localization despite clutter in the scene.

8.5 Maintenance

This section shows results for the maintenance phase on the real datasets. We also present detailed results about the hue-based matching constraint and the correspondence state machine.

8.5.1 Hue-Based Edge Matching Constraint

Figure 8-9 illustrates the color-based matching constraint. A correspondence (shown in red) is tracked over several successive frames despite the presence of several unmodeled edges nearby (shown in yellow).

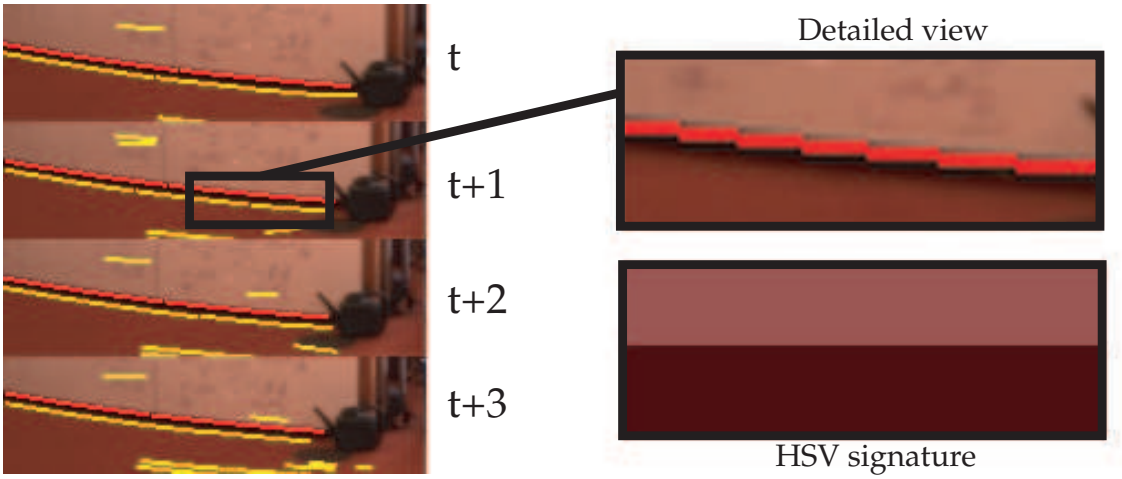

Figure 8-9: The average and standard deviation of the hue along the edge is used to track edges robustly from frame to frame. The tracked edge is displayed in red. The candidates are shown in yellow.

8.5.2 Correspondence Lifetimes

Figure 8-10 shows the lifetime of two correspondences in the HAND-HELD dataset. In the first case (top), the correspondence involves a model line which is consistently occluded by a couch. As a consequence, the correspondence rarely reaches Accepted status. On the other hand, the other correspondence (bottom) involves a model line which is visible up to frame 245, disappears from the camera field of view from frame 245 to frame 350, then becomes visible again after frame 350. The state of the correspondence is updated accordingly. Figure 8-11 shows the corresponding sequence of video frames. The correspondences are shown in green and red (respectively).

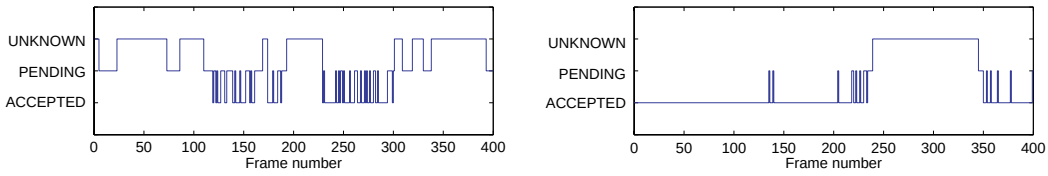

Figure 8-10: Correspondence state of two model lines over time (sequence HANDHELD dataset). Left: a model line which is consistently occluded by a couch. The correspondence spent most of its lifetime in Pending or Unknown state. Right: a model line which is partially occluded from frame 245 to frame 250.

Figure 8-11: Video sequence for the two correspondence in Figure 8-10. The model edge shown in green is visible at frames 200 and 400 but disappears at frame 300. The base of a partition shown in red is consistently occluded by the couch (detail view).

8.6 Real sequences

We present here the results of our algorithm on three real datasets: the LAB dataset, the HAND-HELD dataset, and the CORRIDOR dataset.

8.6.1 The LAB Dataset

The LAB dataset contains 1,554 frames captured at 5 Hz on a wheeled, remote-controlled robot. The camera was setup on the robot in a vertical position, approximately 30 inch high. Although the camera was moving in a flat, 2D world, no assumption was made on the camera motion. However, the recovered flat motion offers a crude validation of the results. The sequence was captured in the Stata Center RVSN Lab, for which a 3D model was built by hand from a coarse 2D map.

The initialization algorithm successfully recovered the camera position in about two minutes given a 2.5 meter search region provided by the user. The algorithm was then able to track the camera motion without losing "lock" until the end of the sequence. Figure 8-12 shows a view of the recovered camera motion in the 3D model coordinates.

The robot started and ended approximately at the same position, which was recovered by the localization algorithm, thus proving the absence of drift. The algorithm tracked the camera motion without losing "lock" across the whole sequence. Several people walked through the lab during the capture, and the lab was significantly cluttered with unmodeled furniture, as can be seen in figure 8-13.

In the middle of the sequence, the camera passed through a narrow passage between a corner of the lab space and a large whiteboard, which induced significant occlusion and a large change in visibility. However, the algorithm succeeded in maintaining the "lock", while simpler versions of the algorithms which did not incorporate the correspondence state machine and the hue-based matching constraint proved to fail at that particular location. In general, the maintenance algorithm proved to succeed in tracking the camera under severe occlusion and scene changes in several other datasets.

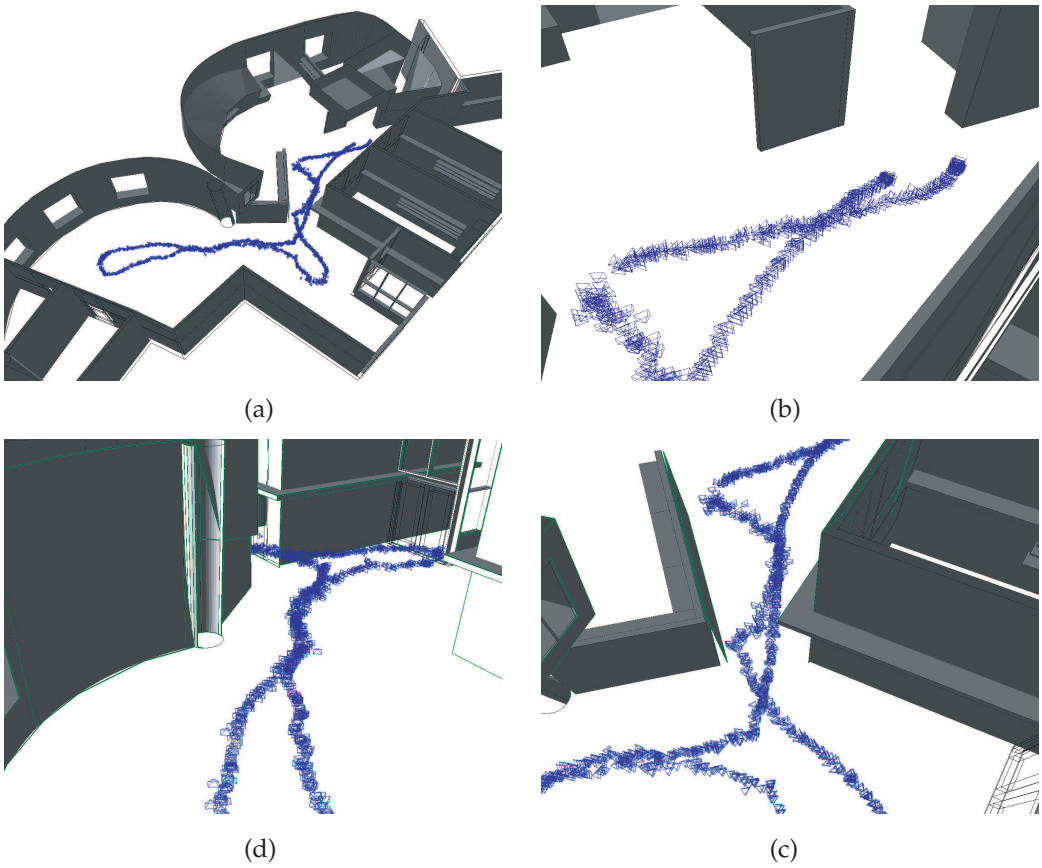

Figure 8-12: Recovered 3D motion for the LAB dataset (1,500 frames, 120 m at 0.4 m/s). The camera was attached to a robot and followed a flat 2D motion. (a) Overview of the excursion; (b) and (d) Detail view of the camera motion; (c) Area of high occlusion and large visibility change.

8.6.2 The HAND-HELD Dataset

The HAND-HELD dataset is composed of 1,954 frames captured at 15 Hz in the Stata Center RVSN Lab. The camera was hand-held and followed a truly arbitrary 3D motion involving combined translations and rotations. In order to avoid motion blur due to low lighting, the camera was moved slowly (0.04 m/s). However, the maintenance algorithm can handle faster motion, as proved in the other datasets.

As in the LAB dataset, the initialization algorithm found the correct camera position in about two minutes. The camera motion was then tracked successfully across the sequence. Figures 8-14 and 8-15 show the recovered camera motion in the 3D model coordinates.

