Global error reduction in vision-based self-localization using a topological graph representation

Karam Shaya

Follow this and additional works at: https://scholar.uwindsor.ca/etd

Recommended Citation
https://scholar.uwindsor.ca/etd/5344

This online database contains the full-text of PhD dissertations and Masters’ theses of University of Windsor students from 1954 forward. These documents are made available for personal study and research purposes only, in accordance with the Canadian Copyright Act and the Creative Commons license—CC BY-NC-ND (Attribution, Non-Commercial, No Derivative Works). Under this license, works must always be attributed to the copyright holder (original author), cannot be used for any commercial purposes, and may not be altered. Any other use would require the permission of the copyright holder. Students may inquire about withdrawing their dissertation and/or thesis from this database. For additional inquiries, please contact the repository administrator via email (scholarship@uwindsor.ca) or by telephone at 519-253-3000ext. 3208.
GLOBAL ERROR REDUCTION IN VISION-BASED SELF-LOCALIZATION USING A TOPOLOGICAL GRAPH REPRESENTATION

by

KARAM SHAYA

A Thesis
Submitted to the Faculty of Graduate Studies through Electrical and Computer Engineering in Partial Fulfillment of the Requirements for the Degree of Master of Applied Science at the University of Windsor

Windsor, Ontario, Canada
2012

© 2012 Karam Shaya
GLOBAL ERROR REDUCTION IN VISION-BASED SELF-LOCALIZATION USING A TOPOLOGICAL GRAPH REPRESENTATION

by

KARAM SHAYA

APPROVED BY:

____________________________

J. Urbanic
Industrial and Manufacturing Systems Engineering

____________________________

J. Wu
Electrical and Computer Engineering

____________________________

X. Chen, Advisor
Electrical and Computer Engineering

____________________________

S. Erfani, Chair of Defense
Electrical and Computer Engineering

October 9, 2012
Declaration of Co-Authorship / Previous Publication

I. Co-Authorship Declaration

I hereby declare that this thesis incorporates material that is result of joint research, as follows: This thesis also incorporates the outcome of joint research undertaken in collaboration with Aaron Mavrinac and Jose Luis Alarcon Herrera under the supervision of professor Xiang Chen. This collaboration is covered in Chapters 1, 3, 5, and 6 in the thesis. In all cases, the key ideas, primary contributions, experimental designs, data analysis, and interpretation were performed by the author, and the contributions of the co-authors was primarily through the provision of technical content assessment and proof-reading.

I am aware of the University of Windsor Senate Policy on Authorship and I certify that I have properly acknowledged the contribution of other researchers to my thesis, and have obtained written permission from each of the co-author(s) to include the above material(s) in my thesis.

I certify that, with the above qualification, this thesis, and the research to which it refers, is the product of my own work.

II. Declaration of Previous Publication

This thesis includes one original paper that has been previously published/submitted for publication in peer reviewed journals, as follows:
I certify that I have obtained a written permission from the copyright owner(s) to include the above published material(s) in my thesis. I certify that the above material describes work completed during my registration as graduate student at the University of Windsor.

I declare that, to the best of my knowledge, my thesis does not infringe upon anyone’s copyright nor violate any proprietary rights and that any ideas, techniques, quotations, or any other material from the work of other people included in my thesis, published or otherwise, are fully acknowledged in accordance with the standard referencing practices. Furthermore, to the extent that I have included copyrighted material that surpasses the bounds of fair dealing within the meaning of the Canada Copyright Act, I certify that I have obtained a written permission from the copyright owner(s) to include such material(s) in my thesis. I declare that this is a true copy of my thesis, including any final revisions, as approved by my thesis committee and the Graduate Studies office, and that this thesis has not been submitted for a higher degree to any other University or Institution.
Abstract

A single-sensor self-localization system which uses a monocular camera and a set of artificial landmarks is presented herein. The system represents the surrounding environment as a topological map (or graph) where each node corresponds to a marker (i.e., artificial landmark) and each edge corresponds to the existence of a relative pose between two markers. The edges are weighted based on an error metric (related to pose uncertainty) and a shortest path algorithm is applied to the map to compute the path corresponding to the least aggregate error. This path is used to localize the camera with respect to a global coordinate system whose origin lies on an arbitrary reference marker (i.e., the destination node of the path). Experimental results demonstrate the performance of the system in reducing the global error associated with large-scale localization.
Dedication

This work is dedicated to my parents, Adel Shaya and Khalida Sakee.
Acknowledgements

First, I would like to express my sincere gratitude towards my thesis advisor, Dr. Xiang Chen, for his encouragement, guidance, patience, and above all, faith in my abilities to complete my research work. Special thanks to my committee members, Dr. Jonathan Wu and Dr. Jill Urbanic, for enriching the quality of my work through their invaluable suggestions and insight. I want to thank my colleagues in the Advanced Control Systems Laboratory, especially those in the computer vision group: Aaron Mavrinac, Jose Alarcon, Davor Srsen, and Durga Rajan. Their support and understanding during my moments of doubt has meant more to me than I could express. Finally, I am forever indebted to my loving family for everything they have done for me throughout my life.
## Contents

Declaration of Co-Authorship / Previous Publication iii  
Abstract v  
Dedication vi  
Acknowledgements vii  
List of Figures xii  
List of Tables xv  

1 Introduction 1  
1.1 Single-Sensor Vs. Multi-Sensor Systems 1  
1.2 The Camera As a Sensor in Self-Localization Systems 2  
1.3 Local and Global Error 2  
1.4 Existing Systems That Utilize Graphical Representations in Pose Estimation Applications 3  
1.5 Single-Sensor, Vision-Based Self-Localization Systems 3  
1.6 Main Contribution 5  

2 Theoretical Foundation 6  
2.1 Image Formation 6  
2.2 The Pinhole Camera Model 9  
2.2.1 3D Euclidean Transformation 10  
2.2.2 3D-2D Projection 10  
2.2.3 2D-2D Transformation 12
CONTENTS

2.2.4 Combining the Transformations ........................................... 13
2.3 Camera Calibration .......................................................... 13
2.4 Pose ................................................................. 16
  2.4.1 Pose Composition ...................................................... 16
  2.4.2 Relative Pose Calculation ............................................ 17
2.5 Shortest Path Algorithm .................................................... 18

3 Proposed Solution ............................................................. 19
  3.1 Preliminaries .............................................................. 19
    3.1.1 Marker Graph ....................................................... 19
    3.1.2 Localization Graph ................................................ 20
  3.2 Global Localization ....................................................... 20
    3.2.1 Assumptions ....................................................... 20
    3.2.2 Problem Definition ............................................... 20
    3.2.3 Self-Localization Method ....................................... 21
    3.2.4 Self-Localization Algorithm ................................... 23

4 Implementation ............................................................... 25
  4.1 Marker Design .......................................................... 25
    4.1.1 Outer Border ...................................................... 29
    4.1.2 Inner Code ....................................................... 30
  4.2 Software Implementation ............................................... 31
    4.2.1 Internal Camera Calibration .................................... 31
    4.2.2 HDevelop Program .............................................. 32
    4.2.3 Python Program .................................................. 37
  4.3 Hardware Implementation ............................................... 40

5 Experiments ................................................................. 41
  5.1 Purpose ............................................................... 41
  5.2 Experimental Setup .................................................... 41
  5.3 Error Metrics .......................................................... 42
    5.3.1 Reprojection Error Metric ..................................... 42
    5.3.2 Blur Metric ....................................................... 42
## CONTENTS

- 5.3.3 Error Ellipse Volume Metric ............................................. 43
- 5.3.4 Perpendicular Distance Metric .......................................... 44

5.4 Experiment to Compare the Error Metrics ................................. 44
  - 5.4.1 Procedure ................................................................. 45
  - 5.4.2 Results ..................................................................... 47
  - 5.4.3 Analysis .................................................................... 49

5.5 Experiment to Examine the Effect of Implementing an Error Metric on the Global Scale ......................................................... 50
  - 5.5.1 Procedure ................................................................. 50
  - 5.5.2 Results ..................................................................... 53
  - 5.5.3 Analysis .................................................................... 55
  - 5.5.4 Limitations ............................................................... 55

6 Conclusion ............................................................................. 56
  - 6.1 Summary of Contributions .................................................. 56
  - 6.2 Future Work .................................................................... 56

A Glossary of Terms .................................................................... 58