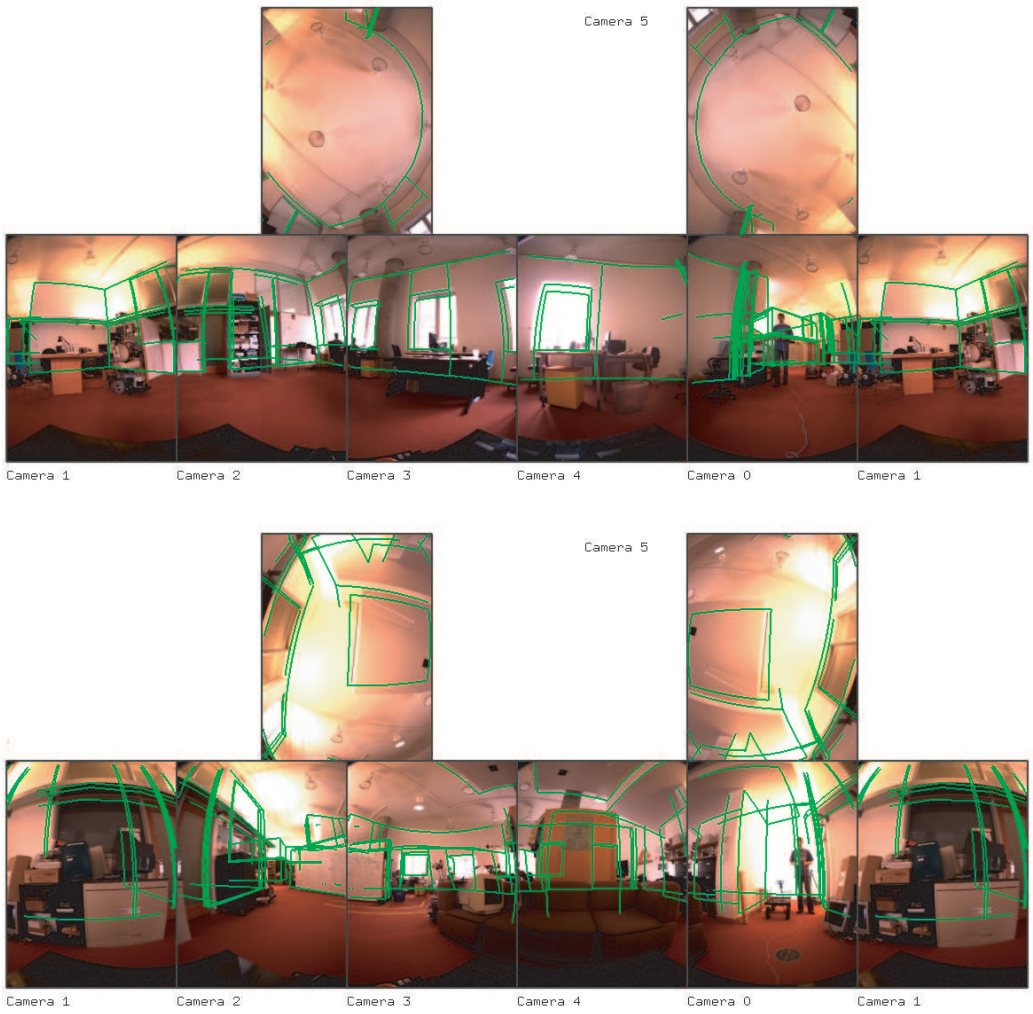

Figure 8-13: LAB dataset; 3D model re-projected on the video.

8.6.3 The CORRIDOR Dataset

The CORRIDOR dataset is composed of 7,890 frames captured at 5 Hz. The camera was setup on the same robot as in the LAB dataset. However, the dataset was captured on a longer excursion (26 minutes) across two buildings and several classrooms. This algorithm also demonstrated the system in a more conventional 3D environment than the Stata Center. The 3D model is courtesy of the MIT Building Model Generation (BMG) Project and was extruded automatically from 2D blueprints of the buildings.

The robot started at the third floor entrance of building 26. The initialization algorithm succeeded in finding the correct camera pose, despite the repetitive aspect

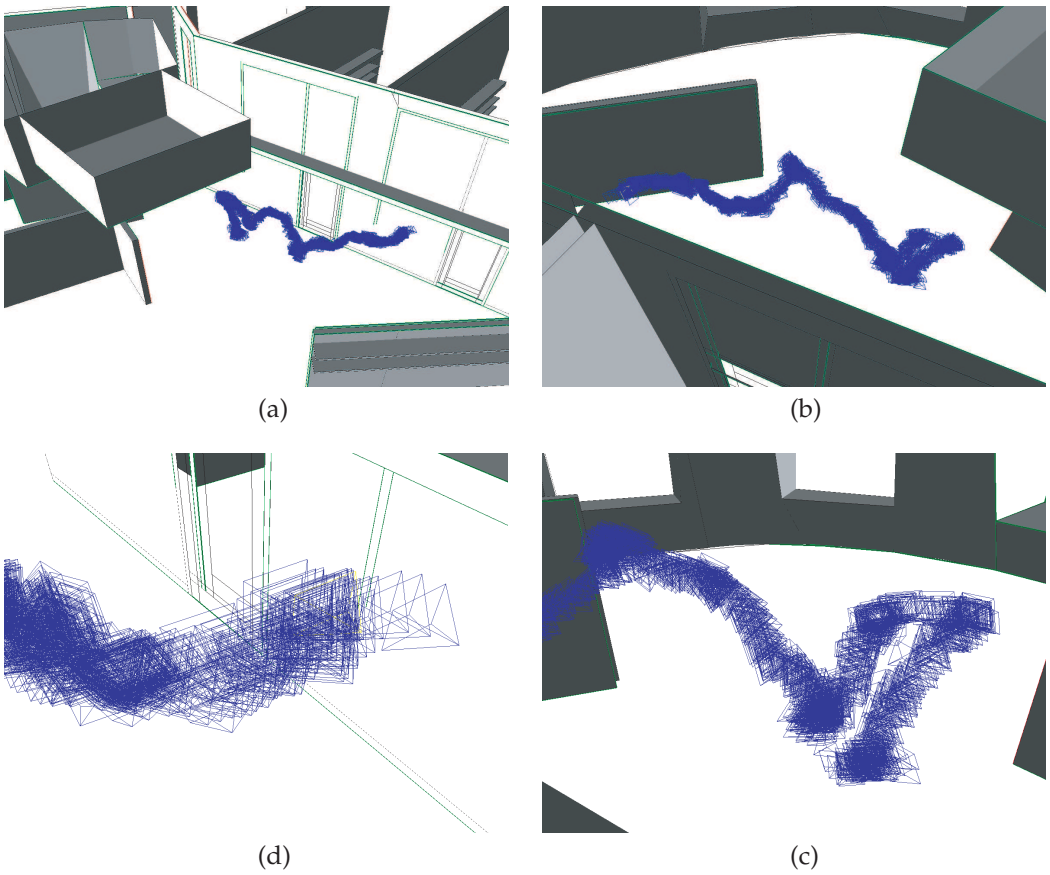

Figure 8-14: Recovered egomotion for the HAND-HELD dataset (1,900 frames, 5 m at 0.04 m/s). The camera followed an arbitrary 3D motion. (a) and (b) Overview of the camera motion; (c) and (d) Detail view of the camera path.

of the corridor. However, the initialization turned out to be sometimes unsuccessful when run along the corridor as a sanity check. The corner-based and vanishing point-based algorithms helped tackle repetitive environments more successfully and provided correct initialization in about 80% of cases.

The robot followed a flat motion across building 26 and 36. In building 36, the robot entered two classrooms. The maintenance algorithm did not loose "lock" despite the drastic change of visibility. Due to the high level of clutter, the algorithm lost "lock" a few times inside the classrooms, but the lock was recovered by running the initialization in the search region occupied by the camera when the lock was lost. The system is able to automatically detect a loss of lock when the number of correspondences in *Accepted* state drops under a given threshold (typically, 10). Figures 8-16