B HDevelop Program Source Code .............................................. 61
  - B.1 Main Program ............................................................... 61
  - B.2 Function: `get_quad_corners` ............................................ 71
  - B.3 Function: `is_quad_inside` ................................................ 75
  - B.4 Function: `calc_cross_ratio` ............................................. 77

C Python Program Source Code .................................................. 78

D Data Tables ............................................................................ 88
  - D.1 Ground Truth Error Vs. Error Metrics ............................... 88
  - D.2 Ground Truth Error Vs. Aggregate Errors of Perpendicular Distance Functions .......................................................... 90

E Letters of Permission to Use Copyrighted Material ..................... 92
  - E.1 Permission to Use Conference Paper ................................. 93
# List of Figures

2.1 Illustration of the process of image formation. . . . . . . . . . . . . . . . . . . . 7  
2.2 Illustration of the pinhole camera model. . . . . . . . . . . . . . . . . . . . . 9  
2.3 Perspective projection of a 3D point $P$ to a 2D point $p$ on an image plane. . 11  
2.4 An example calibration pattern. . . . . . . . . . . . . . . . . . . . . . . . . . . 14  
2.5 Relative pose calculation: The camera can calculate the relative pose transform- 
    ation between multiple markers by capturing them concurrently in its 
    FOV. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 17  

3.1 Camera pose estimation: Arrows indicate the direct pose estimate of one 
    object (camera or marker) with respect to another, where the pose of the 
    object on the arrow’s tail is given with respect to that on the arrow’s head. . 22  
3.2 Localization graph for system in Figure 3.1: Edge weights represent pose 
    estimation uncertainty values. . . . . . . . . . . . . . . . . . . . . . . . . . . . . 23  

4.1 ARToolKit (ATK) marker design . . . . . . . . . . . . . . . . . . . . . . . . . . . . 26  
4.2 Hoffman marker system (HOM) marker design . . . . . . . . . . . . . . . . . 26  
4.3 Institut Graphische Datenverarbeitung (IGD) marker design . . . . . . . . . . . 27  
4.4 Siemens Corporate Research (SCR) marker design . . . . . . . . . . . . . . . 27  
4.5 Finding the ground truth of the marker corners using edge detection, least-
    squares fitting, and intersection calculation. . . . . . . . . . . . . . . . . . . . . 28  
4.6 Marker design: Two examples of the markers used in the implementation 
    of the localization system. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 29  
4.7 Cross-ratio constraint: The diagonal corners of the inner and outer edges 
    (labeled A, B, C, and D) of the border are used to calculate the cross-ratio. . 30  
4.8 Marker code: The coded area of the marker lies within the border and is 
    divided into nine cells. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 31
LIST OF FIGURES

4.9 HALCON’s calibration plates: Sample images from HALCON’s image database illustrating the process of internal camera calibration through the use of calibration plates. 32
4.10 Current frame capture. 33
4.11 Contrasted frame. 33
4.12 Refined image of the marker code displayed without perspective distortion. 34
4.13 Sampled regions of the marker code (one per cell). 34
4.14 Displaying the coordinate axes of the marker after its true pose is found. 35
4.15 Positional relationship between a point and a convex polygon: Considering the polygon as a path to determine whether a point lies within it. 37
4.16 Output of the Python program. 40

5.1 Visual description of the reprojection error (the pixel distance between the arrows). 42
5.2 Progressively blurrier image captures of a marker. Note how the edges are less distinctively defined in the blurrier captures. 43
5.3 An error ellipse representing the positional uncertainty of the marker origin. 44
5.4 The perpendicular distance between the camera and a marker. 45
5.5 An illustration of the different positions that the camera is placed and localization results are recorded for the local scale experiment. The grid starts at a distance of 1.116 m from the marker in the negative z direction, 0.313 m in the y direction and is centered at x = 0 m. The dimensions of the grid are 2.508 × 0.456 m with the spacing between the dots along the x and y directions being 0.228 m. 46
5.6 Graph of ground truth error vs. the reprojection error metric. 47
5.7 Graph of the ground truth error vs. the blur metric. 47
5.8 Graph of the ground truth error vs. the error ellipse volume metric. 48
5.9 Graph of the ground truth error vs. the perpendicular distance metric. 48
5.10 An illustration of the different positions that the camera is placed and localization results are recorded for the global scale experiment. The grid starts at a distance of 0.888 m from the reference marker in the negative $z$ direction, 0.313 m in the $y$ direction and with its right edge at $x = 0$ m. The dimensions of the grid are $2.508 \times 0.456$ m with the spacing between the dots along the $x$ and $y$ directions being 0.228 m.

5.11 Experimental map: (a) arrangement of markers in the environment, (b) topological representation of the map.

5.12 Three image captures of the marker configuration and experiment setup.

5.13 Graph of ground truth error vs. the aggregate error based on the linear function ($f(d) = d$) of the perpendicular distance metric.

5.14 Graph of ground truth error vs. the aggregate error based on the quadratic function ($f(d) = d^2$) of the perpendicular distance metric.

5.15 Graph of ground truth error vs. the aggregate error based on the exponential function ($f(d) = 3^d - 1$) of the perpendicular distance metric.

5.16 Graph of ground truth error vs. the aggregate error based on the exponential function ($f(d) = 50^d - 1$) of the perpendicular distance metric.
List of Tables

D.1 Raw data for the Ground Truth Error Vs. Error Metric figures. . . . . . . . . 88
D.2 Raw data for the Ground Truth Error Vs. Aggregate Error figures. . . . . . 90
Chapter 1

Introduction

With the growing demand for autonomous robots in industrial, medical, domestic, and other domains, a large portion of research in the robotics industry has been geared toward the development and improvement of self-localization systems (i.e., systems that can estimate their own pose within an environment through the use of sensors).

1.1 Single-Sensor Vs. Multi-Sensor Systems

For the purposes of this work, existing self-localization systems in the literature will be divided into two categories: those that obtain their data from multiple sensors (e.g., [14], [3], [8]) and those that obtain them from a single sensor (e.g., [20], [30], [32]). The former type of system takes a sensor fusion approach. One major advantage of sensor fusion is the availability of multiple sources of data, through which the robot may verify the readings of its individual sensors and reduce the overall error of its pose estimates. The disadvantages of using multiple sensors are added complexity (in the localization algorithm and hardware design), larger form factor, and increased cost. Conversely, systems that use single-sensors tend to be simpler, smaller, and less expensive; however, they do not have the redundancy and fusion of multiple independent sensor measurements and must therefore use internal methods to reduce estimation error. The focus of this work is on reducing the estimation error of a single sensor localization system. A camera will be used as the sensor.

1The terms “self-localization” and “localization” refer to the same concept and will be used interchangeably throughout the thesis
CHAPTER 1. INTRODUCTION

1.2 The Camera As a Sensor in Self-Localization Systems

Compared to the sensing modalities used in other solutions (e.g., odometry, sonar, and laser), two-dimensional camera images provide a robot’s localization algorithm with more data about the environment [17]. They can be used by a localization system to detect, identify, and estimate the pose of objects in a scene with respect to the camera’s coordinate system. The pose of the object can then be inverted to estimate the camera’s pose with respect to the object’s coordinate system. Self-localization is achieved when the object’s coordinate system is also the global coordinate system of the environment.

However, when the object falls outside the camera’s sensing range, its pose cannot be found directly. Therefore, a map must be built to represent its surrounding environment such that when the object is outside the camera’s FOV (field of view), there is a path back to the object through the map.

1.3 Local and Global Error

The nature of the map-building process of vision-based self-localization yields two particular types of pose estimation errors: local error, which originates from error and noise in image capture and affects pose estimations made with respect to local coordinate systems in the image, and global error, which arises from the accumulation of local error and affects pose estimations made with respect to a global coordinate system that may not necessarily be in the image. Due to the influence local error has on global error, reducing the former would result in a reduction of the latter. Local error reduction is implemented in the proposed system herein through the use of markers (i.e., artificial landmarks\(^2\)) that can be accurately detected and discerned in a two-dimensional image [12], [7], [19].

With regards to the general goal of this work, however, a different perspective is taken: while reducing local error will cause a reduction in the global error, a more direct approach to reducing global error is by representing the surrounding environment as a topological map/graph whose edge weights reflect the effects of local error.