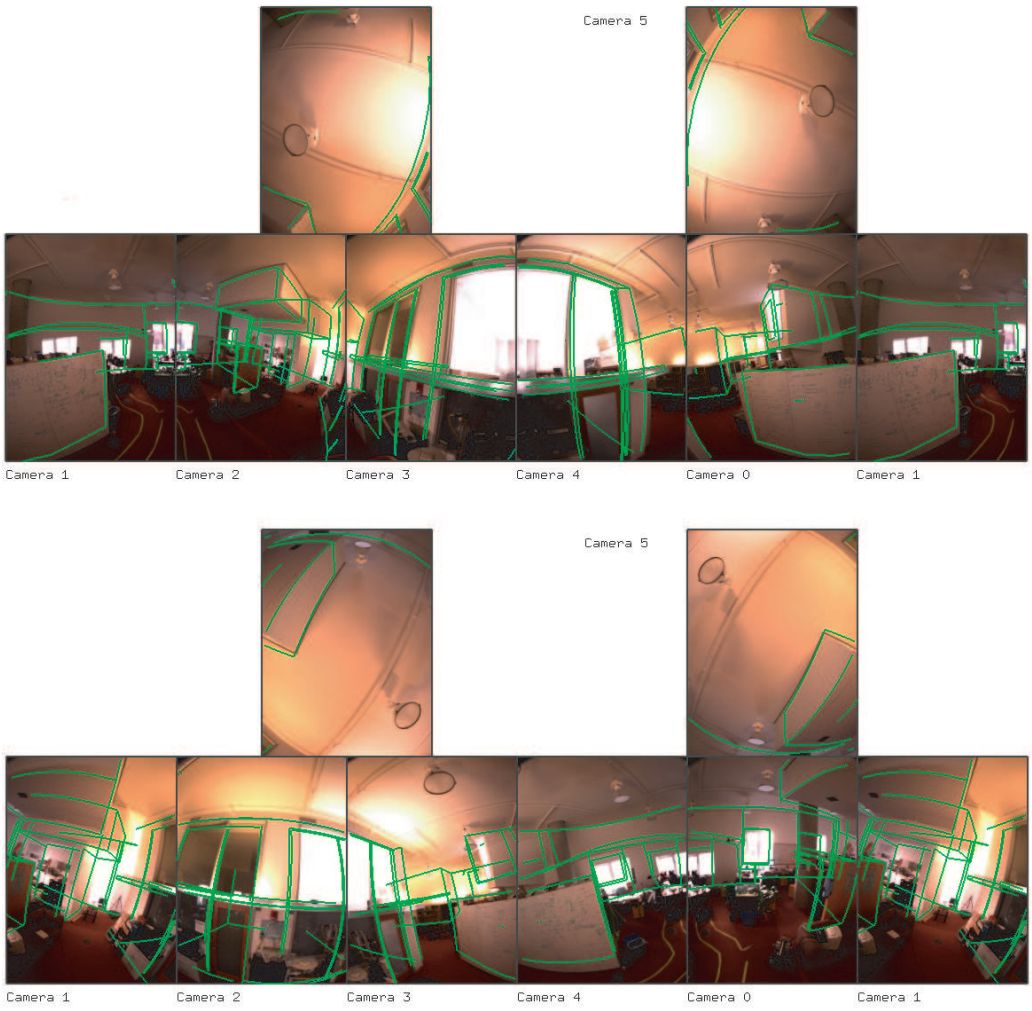

Figure 8-15: HAND-HELD dataset; 3D model re-projected on the video.

and 8-17 show the reconstructed camera motion in the 3D model coordinates. Note that the 3D model is very coarse (walls and door jambs only) and sometimes inconsistent with the true building. However, our random consensus algorithm overcomes this issue.

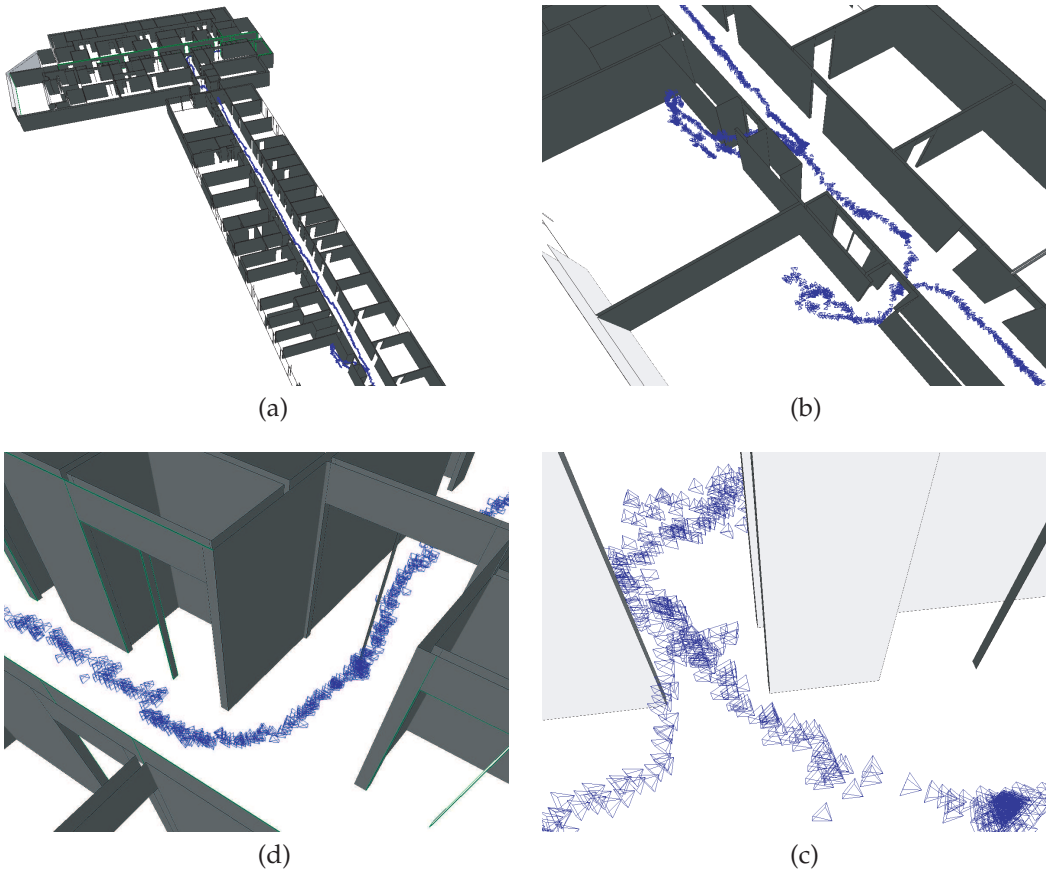

Figure 8-16: Recovered egomotion for the CORRIDOR dataset (7,900 frames, 300 m at 0.20 m/s). The camera was attached to a robot and followed a flat 2D motion. (a) Overview of the excursion; (b) Camera entering two classrooms; (c) Detail view of a classroom entrance; (d) Camera turning at a corner in building 26.

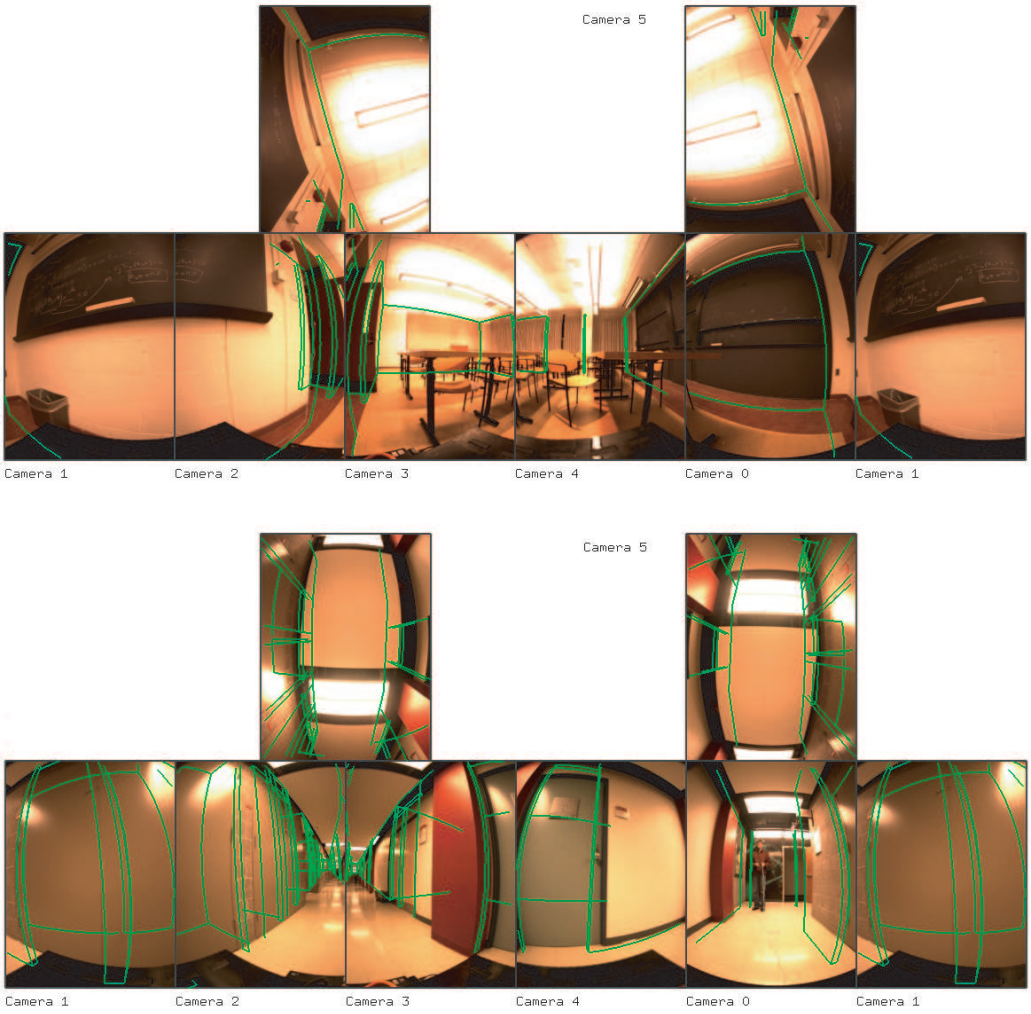

Figure 8-17: CORRIDOR dataset; 3D model re-projected on the video.

8.7 An Application to 3D Model Texture Painting

We applied our system to 3D model texture painting. This technique consists of back-projecting pixel colors from the omnidirectional image onto the 3D model to incorporate dense color information into the model.

Our method includes a basic ray-casting algorithm which computes the intersection between a ray emitted from the camera and the faces of the 3D model. Our algorithm works by computing the intersection of this ray with all visible faces and retaining the intersection with the smallest depth. The off-line visibility computa-

tion bounds the number of faces to consider and avoids the need for an optimized ray-caster.

Once a pixel has been projected onto the model, its RGB color is applied to the model face. In order to support multiple levels of detail, each model face is subdivided into quadrilaterals. The subdivision algorithm proceeds by iteratively dividing each face edge into two equal segments and generating the corresponding sub-faces. The algorithm stops when the maximum dimension of each sub-face is smaller than a given threshold. The subdivision level corresponds to the level of detail specified by the user. In practice, the subdivision level n corresponds to surface patches of size $50/2^n$ inches. For example, level 4 corresponds to 3x3 inch square patches. Figure 8-18 illustrates our subdivision scheme.

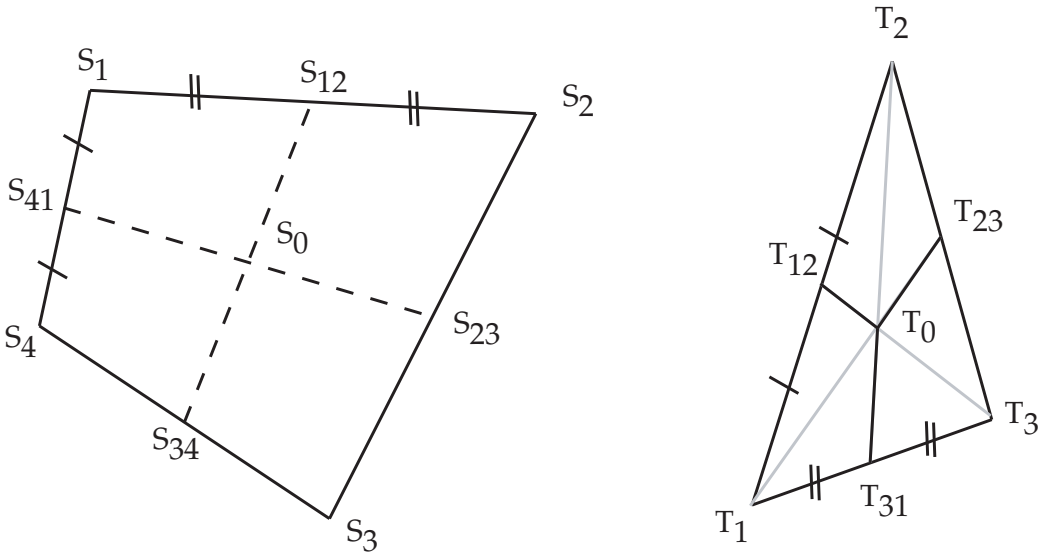

Figure 8-18: Face subdivision scheme for the texture painting algorithm. Left: a quadrilateral $S_1S_2S_3S_4$; each edge is divided into two equal segments defining four new quadrilaterals $S_1S_{12}S_0S_{34}$, ... Right: a triangle $T_1T_2T_3$ is divided into three quadrilaterals in a similar fashion.

Figure 8-19 shows the results of the 3D model texture painting component with various levels of detail. Note the accurate re-projection of objects on the wall.

8.8 Accuracy Analysis

Figure 8-20 shows the localization error (translation and rotation) for the synthetic dataset. We simulate image noise by convolving image edges with Gaussian noise ($\mu = 0, \sigma = .5\,\text{deg}$). Clutter is simulated by removing 25% of the correspondences and adding Gaussian noise of ($\mu = 0, \sigma = 2\,\text{deg}$) to the remaining image edges.

Figure 8-21 shows the distribution of re-projection error for the three data sets (in degrees). Figure 8-22 shows a closeup view of the effect of clutter on localization.

8.9 System Performance

The system currently runs at 1Hz on a four 2GHz CPU standard desktop. Two thirds of the processing time are spent in edge detection and color processing (six 512x384 8-bit images). The remaining is spent in the random sample algorithm. The initialization phase takes one minute given a 2.5 meter uncertainty seed; however, we have implemented an optimized initialization algorithm for the special case of a vertical pose which runs in about 10 seconds. In the vertical case, there are only four degrees of freedom: one for rotation and three for translation. Our algorithm sweeps through a regular grid of 10x10x10 positions in the search region, and tries 10 regularly-spaced rotation angles between 0 and 2π at each position. The algorithm returns the position with the lowest ξ value.

8.10 Discussion

We have demonstrated our system on three real datasets involving both long sequence 4-DOF and 6-DOF motions. The result is a drift-free, accurate localization of the camera in the model coordinates. The algorithm handles clutter and dynamic scenes successfully. However, the algorithm failed on outdoor sequences. We believe that there are several reasons that explain this failure. First, outdoor sequences contain an outstanding amount of clutter (trees, in particular) and unmodeled outdoor structures (e.g. street lights, sidewalk edges). Second, omnidirectional images tend to advantage

short-range features. Therefore, long-range 3D edges (more than 20 meters away) are not accurately detectable on the image. And third, the presence of bright sun light sometimes generates optical artifacts that deteriorate the observations.

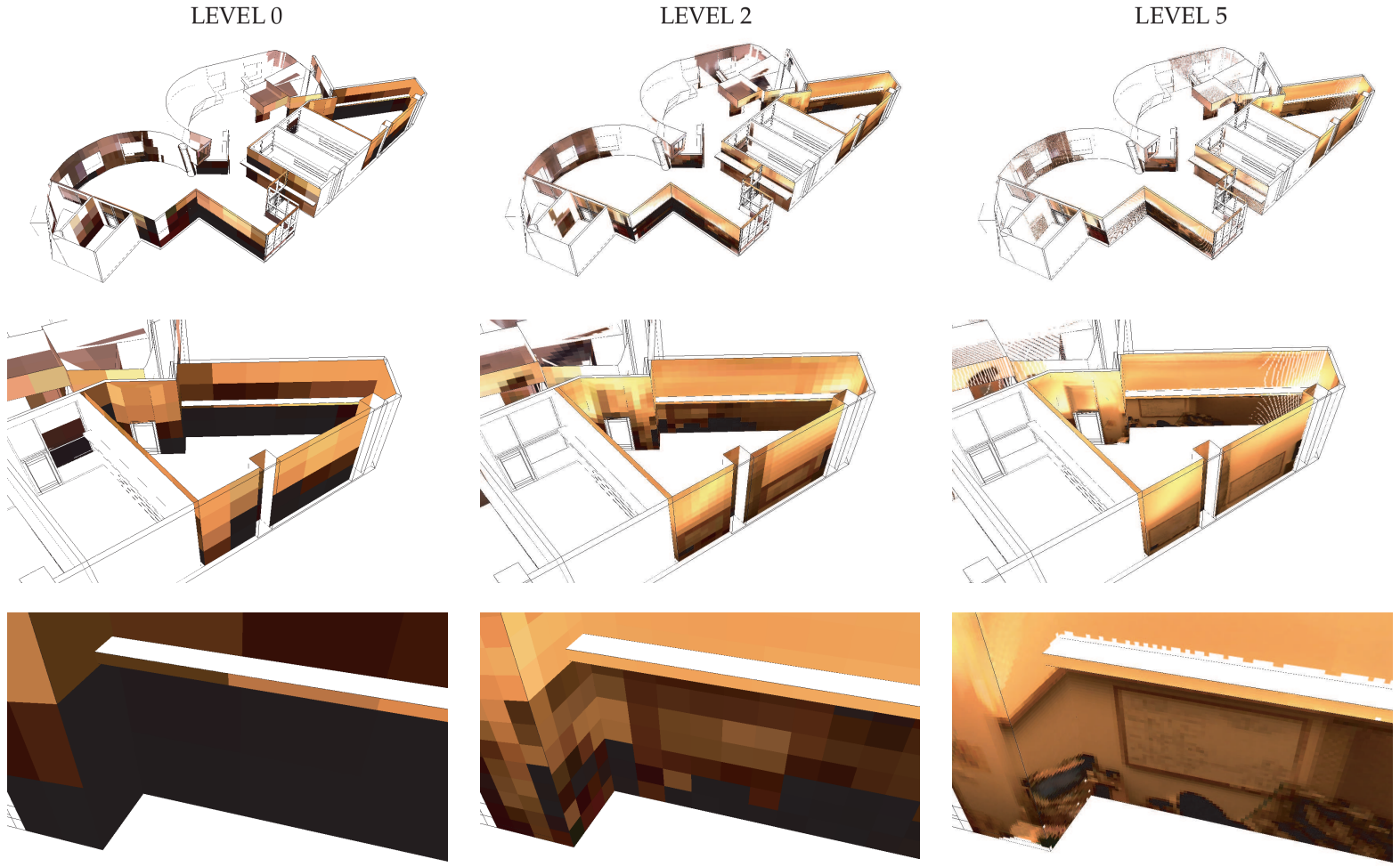

Figure 8-19: 3D model texture painting for various levels of detail. Note the accuracy of details at level 5. Un-modeled 3D objects and localization inaccuracy generate artifacts in the results.

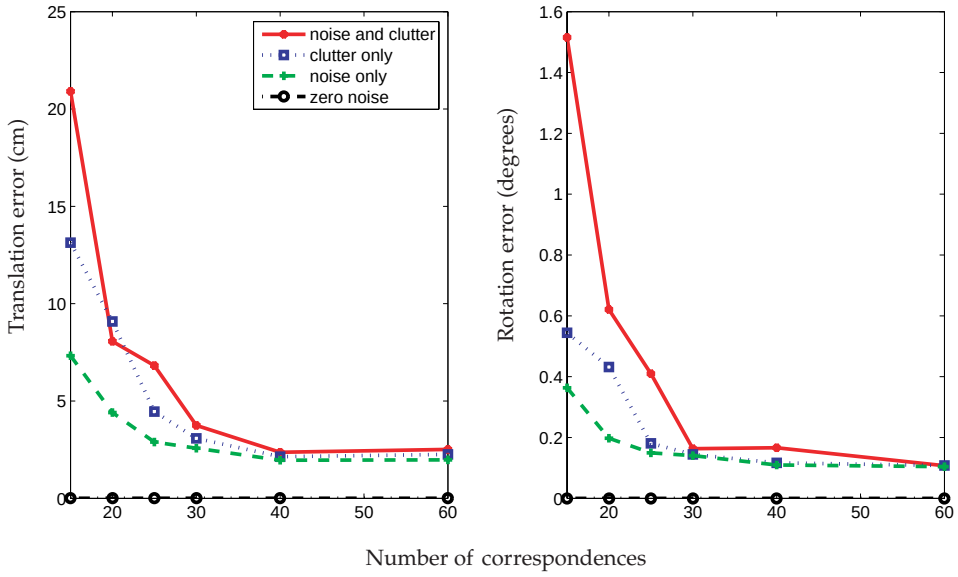

Figure 8-20: Localization accuracy with respect to the number of correspondences. The data was simulated using a Gaussian noise model for image edges. Accuracy plateaus at about 40 correspondences.