---

\(^2\)Landmarks are objects with distinct features that make them relatively easy to separate from their surroundings. Artificial landmarks are custom designed and added to the scene rather than being naturally occurring elements of the environment.
CHAPTER 1. INTRODUCTION

1.4 Existing Systems That Utilize Graphical Representations in Pose Estimation Applications

Graphical representation in similar pose estimation problems has previously been applied to such areas as multiview registration of 3D scenes and large-scale external calibration of camera networks. Sharp et al. [28] present a graphical approach to modeling neighbouring (i.e., overlapping) views in a network of range scanners (laser-based sensors). They apply an optimization to the graph to reduce the global error associated with multiview registration of 3D scenes. In the area of multi-camera calibration (where the external parameters include the relative poses between the cameras), Brand et al. [6] use the graphical approach to apply constraints on the viable positions of the cameras with respect to each other. This, like the system by Sharp et al., also uses neighbouring views to determine relative poses between the cameras.

There also exist a number of multi-sensor self-localization systems that use the graphical representation. The system introduced by Thrun and Montemerlo [29], called Graph-SLAM (where SLAM is an acronym for Simultaneous Localization and Mapping), uses a robot called Segbot that utilizes a scanning laser, an inertial measurement unit, and a GPS as its sensors. Each edge of its graph represents a nonlinear constraint that is weighted based on the uncertainty associated with the sensor measurements and the motion model. The map-building for this particular system is done offline. A similar system called Graphical SLAM, designed specifically for outdoor applications, is proposed by Folkesson and Christensen [13]. Like the Segbot, it uses a scanning laser and an inertial measurement unit as its sensors and like the system by Thrun and Montemerlo, it represents the environment as a topological graph with nonlinear constraints between detected features.

To the best of the author’s knowledge, a single-sensor vision-based self-localization system utilizing the topological graph representation has not been presented in literature.

1.5 Single-Sensor, Vision-Based Self-Localization Systems

The following systems perform localization solely through the use of a vision-based sensor (i.e., a camera). Note that purely vision-based localization systems are not very common in the literature, with most of the contributions coming from the field of augmented re-
ality, where large scale localization is not required in many cases. These systems do not implement a graphical representation of the environment but are presented to introduce the reader to some current vision-based localization systems in literature.

**An Extended Marker-Based Tracking System for Augmented Reality**

Jun et al. [15] propose a ceiling-based marker tracking system. There are two planar marker systems which are explained in detail in this paper: ARToolkitPlus and ARTag. These systems take the captured images of the markers as input and then output their pose with respect to the camera which captured them. ARToolkitPlus uses a simple grayscale intensity threshold to extract the fiducial marker from the image; then it identifies the marker based on its pattern; and finally, it calculates the homography using the square black border which surrounds the marker pattern. However, with the use of a global threshold, the output can be affected by abnormal lighting, even if it only affects a single image. In contrast, the ARTag system completes these tasks by using differential intensity thresholds to extract edges from the marker. ARTag identifies each marker by its extracted edge pattern. It uses the corners of the quadrilaterals in the pattern (formed by the edges) to calculate each marker’s homography.

The idea of using multiple markers stems from the need to extend the range of the global coordinate system. Hence, transformations of the local coordinate systems of each marker to the global coordinate system are required to be found.

**Real-Time Camera Pose Estimation for Augmented Reality System Using a Square Marker**

Most current augmented reality systems use markers in the scene to estimate the pose of the device with respect to the global coordinate system. However, when these markers are blocked by obstacles or they fall out of the view of the camera, the pose estimation cannot be calculated with sufficient accuracy, if at all. An example of such a system is ARToolkit.

Lee et al. [18] present a method to handle the problems caused by occlusions. In the captured image, the features are detected using the Shi-Tomasi corner detector. Next, these features are tracked using a Lukas-Kanade Tracker (LKT), which is appropriate for this application due to its ability to perform in real-time. However, the trade-off for the LKT’s relatively fast calculations is that it has a narrow “search range” which limits its peripheral
vision. When the marker falls out of range of the LKT, the features must be re-detected and the tracking must be restarted.

**A Six-DOF Motion Tracking System for Markered Environment**

Yang et al. [31] describe a method of camera pose estimation using a non-traditional marker-based system. Infrared LEDs are placed on the ceiling as markers in a predetermined pattern. An offline process is undertaken to determine the 3D coordinates of each LED with respect to a common world coordinate system. An infrared filter is placed on the camera to reduce the effects of illumination changes by filtering out most of the visible light spectrum.

During the offline process, a set of images are taken from multiple points of view, their features are extracted, labeled, and matched. Then, an incremental iterative optimization method is applied to obtain each LED’s 3D coordinate.

During the online process, the information obtained through the offline process is used to estimate the pose of the camera: 2D feature points are extracted from each frame in the live video and matched with their corresponding 3D coordinates of the LEDs.

**1.6 Main Contribution**

The main contribution of the proposed system is the application of established topological map representation methods to reduce global error in single-sensor vision-based localization systems.

By applying a shortest path algorithm, the map can be optimized to yield the paths of minimum global error (i.e., accumulated local error) from the camera to the global coordinate system. Furthermore, this system will provide the potential for online map-building in implemented systems, precluding the need for preliminary training.

The topological graph representations of the environment are as follows: the markers will represent the vertices, the existence of a relative pose between two markers will be represented by an edge, and the edge weights will be based on an appropriately derived error metric.
Chapter 2

Theoretical Foundation

Information obtained from a camera comes in the form of 2D images. In applications involving localization in a 3D environment, 2D images can be used to indirectly obtain 3D information. This can be done by using multiple cameras to find point correspondences between multiple views and performing triangulation on the points. Another, more economical, method is through the use of a single camera and a priori data about targets (i.e., markers) in the image – as in the proposed system herein. This chapter will explain the theoretical background regarding the extraction of 3D information from 2D images:

2.1 Image Formation

When a 3D representation of a scene is reduced to 2 dimensions, the result is called an image. More specifically, an image captured by a camera (either CMOS or CCD type) is the result of a 3D geometric transformation, which takes 3D information and describes it in a 2D framework.

A coordinate system having an origin $O$, called the center of projection, and three axes, $O_x, O_y, O_z$, will be used to represent the reference frame (i.e., coordinate system) of the camera; it is illustrated in Figure 2.1. It can be noted that the $O_x$ and $O_y$ axes are parallel to

---

1 The information of Sections 2.1, 2.2, and 2.3 is based on the lecture notes of 03-60-551 (Visual Processing), prepared by Dr. Boubakeur Boufama – a professor in the School of Computer Science at the University of Windsor. The figures used are also credited to the same lecture notes by Dr. Boufama.

2 Complementary Metal-Oxide-Semiconductor

3 Charge-Couple Device
the image plane and are correspondent to the row and column directions, respectively. \( O_z \), called the *optical axis*, is perpendicular to the image plane.

![Diagram of image formation process](image.png)

**Figure 2.1**: Illustration of the process of image formation.

Three geometric transformations, applied in sequential order, are required to express 3D scene points in 2D image coordinates (i.e., pixels):

1. **A 3D Euclidean transformation**: A transformation is applied to the 3D points, defined in the scene coordinate system, such that they are expressed in the camera coordinate system. Six parameters are involved in this transformation, corresponding to the translation and rotation operations, each with respect to the three axes.

2. **A 3D-2D projection**: The 3D points in the camera coordinate system are projected onto the image plane and are now referred to as *normalized coordinates*.

3. **A 2D-2D transformation**: An affine transformation is applied to the normalized coordinates to express the 2D point in pixel coordinates in the image plane. It is expressed in equation (2.1).
\[ A = \begin{pmatrix} \alpha_u & -\alpha_u \cot \theta & u_0 \\ 0 & \alpha_v \sin \theta & v_0 \\ 0 & 0 & 1 \end{pmatrix} \] \quad (2.1)

where,

- \( \alpha_u \) and \( \alpha_v \) are scale factors along the \( O_x \) and \( O_y \) directions, respectively.
  - \( \alpha_u = -f k_u \) and \( \alpha_v = -f k_v \).
  - \( f \) is the distance (usually in \( mm \)) between \( O \) and the image plane, called the \textit{focal length}.
  - \( k_u \) and \( k_v \) are the number of pixels per \( mm \) along the \( O_x \) and \( O_y \) directions, respectively.

- \( u_0 \) and \( v_0 \) are the pixel coordinates of the center of the image, which is defined as the intersection of the optical axis and the image plane.