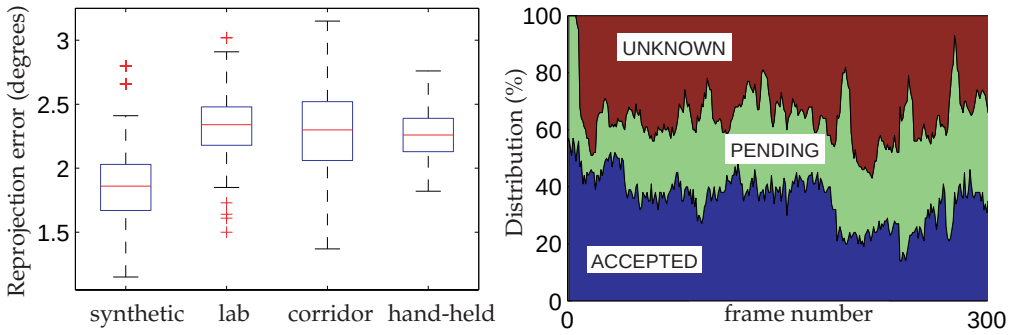

Figure 8-21: Left: re-projection error distribution for each dataset; plus signs indicate the presence of outliers in the data. Right: correspondence state distribution for the LAB dataset. The pool of *accepted* correspondences remains steady despite large changes in visibility across the sequence.

Low clutter

High clutter

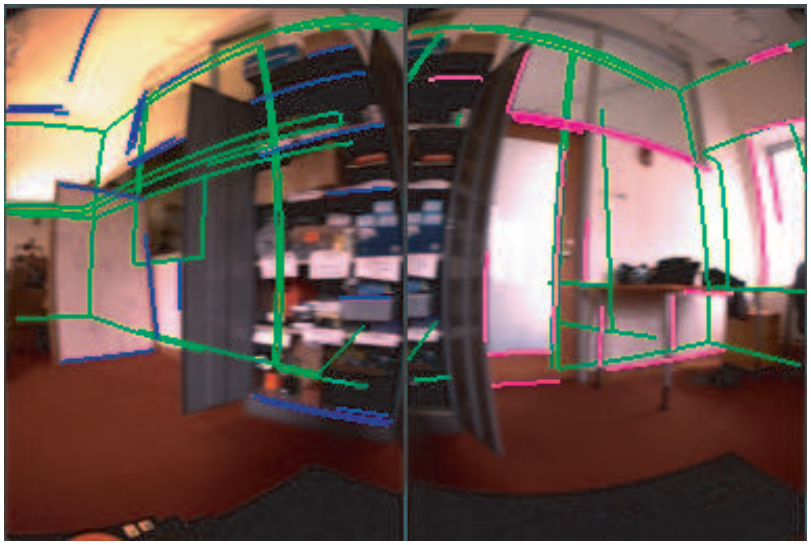

Figure 8-22: The hue constraint helps the system avoid mismatches. However, incorrect correspondences may occur when clutter predominates (bottom). Image edges are color-coded by omnidirectional image tile. Model structure is overlaid in green.

Chapter 9

Conclusion and Future Work

This section discusses several limitations of the current method, and possible directions for its future development.

9.1 System Characteristics

9.1.1 Successes

We described an algorithm for 6-DOF localization from a coarse 3D model and an omnidirectional video sequence. Our system makes few assumptions about the environment other than that it contains prominent straight edges. Our solution algorithm employed two phases, initialization and maintenance, and precomputed visibility analysis to drastically decrease running time and increase the scale of environments that can be handled by the method. We demonstrated the system, and evaluated its performance, on a variety of spatially extended, visually cluttered datasets.

Our system handles large, cluttered environments. It initializes and tracks the camera pose with an accuracy of a few centimeters in translation and a few degrees in rotation. It is robust to unexpected scene changes and is capable of remaining "locked" for more than half an hour at reasonable walking speed (0.5 m/s).

9.1.2 Limitations

The system suffers from the following limitations. First, the method's performance must be improved. This could be achieved through more focused sampling, through code optimization, or with faster hardware. Second, the system's localization accuracy could be higher. Some error is surely due to feature localization; another error source is inaccuracy in the 3D model itself. Third, the current sensor is not light-sensitive enough; it requires slow motion in order to avoid motion blur in indoor environments. Fourth, the initialization method can give ambiguous results in the presence of repeated environment structures such as long corridors. Finally, the visibility pre-computation assumes that a one-meter grid is fine enough to capture most variations in visibility.

9.2 Future Directions

We are currently following several promising directions. First, a geometric signature-based initialization would enable the method to quickly eliminate inconsistent locations and cut down the number of regions in which to run the initialization algorithm. Second, integration of an inertial sensor and a camera motion model would increase the robustness of the maintenance phase. Third, we will investigate tracking of points in addition to line segments. Fourth, visibility pre-processing with varying spatial resolution may increase localization robustness in regions of high visibility variance. Finally, an online update of the model combined with occlusion processing would further decrease the occurrence of false matches.

Bibliography

[1] Olaf Abela. The "CAD-reference-model": An architectural approach to future cad-systems. In *CAD Systems Development*, pages 3–19, 1995.

[2] A. Ansar and K. Daniilidis. Linear pose estimation from points or lines. *ECCV, edited by A. Heyden et al., Copenhagen, Denmark*, 4:282–296, May 2002.

[3] Matthew Antone. An ellipsoidal lens model for fisheye calibration. *Technical Report TR-1940, BAE Systems Advanced Information Technologies, Computer Vision Group*, November 2005.

[4] Matthew Antone and Seth Teller. Scalable extrinsic calibration of omni-directional image networks. *Int. J. Comput. Vision*, 49(2-3):143–174, 2002.

[5] Adrien Bartoli and Peter Sturm. Structure from motion using lines: Representation, triangulation and bundle adjustment. *Computer Vision and Image Understanding*, 100(3):416–441, December 2005.

[6] Bruce G. Baumgart. Winged-edge polyhedron representation for computer vision. In *National Computer Conference*, May 1975.

[7] P. A. Beardsley, A. Zisserman, and D. W. Murray. Sequential updating of projective and affine structure from motion. *Int. J. Comput. Vision*, 23(3):235–259, 1997.

[8] Michael Bosse, Paul M. Newman, John J. Leonard, Martin Soika, Wendelin Feiten, and Seth J. Teller. An Atlas framework for scalable mapping. In *ICRA*, pages 1899–1906, 2003.

[9] Michael Bosse, Richard J. Rikoski, John J. Leonard, and Seth J. Teller. Vanishing points and three-dimensional lines from omni-directional video. *The Visual Computer*, 19(6):417–430, 2003.

[10] George Tao-Shun Chou. Large-scale 3D reconstruction: A triangulation-based approach, MIT PhD thesis. 2000.

[11] S. Coorg and S. Teller. Matching and pose refinement with camera pose estimates. In *IUW 97*, pages 857–862, 1997.

[12] M. Dhome, M. Richetin, and J.-T. Lapreste. Determination of the attitude of 3D objects from a single perspective view. *IEEE Trans. Pattern Anal. Mach. Intell.*, 11(12):1265–1278, 1989.

[13] T. Drummond and R. Cipolla. Real-time visual tracking of complex structures. *IEEE Trans. Pattern Anal. Mach. Intell.*, 24(7):932–946, 2002.

[14] Olivier Faugeras, Quang-Tuan Luong, and T. Papadopoulou. *The Geometry of Multiple Images: The Laws That Govern The Formation of Images of A Scene and Some of Their Applications.* MIT Press, Cambridge, MA, USA, 2001.

[15] Martin A. Fischler and Robert C. Bolles. Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography. *Commun. ACM*, 24(6):381–395, 1981.

[16] Joshua Gluckman and Shree K. Nayar. Ego-motion and omnidirectional cameras. In *ICCV*, pages 999–1005, 1998.

[17] OpenInventor Architecture Group. *Open Inventor C++ Reference Manual: The Official Reference Document for Open Inventor, Release 2.* Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1993.

[18] Robert M. Haralick, Chung-Nan Lee, Karsten Ottenberg, and Michael Nolle. Review and analysis of solutions of the three point perspective pose estimation problem. *Int. J. Comput. Vision*, 13(3):331–356, 1994.

[19] R. I. Hartley and A. Zisserman. *Multiple View Geometry in Computer Vision*. Cambridge University Press, ISBN: 0521540518, second edition, 2004.

[20] Berthold Klaus Paul Horn. *Robot vision*. MIT Press, Cambridge, MA, USA, 1986.

[21] B.K.P. Horn. Closed form solutions of absolute orientation using unit quaternions. *Journal of the Optical Society of America*, 4(4):629–642, April 1987.

[22] Stephen Hsu, Supun Samarasekera, Rakesh Kumar, and Harpreet S. Sawhney. Pose estimation, model refinement, and enhanced visualization using video. In *CVPR*, pages 1488–1495, 2000.

[23] Rakesh Kumar and Allen R. Hanson. Robust methods for estimating pose and a sensitivity analysis. *CVGIP: Image Underst.*, 60(3):313–342, 1994.

[24] David G. Lowe. Distinctive image features from scale-invariant keypoints. *International Journal of Computer Vision*, 60(2):91–110, 2004.

[25] Chien-Ping Lu, Gregory D. Hager, and Eric Mjolsness. Fast and globally convergent pose estimation from video images. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 22(6):610–622, 2000.

[26] John Mallon and Paul F. Whelan. Precise radial un-distortion of images. In *ICPR '04: Proceedings of the Pattern Recognition, 17th International Conference on (ICPR'04) Volume 1*, pages 18–21, Washington, DC, USA, 2004. IEEE Computer Society.

[27] D. Nister, O. Naroditsky, and J. Bergen. Visual odometry. In *Proc. IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR 2004)*, pages 652–659, 2004.

[28] David Nister. Preemptive RANSAC for live structure and motion estimation. In *ICCV '03: Proceedings of the Ninth IEEE International Conference on Computer Vision*, page 199, Washington, DC, USA, 2003. IEEE Computer Society.

[29] David Nister. Automatic passive recovery of 3D from images and video. In *3DPVT '04: Proceedings of the 3D Data Processing, Visualization, and Transmission, 2nd International Symposium on (3DPVT'04)*, pages 438–445, Washington, DC, USA, 2004. IEEE Computer Society.

[30] David Nister. Untwisting a projective reconstruction. *Int. J. Comput. Vision*, 60(2):165–183, 2004.

[31] Thai Quynh Phong, Radu Horaud, Adnan Yassine, and Pham Dinh Tao. Object pose from 2-d to 3-d point and line correspondences. *Int. J. Comput. Vision*, 15(3):225–243, 1995.

[32] Long Quan and Zhong-Dan Lan. Linear n-point camera pose determination. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 21(8):774–780, 1999.

[33] E. Rosten and T. Drummond. Fusing points and lines for high performance tracking. *IEEE International Conference on Computer Vision*, 2:1508–1515, 2005.

[34] Edward Rosten and Tom Drummond. Machine learning for high-speed corner detection. In *European Conference on Computer Vision*, May 2006.

[35] S. Se, David Lowe, , and J. Little. Global localization using distinctive visual features. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 226–231, Lausanne, Switzerland, October 2002.

[36] Ken Shoemake. Animating rotation with quaternion curves. In *SIGGRAPH '85: Proceedings of the 12th annual conference on Computer graphics and interactive techniques*, pages 245–254, New York, NY, USA, 1985. ACM Press.

[37] H. Surmann, A. Nuchter, and J. Hertzberg. An autonomous mobile robot with a 3D laser range finder for 3D exploration and digitalization of indoor environments. *Robotics and Autonomous Systems*, 45:181–198, December 2003.

[38] Rahul Swaminathan and Shree K. Nayar. Nonmetric calibration of wide-angle lenses and polycameras. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 22(10):1172–1178, 2000.

[39] Camillo J. Taylor and David J. Kriegman. Structure and motion from line segments in multiple images. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 17(11):1021–1032, 1995.

[40] Luca Vacchetti and Vincent Lepetit. Stable real-time 3D tracking using online and offline information. *IEEE Trans. Pattern Anal. Mach. Intell.*, 26(10):1385–1391, 2004.

[41] Mason Woo, Jackie Neider, and Tom David. *OpenGL 1.2 Programming Guide, 3rd Edition: The Official Guide to learning OpenGL, Version 1.2.* Addison Wesley, 1999.

[42] Y. Yagi, Y. Nishizawa, and M. Yachida. Map-based navigation for a mobile robot with omnidirectional image sensor COPIS. In *IEEE Trans. Robotics and Automation*, pages 634–648, October 1995.

[43] Wenyi Zhao, David Nister, and Steve Hsu. Alignment of continuous video onto 3D point clouds. *IEEE Trans. Pattern Anal. Mach. Intell.*, 27(8):1305–1318, 2005.

Appendix A

A Note on 3D Models

A.1 Model Assumptions

In our model, we assume that a coarse 3D model of the environment is provided. To make this assumption as easy to satisfy as possible, we assume that only basic and textureless geometry is provided (vertices, faces, edges). Our system assumes that faces have either three of four vertices. No detailed geometry (such as doors, staircase, or ceiling) is required. The model is not required to have a conventional watertightness property. However, counter-clockwise face orientation is required for face visibility.

However, the model is expected to include all major physical edges of the building in an accurate manner. Obviously, the more accurate the model is, the better the system performs.

A.2 Model Representation

The 3D model can be provided in various file formats such as AutoCAD format [1] or Inventor [17]. Regardless of the input format, the model is stored in the system as a list of vertices, faces and edges where each vertex points to all incident edges, each edge points to its two end vertices and its neighbor faces and each face points to its vertices. This data structure is very similar to the classic "winged-edge" data

structure by Baumgart [6]. Each geometric object is assigned a unique integer ID. Figure A-1 illustrates this data structure.

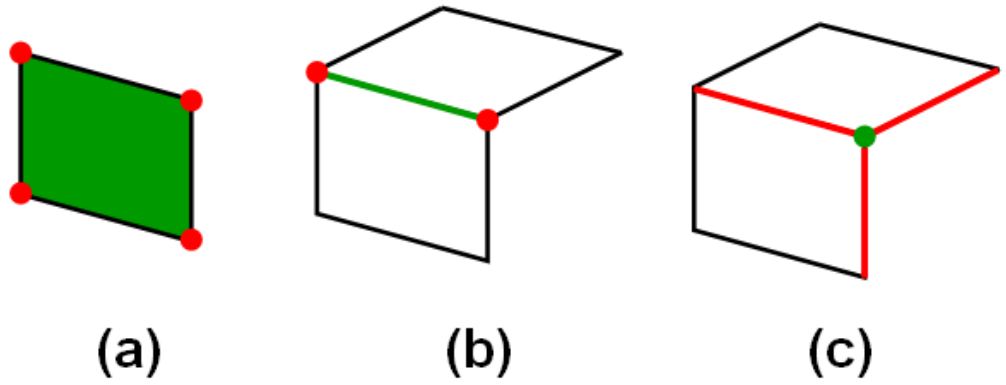

Figure A-1: Model data structure. (a) Faces point to vertices; (b) edges point to vertices and adjacent faces; (c) vertices point to adjacent edges.

A.3 Model Generation

Although 3D models are sometimes provided for modern buildings, there is a need for generating 3D models of existing buildings or refining as-planned models into as-built models. This task can be accomplished manually using a laser ranger finder and modeling software such as AutoCAD. This method allows scanning a complete building in a couple of months and remain today the most reliable technique[1]. However, some automated methods have appeared such as the Building Model Generation (BMG) Project which extrudes 3D models from 2D floor plans and a few adjustable parameters. Also some recent research tackles the difficult problem of generating 3D models from mobile robot observations [37].

[1] As an example, the 3D model of the Stata Center Robotics, Vision, and Sensor Networks Lab used in this project was manually generated in three days.

Appendix B

Representing Rotations

A rotation in $\Re^3$ has three degrees of freedom and is uniquely defined by a vector along the rotation axis u and a rotation angle θ. This representation is called the *axis-angle* representation. Unlike the *rotation matrix* representation, it has no redundancy and provides an intuitive way of representing rotations. However, it is not a convenient representation for computation.

The *Euler angles* representation consists in decomposing the rotation into three rotations R_x, R_y and R_z along the three basis axes X, Y and Z. The three rotation matrices are given by :

$$R_x(\alpha) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & cos(\alpha) & sin(\alpha) \\ 0 & -sin(\alpha) & cos(\alpha) \end{bmatrix} \tag{B.1}$$

$$R_y(\beta) = \begin{bmatrix} cos(\beta) & 0 & -sin(\beta) \\ 0 & 1 & 0 \\ sin(\beta) & 0 & cos(\beta) \end{bmatrix} \tag{B.2}$$

$$R_z(\gamma) = \begin{bmatrix} cos(\gamma) & sin(\gamma) & 0 \\ -sin(\gamma) & cos(\gamma) & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{B.3}$$

This representation is often used in linear algebra algorithms but it suffers from a

high level of redundancy since a rotation matrix has 9 parameters. Therefore, extra care has to be taken to ensure that the matrix obtained at the end of the algorithm is indeed a rotation matrix.

In 1843, William R. Hamilton invented a new group called *quaternions*. A *quaternion* is a 4D-vector and is defined by analogy with complex numbers with a real part and an imaginary part, except that a quaternion imaginary part has three dimensions:

$$q = a \cdot 1 + b \cdot i + c \cdot j + d \cdot k \tag{B.4}$$

The multiplication between quaternions is defined as:

$$q_2.q_1 = (s_2, v_2) \cdot (s_1, v_1) = (s_1 s_2 - v_2 \cdot v_1, s_1 v_2 + s_2 v_1 + v_2 \times v_1). \tag{B.5}$$

Unit quaternions provide a powerful representation for rotations. Indeed, if an axis-angle rotation (u, θ) is represented by the unit quaternion:

$$q = (cos(\theta/2), sin(\theta/2) \cdot u) \tag{B.6}$$

then the rotation can be easily applied to any vector $p \in \Re^3$:

$$p = q \cdot p \cdot \hat{q} \tag{B.7}$$

where $\hat{q}$ is the conjugate of q:

$$\hat{q} = q_1 - q_2 i - q_3 j - q_4 k. \tag{B.8}$$

More about quaternions is available in [36].