- \( \theta \) is the angle between the \( u \) and \( v \) axes of the image. Due to some errors that arise during the manufacturing process of the camera, this angle may not be exactly \( \pi/2 \). However, in the modern cameras, it can be assumed that it is equal to \( \pi/2 \) because the error is so small. Therefore, equation (2.1) may be expressed as shown in equation (2.2).

\[ A = \begin{pmatrix} \alpha_u & -\alpha_u & u_0 \\ 0 & \alpha_v & v_0 \\ 0 & 0 & 1 \end{pmatrix} \] \quad (2.2)

Combining the three geometric transformations above, the following matrix \( M \) is obtained:

\[ M = AID \] \quad (2.3)

where,

- \( A \) is the 2D-2D transformation,
- \( I \) is the 3D-2D projection,
CHAPTER 2. THEORETICAL FOUNDATION

- and $D$ is the 3D Euclidean transformation.

2.2 The Pinhole Camera Model

There are a number of ways to approximately model the geometry of a camera; three popular ones are the pinhole model, the orthographic model, and the weak perspective model. The most widely used approximation for cameras, however, is the pinhole model. This is the model on which the proposed system will be based. In this model, the 3D-2D projection of a point a 3D point $P$ to a 2D point $p$ (on an image plane) is a pure perspective projection through $O$ (as in Figure 2.2). With this information, the geometric transformations will be explained in the context of the pinhole camera model.

Figure 2.2: Illustration of the pinhole camera model.
2.2.1 3D Euclidean Transformation

Assume that points $P = (X, Y, Z)$ and $P' = (X', Y', Z')$ represent the same physical point, where $P$ is defined in the scene coordinate system, and $P'$ is defined in the camera coordinate system. The relationship between $P$ and $P'$ is given in equation (2.4), where the matrix of $r_{ij}$ entries represents a rotation and the vector with entries $t_x, t_y,$ and $t_z$ represents a translation. Both operations are being applied to the point $P$.

$$
\begin{pmatrix}
X' \\
Y' \\
Z'
\end{pmatrix} =
\begin{pmatrix}
r_{11} & r_{12} & r_{13} \\
r_{21} & r_{22} & r_{23} \\
r_{31} & r_{32} & r_{33}
\end{pmatrix}
\begin{pmatrix}
X \\
Y \\
Z
\end{pmatrix} +
\begin{pmatrix}
t_x \\
t_y \\
t_z
\end{pmatrix}
$$  \hspace{1cm} (2.4)

Using homogeneous coordinates, equation (2.4) can be expressed as

$$
\begin{pmatrix}
X' \\
Y' \\
Z' \\
1
\end{pmatrix} =
\begin{pmatrix}
r_{11} & r_{12} & r_{13} & t_x \\
r_{21} & r_{22} & r_{23} & t_y \\
r_{31} & r_{32} & r_{33} & t_z \\
0 & 0 & 0 & 1
\end{pmatrix}
\begin{pmatrix}
X \\
Y \\
Z \\
1
\end{pmatrix}
$$  \hspace{1cm} (2.5)

or more conveniently as

$$
P' = DP
$$  \hspace{1cm} (2.6)

where,

- $D$ is a $4 \times 4$ matrix that represents a rigid 3D displacement.

- $P$ and $P'$ are the homogeneous coordinates expressed with respect to the scene and camera coordinate systems, respectively.

2.2.2 3D-2D Projection

Figure 2.3 illustrates how a 3D point, $P(X, Y, Z)$ is projected onto an image plane to become a 2D point, $p(x, y)$.

Note that the focal length $f$ is also represented in the image.

From the properties of similar triangles, the coordinates of point $p$ ($x$ and $y$) can be expressed in terms of the coordinates of point $P$ ($X$, $Y$, and $Z$) as in the following equations:
Figure 2.3: Perspective projection of a 3D point $P$ to a 2D point $p$ on an image plane.

\[
\frac{Z}{f} = \frac{X}{x} \Rightarrow x = f \frac{X}{Z} \tag{2.7}
\]

\[
\frac{Z}{f} = \frac{Y}{y} \Rightarrow y = f \frac{Y}{Z} \tag{2.8}
\]

These equations can be expressed in matrix form as in equation (2.9), where $f = 1$ is assumed.

\[
\begin{pmatrix}
    x \\
    y \\
    1
\end{pmatrix}
= \lambda
\begin{pmatrix}
    1 & 0 & 0 & 0 \\
    0 & 1 & 0 & 0 \\
    0 & 0 & 1 & 0
\end{pmatrix}
\begin{pmatrix}
    X \\
    Y \\
    Z \\
    1
\end{pmatrix} \tag{2.9}
\]

which can be compacted to the following:

\[
p = \lambda IP \tag{2.10}
\]
CHAPTER 2. THEORETICAL FOUNDATION

where,

- \( I \) is a \( 3 \times 4 \) matrix that represents perspective projection.
- \( \lambda \) is a scale factor.

Therefore, obtaining a projected point \( p \) of a 3D point \( P \), expressed in the scene coordinate system would require the operation

\[
p = \lambda IDP
\]  

where \( D \) alters \( P \) such that it is expressed in the camera coordinate system and \( I \) applies a perspective projection to express its 2D projection on the plane.

2.2.3 2D-2D Transformation

To express \( P \) in pixel coordinates, two further operations must be performed: a scaling and a translation. This is done by applying the matrix \( A \), defined in Section 2.1. Although the scaling changes the units to pixels, the translation must be applied so that the origin of the coordinate system is moved to the upper-left corner of the image plane\(^4\). In equation form, the pixel coordinates \((u, v)\) (where \( u \) and \( v \) are the column and row indices of a pixel, respectively) corresponding to the projection of \( P \) are given in equations (2.12) and (2.13).

\[
u = \alpha_u x + u_0
\]  

\[
v = \alpha_v y + v_0
\]  

The following equations express these in matrix form:

\[
\begin{pmatrix}
u \\
v \\
1
\end{pmatrix} = \begin{pmatrix}
\alpha_u & 0 & u_0 \\
0 & \alpha_v & v_0 \\
0 & 0 & 1
\end{pmatrix} \begin{pmatrix}
x \\
y \\
1
\end{pmatrix}
\]  

\(^4\)This is the conventional position of the origin of an image defined in pixel coordinates.
2.2.4 Combining the Transformations

Recall from equation (2.3) that $M = AID$ is the matrix representing the three combined transformations. After performing matrix multiplication, $M$ is expressed in the form of

$$M = \begin{pmatrix} \alpha u r_{11} + u_0 r_{31} & \alpha u r_{12} + u_0 r_{32} & \alpha u r_{13} + u_0 r_{33} & \alpha u t_x + u_0 t_z \\ \alpha v r_{21} + v_0 r_{31} & \alpha v r_{22} + v_0 r_{32} & \alpha v r_{23} + v_0 r_{33} & \alpha v t_y + v_0 t_z \\ r_{31} & r_{32} & r_{33} & t_z \end{pmatrix}$$ \hspace{1cm} (2.15)

However, for simplicity, it will be represented as

$$M = \begin{pmatrix} m_{11} & m_{12} & m_{13} & m_{14} \\ m_{21} & m_{22} & m_{23} & m_{24} \\ m_{31} & m_{32} & m_{33} & m_{34} \end{pmatrix}$$ \hspace{1cm} (2.16)

Hence, the pixel coordinates $p_i = (u_i, v_i)$ of the projected 3D point $P_i = (X_i, Y_i, Z_i)$ can be found by the following equation:

$$\begin{pmatrix} u_i \\ v_i \\ 1 \end{pmatrix} = \lambda \begin{pmatrix} m_{11} & m_{12} & m_{13} & m_{14} \\ m_{21} & m_{22} & m_{23} & m_{24} \\ m_{31} & m_{32} & m_{33} & m_{34} \end{pmatrix} \begin{pmatrix} X_i \\ Y_i \\ Z_i \\ 1 \end{pmatrix}$$ \hspace{1cm} (2.17)

which, in equation form, becomes

$$u_i = \frac{m_{11} X_i + m_{12} Y_i + m_{13} Z_i + m_{14}}{m_{31} X_i + m_{32} Y_i + m_{33} Z_i + m_{34}}$$ \hspace{1cm} (2.18)

$$v_i = \frac{m_{21} X_i + m_{22} Y_i + m_{23} Z_i + m_{24}}{m_{31} X_i + m_{32} Y_i + m_{33} Z_i + m_{34}}$$ \hspace{1cm} (2.19)

2.3 Camera Calibration

Camera calibration is a process by which a camera’s internal/intrinsic (i.e., focal length, pixel cell dimensions, optical center, etc.) and external/extrinsic (i.e., pose of the camera in relation to the scene) parameters are determined. In terms of the outlined theory, it is the
estimation of the projection matrix $M$.