Appendix C

Alternative Initialization Methods

In addition to the probabilistic line-edge matching algorithm and the rotation voting algorithm, we present here two alternative initialization methods presenting complementary failure modes.

C.1 Initialization from Vanishing Point Alignment

Vanishing points are relevant features in omnidirectional systems since a single image is likely to observe vanishing points in various directions [9]. Aligning the vanishing points with the expected vanishing points computed from the 3D model enables the determination of candidate camera rotations in an efficient manner.

Given our representation of image edges, image vanishing points can be obtained by simply computing the cross product of the normal vectors for every pair of skew image edges. Each of these cross products corresponds to a candidate vanishing point. The true image vanishing points are then obtained by examining aggregates of such candidates around a common point.

However, given a set of model lines and an estimate of the camera position, it is possible to compute the set of vanishing points that the camera would expect to see. The process is the same as computing the image vanishing points, except that image edges are now replaced by virtual image edges represented by the camera center and the line end points.

Once vanishing points have been computed for the model and the image, the algorithm considers every pair of image vanishing points $\{vp_1^{edge}, vp_2^{edge}\}$ and every pair of model vanishing points $\{vp_1^{model}, vp_2^{model}\}$ and computes the rotation that brings the first pair into alignment with the second. Here again, the rotation is uniquely defined up to a symmetry if the camera is not allowed to translate.

Once a set of candidate rotations has been obtained, the system proceeds as in the rotation voting algorithm and determines the best camera pose possible. Figure C.1 summarizes the algorithm.

1: Inputs: an omnidirectional image; a set of expected model lines; an estimate of the camera position.
2: Compute the image vanishing points from image edges.
3: Compute the model vanishing points from model lines.
4: **for** each pair of image vanishing points $\{vp_1^{edge}, vp_2^{edge}\}$ **do**
5: **for** each pair of model vanishing points $\{vp_1^{model}, vp_2^{model}\}$ **do**
6: Check whether the angle between vp_1^{edge} and vp_2^{edge} is equal to the angle between vp_1^{model} and vp_2^{model} up to some threshold error (typically five degrees)
7: If no, continue.
8: Otherwise, compute the rotation that brings $\{vp_1^{edge}, vp_2^{edge}\}$ into alignment with $\{vp_1^{model}, vp_2^{model}\}$
9: Score the camera pose.
10: Return the best camera pose found so far.

Table C.1: The Vanishing Point Algorithm for Initialization

C.2 Initialization from Corner Alignment

In our corner-based initialization, corners on the image are matched with corners in the 3D model. An image corner is defined as an image point and its two attached image edges. A model corner is defined as a model vertex and its two attached model edges. A correspondence between two image corners and two model corners yields a unique solution for the camera pose. Therefore, by considering the set of all possible model corners and the set of all possible image corners, the algorithm need only find two correct correspondences to find the camera pose. In practice, the number of corners is small enough to allow for an exhaustive search.

Once an estimate of the camera position is provided, model corners are obtained in a straightforward manner by considering all visible vertices and all visible edges attached to them. Image corners are obtained by computing images edges and looking for points on the image where two edges meet. The neighbor window used for finding neighbor end points is typically a 5x5 pixel square. FAST features [33, 34] are incorporated in the detector to improve corner detection.

For each pair of image corners and each pair of model corners, the algorithm computes the corresponding camera position and scores it. The pose with the highest score is then retained. Figure C.2 summarizes the corner matching algorithm. The complexity of the algorithm is p^2q^2 for p image corners and q model corners. Therefore, this algorithm runs only on relatively small numbers of corners.

1: Compute the image corners.
2: Compute the model corners.
3: **for** each pair of image corner $\{c_1^{edge}, c_2^{edge}\}$ **do**
4: **for** each pair of model corner $\{c_1^{model}, c_2^{model}\}$ **do**
5: Compute the camera pose assuming that the image corners $\{c_1^{edge}, c_2^{edge}\}$ correspond to the image corners $\{c_1^{model}, c_2^{model}\}$
6: Compute the score.
7: Return the best camera pose found so far.

Table C.2: The Corner Matching Algorithm for Initialization

Appendix D

Ladybug Calibration Parameters

In this thesis, the Ladybug 1 has been used (ID 5040012). We present here the actual calibration parameters of the camera.

D.1 Intrinsic Parameters

The intrinsic calibration parameters have been recovered thanks to a line-based calibration tool developed by Matt Antone. Each sensor is characterized by the image width and height in pixels (w_{pixels}, h_{pixels}), the single radial distortion parameter (α), the focal length in centimeters (f_{cm}), the center point expressed in the normalized image coordinate frame (c_x, c_y), and the pixel width and height in millimeters (pw_{mm}, ph_{mm}). In normalized coordinates, the image center is $(0, 0)$ and axis values vary between -0.5 and 0.5. Table D.1 presents the recovered values for each of these parameters for each sensor.

	w_{pixels}	h_{pixels}	α	f_{cm}	c_x	c_y	pw_{mm}	ph_{mm}
Sensor 0	512	384	1.2573	.8016	.0190	-.0239	.666	.666
Sensor 1	512	384	1.2469	.7917	.0204	-.0260	.666	.666
Sensor 2	512	384	1.2513	.7627	.0157	-.0200	.666	.666
Sensor 3	512	384	1.2405	.7677	.0372	-.0220	.666	.666
Sensor 4	512	384	1.2613	.7424	.0262	-.0220	.666	.666
Sensor 5	512	384	1.2566	.7398	.0367	.0193	.666	.666

Table D.1: Ladybug 5040012 Intrinsic Calibration Parameters

D.2 Extrinsic Parameters

The calibration file provided by PointGrey (*ladybug5040012.cal*) contains the rigid-body transformation between each sensor inside the Ladybug and the virtual camera frame. Each sensor is characterized by a $4x4$ transformation matrix $P_{i,0\leq i\leq 5}$ which allows us to transform a point X_i expressed in the i-th homogeneous sensor coordinate into a point $\tilde{X}$ expressed in homogeneous virtual camera coordinates by :

$$\tilde{X} = P_i X_i \tag{D.1}$$

Table D.2 presents the actual values for each transformation matrix. Note that the matrices are not normalized. The PointGrey calibration file also contains a large lookup table for pixel rectification for each sensor. However, we found these tables to be less accurate than our own intrinsic calibration and we did not use them.

$$P_0 = \begin{bmatrix} 0.006188 & -0.000413 & 0.999981 & 0.032996 \\ 0.001264 & 0.999999 & 0.000405 & -0.000151 \\ -0.999980 & 0.001262 & 0.006189 & -0.000169 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$P_1 = \begin{bmatrix} 0.007559 & 0.951768 & 0.306727 & 0.010206 \\ -0.007162 & 0.306779 & -0.951754 & -0.031316 \\ -0.999946 & 0.004997 & 0.009136 & -0.000181 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$P_2 = \begin{bmatrix} 0.005651 & 0.585441 & -0.810695 & -0.027375 \\ -0.008360 & -0.810652 & -0.585469 & -0.019643 \\ -0.999949 & 0.010086 & 0.000313 & 0.000178 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$P_3 = \begin{bmatrix} -0.009852 & -0.590630 & -0.806882 & -0.026548 \\ 0.003887 & -0.806938 & 0.590623 & 0.019349 \\ -0.999944 & 0.002682 & 0.010246 & -0.000394 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$P_4 = \begin{bmatrix} -0.009088 & -0.949388 & 0.313975 & 0.010720 \\ 0.000952 & 0.313980 & 0.949429 & 0.031761 \\ -0.999958 & 0.008928 & -0.001950 & 0.000566 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$P_5 = \begin{bmatrix} 1.000000 & 0.000412 & 0.000459 & -0.000102 \\ -0.000413 & 0.999999 & 0.001262 & 0.000008 \\ -0.000458 & -0.001262 & 0.999999 & 0.033660 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Table D.2: Ladybug 5040012 Extrinsic Calibration Parameters

Appendix E

Development Instructions

This project requires Windows XP and Microsoft Visual C++ 6.0. Note that the code has not been tested on later versions of Microsoft Visual C++ (such as 7.0 or .NET).

E.1 Location and Organization of Source

The source code is stored under SVN (Subversion) in the RVSN site. To checkout the source, type the following command in the C:/ directory (all one line):

```
svn co
svn+ssh:\\svn.csail.mit.edu/afs/csail.mit.edu/group/rvsn/repositories/omni3d
```

The result should be a directory *omni3d* containing the following sub-directories:

- config: calibration files;
- data: datasets (see appendix F);
- docs: code documentation;
- fltk-1.1.7: source code for the FLTK library;
- models: 3D models (see appendix F);

- src: source code;
- vision_atlas: source code by Mike Bosse (useful for edge detection only).

The source code directory (*src*) is organized as described in Table E.1.

Directory name	Content
Low-level toolkits	
geom	basic geometric components
ladybug	Ladybug specific code
math	basic math operations
scripts	Perl scripts for Ladybug image handling
timer	performance timer
util	utilities such as file handling
viewer	OpenGL code
Image processing	
camera	projection, rectification
database	image dataset management
fast	FAST feature detection and tracking
Model processing	
model	3D model handling
signature	visibility computation
viscomp	separate code and user interface for batch visibility computation
High-level operations	
framework	user interface
include	generic include files
lib	output static libraries

Table E.1: Source code directories

E.2 Build Instructions

The build instructions are specified in the BUILD file located at the top of the source code tree. Note that the code relies on the following third-party software which needs to be installed prior to building the application:

- ImageMagick (http://www.imagemagick.org);
- OpenCV (http://sourceforge.net/projects/opencvlibrary/);

- the VL Vector library by Andrew Willmott, Graphics Group, SCS, CMU (http://www.cs.cmu.edu/ ajw/software/index.html);
- the PointGrey Ladybug API (http://www.ptgrey.com).