In $M$, there are a total of 12 unknowns to be estimated. To solve the unknowns, corresponding pixel and 3D coordinates of the same points in a scene can be used as constraints. Pixel coordinates of points of interest can be retrieved from a captured image after they are detected by their features (either manually or automatically). The corresponding 3D coordinates of the same points must be known a priori through measurement (e.g., using laser or accurate ruler). Calibration patterns, such as the one in Figure 2.4, are used because the 3D coordinates of their patterns with respect to the scene coordinate system can be easily obtained and their features can be accurately detected in an image. With calibration plates, the scene coordinate system can be chosen to lie on their surface so that the 3D coordinates are essentially reduced to 2 dimensions, further simplifying the process.

![Figure 2.4: An example calibration pattern.](image)

The calibration process begins by putting a calibration pattern in front of the camera and adjusting the camera (i.e., focus) until a clear picture of the pattern can be seen. From this position, an image is taken. The points of interest in the image are extracted and matched with their corresponding 3D points. The 2D and 3D points are input into a calibration software to estimate $M$.

**Estimating $M$**

Assuming the availability of 2D-3D point correspondences, $(u_i, v_i)$ and $(X_i, Y_i, Z_i)$ in equations (2.18) and (2.19) are known. Rearranging these equations results in the following:
CHAPTER 2. THEORETICAL FOUNDATION

\[ u_i = \frac{m_{11}X_i + m_{12}Y_i + m_{13}Z_i + m_{14}}{m_{31}X_i + m_{32}Y_i + m_{33}Z_i + m_{34}} = 0 \] (2.20)

\[ v_i = \frac{m_{21}X_i + m_{22}Y_i + m_{23}Z_i + m_{24}}{m_{31}X_i + m_{32}Y_i + m_{33}Z_i + m_{34}} = 0 \] (2.21)

Note that these equations are non-linear. The equations are linearized by multiplying both sides of the equations by the denominators:

\[ u_i (m_{31}X_i + m_{32}Y_i + m_{33}Z_i + m_{34}) - (m_{11}X_i + m_{12}Y_i + m_{13}Z_i + m_{14}) = 0 \] (2.22)

\[ v_i (m_{31}X_i + m_{32}Y_i + m_{33}Z_i + m_{34}) - (m_{21}X_i + m_{22}Y_i + m_{23}Z_i + m_{24}) = 0 \] (2.23)

When \( n \) pairs of points \( (P_i, p_i), i = 1, \ldots, n, \) are available, equations (2.22) and (2.23) can be written in matrix form as

\[
\begin{pmatrix}
-X_1 & -Y_1 & -Z_1 & -1 & 0 & 0 & 0 & 0 & u_1X_1 & u_1Y_1 & u_1Z_1 & u_1 \\
0 & 0 & 0 & 0 & -X_1 & -Y_1 & -Z_1 & -1 & v_1X_1 & v_1Y_1 & v_1Z_1 & v_1 \\
-X_2 & -Y_2 & -Z_2 & -1 & 0 & 0 & 0 & 0 & u_2X_2 & u_2Y_2 & u_2Z_2 & u_2 \\
0 & 0 & 0 & 0 & -X_2 & -Y_2 & -Z_2 & -1 & v_2X_2 & v_2Y_2 & v_2Z_2 & v_2 \\
\vdots
\end{pmatrix}
\begin{pmatrix}
m_{11} \\
m_{12} \\
m_{13} \\
m_{14} \\
m_{21} \\
m_{22} \\
m_{23} \\
m_{24} \\
m_{31} \\
m_{32} \\
m_{33} \\
m_{34}
\end{pmatrix}
= 0
\] (2.24)

Representing this in a compact form:

\[ AV = 0 \] (2.25)

where,
• $A$ is a $2n \times 12$ measurement matrix.

• $V$ is a 12-element vector consisting of unknown elements.

To solve for the 12 entries of $M$, $n$ must be greater than or equal to 6, implying that at least 6 point correspondences are necessary to solve for all entries of $M$.

Note that if the internal camera parameters, such as the focus, are not altered during the course of self-localization, the results do not require recalculation; a single calibration is sufficient before the online self-localization begins. However, the same does not hold true for the external parameters, as they are constantly changing as the camera moves.

2.4 Pose

2.4.1 Pose Composition

A pose $P_{\alpha\beta}$ is a rigid three dimensional Euclidean transformation from the coordinate system of object $\alpha$ to the coordinate system of object $\beta$. This may be referred to as the pose of object $\alpha$ with respect to object $\beta$.

The inverse of pose $P_{\alpha\beta}$ may be denoted $P_{\alpha\beta}^{-1}$ or $P_{\beta\alpha}$. The former notation will be used here to emphasize that $P_{\alpha\beta}$ is the available direct estimate.

Successive pose transformations may be composed into a single pose:

$$P_{\alpha\gamma}(p) = (P_{\alpha\beta} \circ P_{\beta\gamma})(p) = P_{\beta\gamma}(P_{\alpha\beta}(p)) \quad (2.26)$$

Note that a left-composition convention is used to better illuminate the sequence of pose transformations.

The details of pose inversion and composition vary depending on the representation used. The reader is directed to any of the numerous texts on Euclidean geometry for a treatment appropriate to his or her working representation.
2.4.2 Relative Pose Calculation

Referring to Figure 2.5, the relative pose transformation of a marker $\alpha$ with respect to another marker $\beta$ is determined through pose composition to be

$$P_{\alpha\beta} = P_{\alpha c} \circ P_{\beta c}^{-1}$$

(2.27)

where $P_{\alpha c}$ and $P_{\beta c}$ are the poses of $\alpha$ and $\beta$ with respect to the camera $c$ (assumed to be available). Note that this calculation is made possible by the fact that both markers are in the camera’s FOV (shaded area in Figure 2.5) at the same time; more specifically, pose estimates are taken from the same captured camera frame.

Figure 2.5: Relative pose calculation: The camera can calculate the relative pose transformation between multiple markers by capturing them concurrently in its FOV.
2.5 Shortest Path Algorithm

Shortest path algorithms can be applied to graphs to find the minimum topological path between two nodes. The famous shortest path algorithm by Dijkstra [10] solves the single-source shortest path problem of undirected graphs with non-negative edge weights. Since globally localizing a camera involves obtaining its pose with respect to a single source (i.e., the global coordinate system), Dijkstra’s algorithm is appropriate for computing the shortest path of edges from said source to the camera.
Chapter 3

Proposed Solution

3.1 Preliminaries

3.1.1 Marker Graph

The marker graph (based on the calibration graph introduced by Mavrinac et al. [22]) is a method of representing a set of markers as a topological map. It is a weighted, undirected graph $G_M = (\mathcal{M}, E_M, W_M)$, where $\mathcal{M}$ is the set of detected markers in the system, $E_M$ is a set of edges, and $W_M$ is the set of weights corresponding to the edges in $E_M$. The existence of an edge $\{\alpha, \beta\} \in E_M$ indicates that a relative pose transformation from marker $\alpha$ to marker $\beta$ (or vice versa) is available.

Since it is trivial to invert a pose, the availability of $P_{\alpha\beta}$ implies availability of $P_{\beta\alpha}$. The edge weight $(w_{\alpha\beta} \in \mathbb{R}^+) \in W_M$ is the estimation uncertainty of $P_{\alpha\beta}$.

A path $p = \langle \alpha, \ldots, \beta \rangle$ in $G_M$, from node $\alpha$ to node $\beta$, represents a sequence of pose transformations which may be composed to yield $P_{\alpha\beta}$. If $p = \langle v_1, v_2, \ldots, v_n \rangle$,

$$P_{1,n} = P_{1,2} \circ P_{2,3} \circ \cdots \circ P_{n-1,n} \quad (3.1)$$

where $P_{i,j}$ is the pose transformation from $v_i$ to $v_j$. If any $P_{i,j}$ is not available, $P_{i,j} = P_{j,i}^{-1}$. The aggregate error associated with this pose is

$$w_{1,n} = \sum_{k=2}^{n} w_{k-1,k} \quad (3.2)$$
which is the length of path $p$.

### 3.1.2 Localization Graph

The *localization graph* is essentially a marker graph that includes the camera $c$ as an additional node. It is a weighted, undirected graph $G_L = (L, E_L, W_L)$, where $L = c \cup M$, $E_L$ is the set of edges between nodes $L$, and $W_L$ is, again, the set of associated edge weights. The localization graph is incrementally updated as the camera $c$ moves through the environment. Note that $G_L \supset G_M$.