These applications are available for free on the Internet (except for the PointGrey Ladybug API which can be retrieved by contacting the company directly).

E.3 Dataset Import

In order to run the application on a sample dataset, the dataset must be imported into the *data* directory in the source code directory. Several datasets are available on the RVSN site (see appendix F).

E.4 Execution Instructions

Dataset and Model Setup

Once the **framework** project has been compiled and a dataset has been imported, run the application by pressing F5 in Microsoft Visual C++ 6.0. A user interface appears. Figure E-1 shows screenshots of the application. Select *File* → *Open Database*. Select the desired dataset directory and press OK. The first frame of the dataset is displayed.

Open the *data.ini* file of the dataset to determine which 3D model it corresponds to. Select *File* → *Open 3D Model*. Browse to the corresponding 3D model, select the *config.ini* file and press OK. The 3D model appears. If the output pose file *poses.data* is present in the dataset, the set of camera poses listed in the file is shown in blue in the 3D model. You may move the viewpoint by dragging the mouse with the left button pressed (rotation), the middle button pressed (translation) or the right button pressed (scale).

On the control bar on the right, press *View Grid* to toggle the node display on and off. Nodes are displayed in blue. By pressing the left button of the mouse while holding the SHIFT key down, you may specify an estimate of the camera position for

the first frame. The camera position appears in yellow. The closest node turns red. The set of visible model lines appears in green.

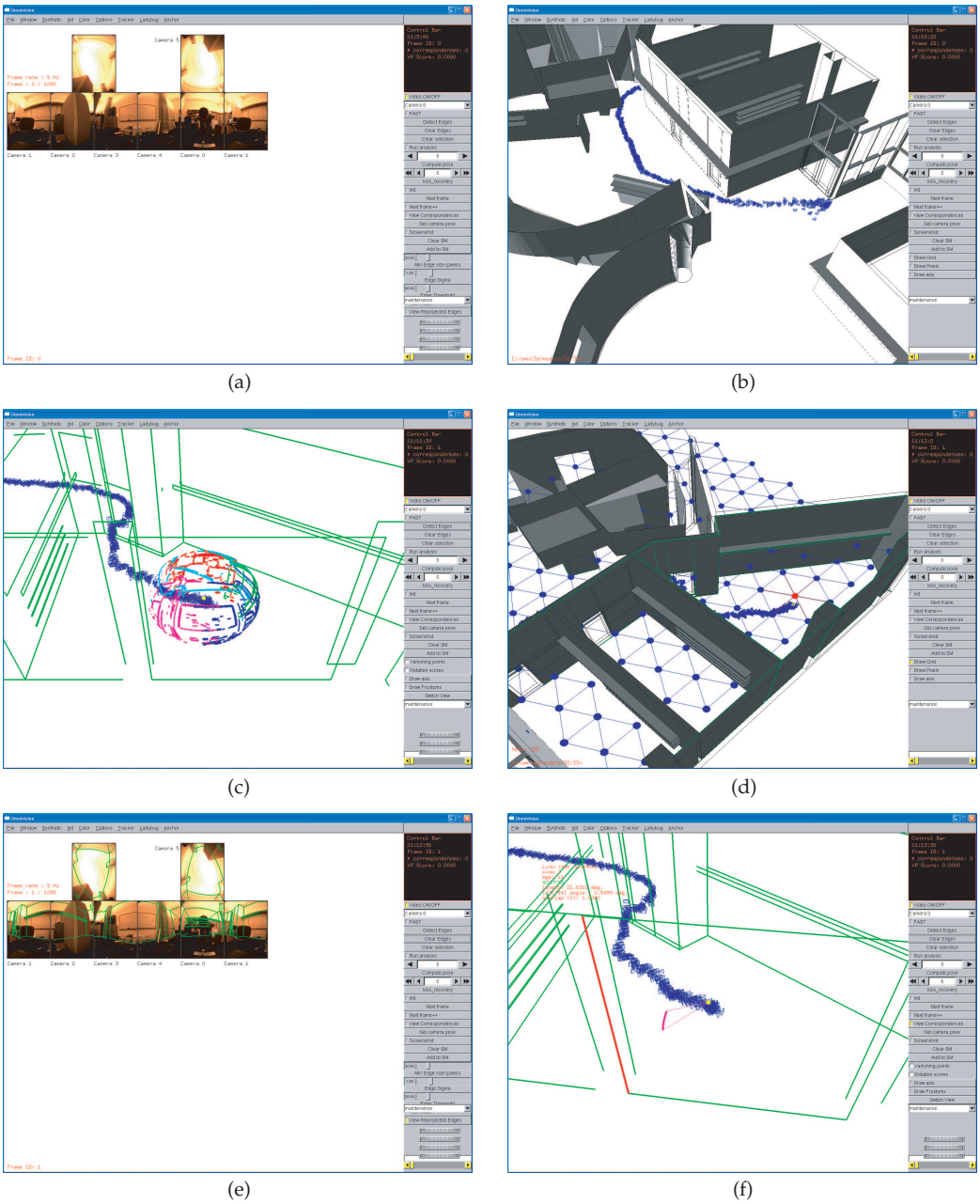

Figure E-1: Screenshots of the software application. (a) after opening a database; (b) after opening the model (the camera motion is shown in blue); (c) in *Sphere* mode after running edge detection; (d) 3D map with grid view enabled. The closest node is shown in red; the visible model edges are shown in green; (e) re-projection of the structure after initialization; (f) example of a correspondence after initialization.

User Interface Modes

The user interface has four modes: Video, 3D Model, Sphere and Calibration. You may switch modes using the *Window* tab.

Initialization

To run the initialization, move the camera position estimate close to the recovered camera position for the first frame in the 3D model (alternatively, you may try various locations and keep the best result). By selecting *Init*, display the list of algorithms to be used for initialization (line-edge matching, rotation voting, vanishing points, corner matching, exhaustive search, vertical pose). By default, all of them are selected. Keep only the first two for faster initialization. Finally, select *Init* → *Initialization*. After a couple of minutes, the correct camera pose is recovered, which can be tested by two means: first, press *View Reprojected Edges* in Video mode. The model lines are reprojected in green on the video and should match the image. Second, press *Set Camera Pose* and read the displayed camera pose. The pose should be the same as the one written in the poses.dat file (if it exists) for the first frame.

Maintenance

Once the camera pose has been initialized, select *Init* → *Init Correspondences* in order to generate a set of initial correspondences. To view these correspondences, move to the Sphere mode. Turn on the *View Correspondences* button. The first correspondence is highlighted. Press on the left and right arrows in the right control bar to move across the correspondences.

Turn off the *View Correspondences* flag. Move to the 3D model mode. Press the *Next frame* button. The next frame is processed and the camera pose is updated. You can see the camera moving slightly on the 3D model. Turn on the *Next Frame++* button to run the maintenance continuously. Again, the camera pose is updated in the 3D model as the frame number is incremented. Also, the model lines should reproject correctly on the video.

Switching datasets

In order to switch to a new dataset, turn off the *Next Frame++* button. Select *File* → *Exit Database* and open a new database.

Appendix F

Datasets

F.0.1 Image Datasets

The datasets are posted on the RVSN site at the following location:

`/afs/csail.mit.edu/group/rvsn/www/data/static-content/omni3d/data`

and at the following URL:

`http://rvsn.csail.mit.edu/static-content/omni3d/data/`

Each directory corresponds to one dataset. The naming convention for a dataset directory is *yyyymmdd_bbb_name* where *yyyy*, *mm*, *dd* are the year, month and day (respectively) the sequence was captured, *bbb*, the building number at MIT and *name*, a name specific to the dataset (e.g. *robot*).

Each dataset contains the Ladybug images in JPG format, a configuration file (*data.ini*), the output camera pose at every frame as recovered using our method (*poses.dat*) and a bird's eye view of the 3D model and reconstructed camera motion (*birds-eye.jpg*). The datasets are usable as is by pointing to their location when opening a dataset (see section E.4).

Ladybug images are named *xxxxxx*_cam*y*.jpg where *xxxxxx* is the frame ID and y is the camera ID (between 0 and 5).

- `20060506_33x_robot`: 1,098 frames captured at 5 Hz in the Stata Center RVSN Lab. The camera was mounted in vertical position on a robot at a height of 30 inches.

- `20061006_outdoor`: 2,807 frames captured at 5 Hz outside the MIT Media Lab. The camera motion could not be recovered for this dataset (see explanations in section 8.10).

- `20061107_33x`: 1,554 frames captured at 5 Hz in the Stata Center RVSN Lab. The camera was mounted in vertical position on a robot at a height of 30 inches. This is the LAB dataset.

- `20061121_33x_3d`: 1,954 frames captured at 15 Hz in the Stata Center RVSN Lab. The camera was hand held. This is the HAND-HELD dataset.

- `20061121_36-26-16`: 7,890 frames captured at 5 Hz across buildings 26 and 36. The camera was mounted in vertical position on a robot at a height of 30 inches. This is the CORRIDOR dataset.

F.0.2 3D Models

The 3D models are stored in the source code repository in the *model* directory. Models are ordered by building number and floor number. Each directory corresponds to one model. Each model contains:

- the vertices, edges and faces of the model;

- a configuration file (*config.ini*);

- the visibility set of the model.

The detailed content of each file and naming conventions are explained in the README file in the *model* directory of the source code.