### 3.2 Global Localization

#### 3.2.1 Assumptions

**Internal Calibration and Pose Estimation**

It is assumed that there exists some means by which a camera may estimate, from a single view, its relative three dimensional pose with respect to a calibration target of known structure [35], [4]. This normally implies that the camera is internally calibrated.

**Marker Constraints**

It is assumed that (i) if a marker $m \in M$ is connected to an edge, it remains fixed in its position, and (ii) the selected reference node $R$ corresponds to a marker that is available and detectable in the environment.

**Map Updates**

It is assumed that any operation that updates $G_M$ simultaneously updates $G_L$, and vice versa.

#### 3.2.2 Problem Definition

The problem of global localization using computer vision is formalized as follows:

Given a monocular camera $c$, a set of markers $M$, and an arbitrary global reference frame $R \in M$, find $P_{cR}$ as $c$ traverses the environment.
Let us further define a set of markers $\mathcal{V}$ which are in the camera’s current FOV (e.g., shaded area in Figure 2.5). Then, it can be noted that $P_{cR}$ may either be obtained directly from $R$ (when $R \in \mathcal{V}$) or indirectly from another marker $v \in \mathcal{V}$ (when $R \notin \mathcal{V}$), assuming there exists a path in $\mathcal{G}_L$ from $c$ to $R$.

It is additionally desirable to decrease the global error by using the path $p$ yielding the minimum aggregate error (as defined in (3.2)) for each estimated $P_{cR}$.

### 3.2.3 Self-Localization Method

The method will be explained with the aid of an example. Suppose it is desired to find the pose of a monocular camera $c$ within an environment consisting of markers $\mathcal{M} = \{W, X, Y, Z, R\}$, as shown in Figure 3.1. In this case, $R$ is selected as the global reference frame, so the problem is to find $P_{cR}$. As mentioned previously, there are two ways of finding $P_{cR}$: either directly through $R$ (when $R$ is in the FOV) or indirectly through intermediate markers $W, X, Y, Z$ (when $R$ is not in the FOV).

As shown in Figure 3.2, the localization graph is connected assuming that the direct pose estimates in Figure 3.1 are all available. Thus, the camera $c$ can be localized with respect to the common reference frame $R$ using any of the markers in the map.

The minimum requirement to achieve global localization is that there must exist a path from the current position of $c$ to the reference frame $R$ in $\mathcal{G}_L$. Additional edges may yield shorter paths (i.e., pose compositions with lower aggregate error). The positioning of the set of markers $\mathcal{M}$ should be chosen appropriately. Note that there is no disadvantage, aside from additional effort positioning and obtaining pose estimates, to increasing the size of $\mathcal{M}$.

In this example, direct pose estimates $P_{Wc}, P_{WX}, P_{YX}, P_{Zx}, P_{YZ}, P_{ZR}$, and $P_{ZR}$ are obtained, along with their respective pose uncertainties. The availability of direct pose estimates is encapsulated in the localization graph of Figure 3.2.

The solution is obtained through composition of the estimated poses, according to (3.1), where the shortest paths are computed using Dijkstra’s algorithm or similar. As an example, suppose the shortest path from $c$ to $R$ in $\mathcal{G}_L$ is $\langle c, W, X, Y, Z, R \rangle$. Then,

$$P_{cR} = P_{Wc}^{-1} \circ P_{WX} \circ P_{YX}^{-1} \circ P_{YZ} \circ P_{ZR} \quad (3.3)$$
CHAPTER 3. PROPOSED SOLUTION

Figure 3.1: Camera pose estimation: Arrows indicate the direct pose estimate of one object (camera or marker) with respect to another, where the pose of the object on the arrow’s tail is given with respect to that on the arrow’s head.

as per (3.1). The associated aggregate error is \( w_{cR} = w_{Wc} + w_{WX} + w_{YX} + w_{YZ} + w_{ZR} \).

Map Updating

When there are multiple markers in the FOV, the relative poses between all possible pairs of markers in the frame are calculated and connecting edges (with associated weights) are created between them. If in a subsequent frame an edge is re-detected and is found to have a lower weight than the existing edge, its relative pose transformation and weight overwrite those corresponding to the existing edge. In this way, the edges of the graph maintain the minimum weights (and thus the relative poses with the least uncertainty) at all times.
3.2.4 Self-Localization Algorithm

In the formal expression of the algorithm (Algorithm 1), let primed (') variables represent calculations made in the current frame (e.g., $P'_{ij}$ represents the relative pose between markers $i$ and $j$ as calculated from the current frame). Three previously undefined functions are used in this algorithm: The first, $calcw(\{\alpha, \beta\})$, calculates the weight of the edge connecting nodes $\alpha$ and $\beta$ based on an appropriately derived error metric; the second, $con(G, s, d)$, returns $True$ if there exists a path from $s$ to $d$ in $G$; the third, $sp(G, m, R)$, returns the shortest path $p_s$ from $m$ to $R$ in $G$ (using Dijkstra’s algorithm) in the form defined in Section 3.1.1. The boolean variable $n$ will be used to indicate that a map update has occurred.
Algorithm 1 Proposed Self-Localization Algorithm (Finding $P_{cR}'$)

1: $\mathcal{M} \leftarrow \{R\}$
2: $E_M, W_M, V \leftarrow \emptyset$
3: $n \leftarrow False$
4: loop
5:     Capture frame
6:     if $V \neq \emptyset$ then
7:         $\mathcal{M} \leftarrow \mathcal{M} \cup V$
8:         if $|V| > 1$ then
9:             for all $\{v_i, v_j\} \in \binom{V}{2}$ do
10:                 $P_{ij}' \leftarrow P_{ic}' \circ P_{jc}'^{-1}$
11:                 $P_{ji}' \leftarrow P_{ij}'^{-1}$
12:                 if $\{v_i, v_j\} \notin E_M$ then
13:                     $E_M \leftarrow E_M \cup \{v_i, v_j\}$
14:                     $w_{ij}, w_{ji} \leftarrow \infty$
15:                     $W_M \leftarrow W_M \cup w_{ij}, w_{ji}$
16:                 end if
17:                 if $calcw'({v_i, v_j}) < w_{ij}$ then
18:                     $w_{ij}, w_{ji} \leftarrow calcw'({v_i, v_j})$
19:                     $P_{ij} \leftarrow P_{ij}'$
20:                     $P_{ji} \leftarrow P_{ji}'$
21:                     $n \leftarrow True$
22:             end if
23:         end for
24:     end if
25:     if $n = True$ then
26:         for all $\{m \in \mathcal{M} | \text{con}(G_M, m, R)\}$ do
27:             $p_s \leftarrow sp(G_M, m, R)$
28:             $P_{mR} \leftarrow P_{p_s,1} \circ P_{p_s,2}$
29:             $w_{mR} \leftarrow w_{p_s,1} + w_{p_s,2}$
30:             for $k = 2 \rightarrow |p_s| - 1$ do
31:                 $P_{mR} \leftarrow P_{mR} \circ P_{p_s,k,p_s,k+1}$
32:                 $w_{mR} \leftarrow w_{mR} + w_{p_s,k,p_s,k+1}$
33:             end for
34:         end for
35:     end if
36:     if $R \in V$ then
37:         $P_{cR}' = P_{Rc}'^{-1}$
38:     else if $\exists \{v \in V | \text{con}(G_M, v, R)\}$ then
39:         $v_m \leftarrow \arg\min_{v \in V} calcw'({v, c}) + w_{vR}$
40:         $P_{cR}' \leftarrow P_{v_m}^{-1} \circ P_{v_mR}$
41:     end if
42: end if
43: $n = False$
44: end loop
Chapter 4

Implementation

4.1 Marker Design

Existing Marker Designs in Literature

The survey paper by Zhang et al. [33] provides a comparison of a few marker designs to give the reader a background of existing marker systems in literature\(^1\). These systems are compared quantitatively and qualitatively in terms of four criteria: usability, efficiency, accuracy, and reliability. In most marker systems, the square shape is utilized for pose estimation because it provides four easily detectable point correspondences (corners of the square). The following square-based marker systems were compared:

- ARToolKit (ATK) [16]
- Hoffman marker system (HOM)\(^2\)
- Institut Graphische Datenverarbeitung (IGD) [1]
- Siemens Corporate Research (SCR) [34]

These four systems were chosen for their availability, expandability, and because they are suitable to represent other similar marker systems. The design of each marker is presented as follows:

\(^1\)The information and figures of the existing marker designs presented in this section are credited to Zhang et al. [33]

\(^2\)Developed by C. Hoffmann in 1994 at Siemens AG.
**ARToolKit (ATK) (Figure 4.1)** The ATK system performs simple template matching, where the marker is detected and identification is achieved by comparing the captured image of the marker with an internal database of marker images.

![ARToolKit (ATK) marker design](image)

Figure 4.1: ARToolKit (ATK) marker design

**Hoffman marker system (HOM) (Figure 4.2)** The HOM system uses binary decoding rather than template matching to decode the marker. It also includes 6 bits of encoding on its sidebar for added reliability in marker recognition. Developed in 1994, it has been used by Siemens and Framatome ANP for camera calibration and 3D reconstructions of power plants, chemical plants, and oil platforms.

![Hoffman marker system (HOM) marker design](image)

Figure 4.2: Hoffman marker system (HOM) marker design

**Institut Graphische Datenverarbeitung (IGD) (Figure 4.3)** The IGD marker system also uses binary coding. It is made of a $6 \times 6$ grid of black and white cells, where the inner $4 \times 4$ grid determines the orientation and coding, while the outer cells are used for detection and pose estimation.

**Siemens Corporate Research (SCR) (Figure 4.4)** The SCR marker system is similar to IGD except it codes its markers using filled circles within a select group of cells.
As mentioned, four criteria are used to compare the marker systems. The first is stability and it refers to a marker system’s compatibility with different AR (Augmented Reality) systems or computer platforms and operating systems. The second criterion is efficiency and it is evaluated by calculating the tracking time performance - i.e., the amount of time it takes to detect and decode a marker or the frame rate of the captured video when marker tracking is taking place. The third criterion is accuracy and it is represented by the error in feature extraction from the 2D images. A ground truth of the feature locations is found through a custom system developed by the authors. The final comparable criterion is reliability, which measures a marker system’s ability to perform under non-ideal conditions such as poor camera focus, large projective distortion, small region of interest, etc.

All systems are found to have satisfactory usability except for IGD, which requires the creation of a wrapper library, followed by compilation using an Intel C++ compiler in order function on a Windows OS.

In comparing the efficiencies of these systems, it is noted that ATK results in the best running time performance while SCR results in the worst. However, unlike SCR, the performances of ATK and HOM are significantly affected by the number of markers in an image.

The accuracy of the correspondences found by the marker tracking system was determined through a comparison with ground truths obtained by an OpenCV corner detector.
Another ground truth used by the authors consisted of detecting the edges of the marker, performing least-squares fitting of lines, and finding the intersections of those lines to be ground truths of the marker corners (Figure 4.5). Through experimentation, the authors found ATK to be the least accurate. This is because it directly extracts the features from the binary image (which reduces computational complexity but results in larger error in feature extraction).

![Image](image.png)

**Figure 4.5:** Finding the ground truth of the marker corners using edge detection, least-squares fitting, and intersection calculation.

Marker recognizability was tested under projective distortion, multiple marker images, small region of marker, and poorly-focused video. The results are given below:

- **Projective distortion** - best: HOM, worst: SCR
- **Images with multiple markers** - best: HOM, worst: SCR (excluding IGD which did not respond to multiple markers)
- **Small region of marker** - best: ATK, worst: IGD
- **Poorly-focused video** - best: ATK, worst: N/A

**Marker Design Used in This Work**

The marker used for the implementation is shown in Figure 4.6. It includes the following features:
• A black and white colour scheme creates a contrast that is advantageous for edge or region detection as it clearly defines the borders between the colour regions.

• Because of its simple 2-dimensional design, it can be printed on regular printing paper.

• The marker is divided into two areas:
  – Outer border: used for marker detection and pose estimation (see Section 4.1.1).
  – Inner code: used for identifying the marker and determining its proper angular orientation (see Section 4.1.2).

Figure 4.6: Marker design: Two examples of the markers used in the implementation of the localization system.

Note that this marker design was chosen for its ease of implementation rather than for achieving optimal performance and accuracy results.

4.1.1 Outer Border

The outer border aids in marker detection by providing two key constraints that can be used to distinguish the marker from its surroundings. The first, as mentioned above, is its black colour which can be used by colour thresholding algorithms to separate it from lighter areas in the image. The second is a geometric constraint called the cross-ratio (or “ratio of ratios”); its usefulness lies in the fact that it is invariant under Euclidean, affine, and projective geometry (which means that it remains constant under any perspective distortion in the image of the marker). The following is a detailed explanation of how it is derived in the context of the outer border:
Outer Border Cross-Ratio Constraint

The cross-ratio can be calculated using four collinear points. In the case of this specific marker design, the points are chosen to be the four diagonal corners of the inner and outer edges of the border. These points are shown in Figure 4.7 (labeled A, B, C, and D).

![Cross-ratio constraint: The diagonal corners of the inner and outer edges (labeled A, B, C, and D) of the border are used to calculate the cross-ratio.](image)

Equation (4.2) gives the formula for the cross ratio in terms of the line segments connecting the four points.

\[
\text{Cross-Ratio} = \frac{AC}{BD} = \frac{AB}{CD}
\]  

(4.1)

4.1.2 Inner Code

The design and purpose of the inner code will be explained with the aid of Figure 4.8(a). The code itself is found inside the outer border and is divided into nine cells (numbered in the figure). The colour of these cells (black or white) determines their value (1 or 0, respectively) in a nine-digit binary number that uniquely identifies each marker. Every marker has three black corner cells (cells 3, 7, and 9) and one white corner cell (cell 1); the white corner cell corresponds to the left-most binary digit and the subsequent digits correspond to the cells occurring consecutively from left to right and top to bottom (with respect to the white corner cell). Therefore, for the marker in the referred figure, the order of the cells is 1-2-3-4-5-6-7-8-9 and the corresponding binary code is 00111101. For ease of interpretation, the binary code is converted to a decimal number, 125. As another example, Figure 4.8(b) shows a marker with its white corner cell in a different location.
The order of the binary number in this case is 7-4-1-8-5-2-9-6-3 and the binary number itself would be 001110101 (or 117 in decimal).

![Marker code: The coded area of the marker lies within the border and is divided into nine cells.](image)

Note that the corner cells are also used to define the angular orientation of the marker. In Figure 4.8, marker (a) is oriented upright at $0^\circ$ while marker (b) is rotated $90^\circ$ in the counterclockwise direction.

Because the corner cells occupy four positions, five cells remain to differentiate each marker ID from the others. Therefore, there are $2^5 = 32$ distinct marker IDs.

## 4.2 Software Implementation

The software implementation for the localization system uses a combination of the HALCON [24] Integrated Development Environment (IDE) – called HDevelop [25] – and the Python programming language [11]. Sockets are used as a means of communication between the HDevelop IDE and the Python program. In this case, HDevelop is primarily used for marker detection and localization, while Python is used for pose composition and map updates. A preliminary internal calibration (using HDevelop) is done to find the internal parameters of the camera.

### 4.2.1 Internal Camera Calibration

The internal parameters of the camera are found using the HALCON Calibration Assistant. For a pinhole camera, such as the one being used in this implementation, this is done
by capturing multiple images of a calibration plate (whose dimensions are known) from multiple positions and angles in the camera’s FOV (see Figure 4.9). These images are used by HALCON to compute and optimize the internal parameters of the camera. Refer back to Section 2.3 for more details about the calibration process.

![Figure 4.9: HALCON’s calibration plates: Sample images from HALCON’s image database illustrating the process of internal camera calibration through the use of calibration plates.](image)

### 4.2.2 HDevelop Program

The following is a general outline of the HDevelop program, which can be found in Appendix B.

1. Send a request to the Python program and wait for acceptance to initialize socket communication.

2. Read the internal camera parameters obtained through the process outlined in Section 4.2.1.

3. Recompute the internal camera parameters to adjust for radial distortion in the captured frame.

4. While images are being received from the camera perform the following actions:
   
   A. Capture the current frame seen by the camera.
   
   B. Convert the frame to grayscale (if it is not already so).
   
   C. Enhance the contrast of the frame.
D. Use the adjusted internal parameters to remove the radial distortion from the frame.

E. Apply a threshold that selects the dark regions of the frame.

F. Eliminate the regions that are less than an appropriate minimum area (in $\text{pixels}^2$).

G. Extract the contours of the regions.

H. Eliminate the contours that are less than an appropriate minimum length (in $\text{pixels}$).

I. For the remaining contours, perform the following actions:
   (i) Detect the corners of all quadrilateral contours (the shape of the inner and outer edges of the marker’s border) and eliminate all contours of other shapes.
   (ii) Eliminate all contours that do not possess a child/parent contour inside/outside them.
   (iii) Sort the quadrilateral corners for use in calculating the cross-ratio.
   (iv) Calculate the cross-ratios of both diagonals of the marker (as explained in Section 4.1.1) and find their average.
(v) Eliminate all contours whose average cross-ratio error exceeds a certain threshold.

J. Because the inner code is yet to be processed, the true orientation of the marker cannot be known yet. Hence, for the time being, calculate a temporary pose using an HDevelop function that takes the marker’s parent contour, its physical dimensions (length and width), and the internal parameters of the camera as inputs and gives back the pose, the pose covariance matrix, and the reprojection error as outputs.

K. For each identified marker, perform the following actions:

(i) Using an HDevelop refining function, remove the perspective distortion on the frame so that it appears the marker code is directly in front of and parallel to the image plane (as in Figure 4.12). Give the internal camera parameters, the temporary pose of the marker, and the marker dimensions as input to this function.

![Figure 4.12: Refined image of the marker code displayed without perspective distortion.](image)

(ii) Enhance the contrast of the refined image to reduce colour ambiguity in the code cells.

(iii) Define nine regions – one for each code cell – and calculate the mean gray value of each one.

![Figure 4.13: Sampled regions of the marker code (one per cell).](image)
(iv) Use a gray value threshold to determine whether each cell is black or white and, based on the decision, assign the cell a binary value of 1 or 0, respectively, so that the marker’s temporary code is represented by a nine-digit binary number. Again, the determined code is temporary because the true orientation of the marker is not yet known.

(v) Determine the orientation of the marker by examining the corner cells as explained in Section 4.1). Using this information, rotate the temporary pose and reorder the temporary binary code (if the marker is not already in its true orientation) to find the true pose and binary code of the marker.

![Figure 4.14: Displaying the coordinate axes of the marker after its true pose is found.](image)

(vi) Define the ID of the marker as the decimal equivalent of its binary code.

L. Implement a custom error metric (or use the previously-obtained reprojection error) to quantify the quality of the estimated pose.

M. Convert all of the information obtained from the current frame into string format and use socket communication to send it to the Python program in the following form:

\[
(marker\ ID\ 1) : (pose\ 1) \mid (error\ 1) ; (marker\ ID\ 2) : (pose\ 2) \mid (error\ 2) ; \ldots
\]

5. Terminate socket communication.

Two functions in the above procedure (steps 4(I)i and 4(I)ii) are explained in greater detail as follows:
Finding Corners of Quadrilateral Contours

1. When determining the locations of the corners of a quadrilateral contour in the image, the contour must be segmented into line segments. The first step in doing this is to use an HDevelop operator that computes the convex hull of the contour; this simplifies the shape by removing jaggedness.

2. If a contour has less than four sides, it is excluded as a marker candidate.

3. The line segments making up the contour are sorted by length in descending order.

4. The marker may be occluded by the image boundary. In such a case, the algorithm may falsely assume that the boundary of the image is a side of the marker contour. Therefore, before moving forward, it is asserted that none of the lines of the contour lie on the image boundary.

5. The four longest lines of the contour are selected and all others are discarded.

6. HDevelop sorts the line segments in the order they occur in the counterclockwise direction. The angular difference between each pair of consecutively occurring line segments is calculated to assert that each of them is on a different side of the quadrilateral marker contour (i.e., if the angular difference is found to be less than a minimum angle, it is assumed that the consecutive pair of line segments are on the same side of the quadrilateral, and therefore the four line segments do not form a quadrilateral shape).

7. The line segments are extended and their intersections are defined to be the corners of the quadrilateral.

8. The four corners (defined in pixel coordinates) are returned from the function.

Determining if a Contour is Inside Another Contour

In determining if a contour is entirely enveloped by another contour, the problem is simplified by assuming that the contours in question are convex (this can be safely assumed because the shape of the marker itself is convex). Bourke [5] explains a method that determines the positional relationship between a point of interest and a convex polygon. By
considering the polygon as a directed “path” (as in Figures 4.15(a) and 4.15(b)), it can be noted that if a point of interest $P(x, y)$ lies on the same side of all the line segments in the path, it is inside the polygon. For example, in Figure 4.15(a), consider the line with endpoints $P_0(x_0, y_0)$ and $P_1(x_1, y_1)$. In this case, the path of the convex polygon is counterclockwise and the point $P(x, y)$ lies to the left of the line segment $P_0P_1$. Upon further inspection, the point $P$ also lies to the left of the other three line segments of the polygon. It is therefore concluded that $P$ lies inside the polygon. Figure 4.15(b) demonstrates a clockwise path with $P$ lying to the right of all four line segments. The formula $a = (y - y_0)(x_1 - x_0) - (x - x_0)(y_1 - y_0)$ is used to determine where $P$ lies in relation to $P_0P_1$; in an image with a coordinate system located in its top left corner, $a > 0$ implies that $P$ lies to the right, $a < 0$ implies that $P$ lies to the left, and $a = 0$ implies that $P$ is on the line $P_0P_1$.

Applying the above method to this function, the four corners of the quadrilateral (obtained previously) may be considered four points of interest with respect to a potential parent contour. If all four corners are found to be within the potential parent contour, it is concluded that the two contours possess a parent/child relationship (i.e., one is inside the other).

### 4.2.3 Python Program

The following is a general outline of the Python program, which can be found in Appendix C.

![Diagram](image)
1. Initialize program with the ID of the marker that is desired to be the reference.

2. Set up socket communication. Wait for and accept any incoming requests to begin socket communication with a program (i.e., HDevelop).

3. Initialize an empty marker graph with the aid of Hypergraph [21], a Python module for graphs and hypergraphs.\(^3\)

4. For the duration of the program perform the following actions:

   A. Parse the string sent from HDevelop (see step 4M in Section 4.2.2) to interpret the marker IDs, poses, and pose estimation errors. Each string sent from HDevelop represents the information from one frame.

   B. Update the set of vertices of the marker graph with the markers found in the current frame (i.e., add any markers that have not been previously added to the graph).

   C. Using the Adolphus [23] computer vision suite, perform the following actions for each marker in the frame:\(^4\)

      (i) Referring to the current marker as “marker A,” perform the following actions for all other markers in the frame:

         a) Choose a “marker B.”

         b) Find the relative pose of marker A with respect to marker B (directly) and also, that of marker B with respect to marker A (through inversion).

         c) Calculate the weight of the edge between markers A and B by averaging their respective estimation errors:

            \[
            \text{edge weight} = \frac{\text{error of marker } A + \text{error of marker } B}{2}
            \]

            \(\text{(4.2)}\)

         d) • If the edge joining markers A and B does not already exist in the marker graph . . .

            – add it to the marker graph and,

---

\(^3\)The Hypergraph module will be used in all subsequent graph operations, such as addition of vertices.

\(^4\)The Adolphus suite will be used in all subsequent computer vision operations, such as pose composition.
CHAPTER 4. IMPLEMENTATION

– if the reference marker is in the marker graph, (re-)calculate the shortest paths between each marker and the reference.

• However, if there is already an existing edge joining the two markers and the edge weight of the one in the current frame is less than that of the existing one...
  – replace the old edge and weight with the ones of the current frame and,
  – if the reference marker is in the marker graph, (re-)calculate the shortest paths between each marker and the reference.

(ii) If either of the conditions in step 4(C)id were met (i.e., if the edges of the graph were updated), compute the pose of each marker with respect to the reference by applying pose composition through the shortest paths. Also, calculate the associated aggregate error (i.e., the sum of the edge weights from a marker to the reference) for each marker.

D. The weights of the edges between the camera and each marker in the frame are added to each marker’s respective aggregate error and compared to determine which yields the least overall error. Using the path through the marker that yields the minimum error, (i) the pose of the chosen marker with respect to the reference marker and (ii) the pose of the camera with respect to the chosen marker are combined (through pose composition) to yield the pose of the camera with respect to the reference marker. At this point, the camera is localized with respect to a global coordinate system.

Figure 4.16 shows the output of the Python program as it is running. The information of each frame is separated by the dashed lines. The first piece of information (“through marker”) gives the ID of the marker in the frame to be used for global localization (this marker yields the least aggregate error). The second piece of information (“pose”) gives the coordinates of the camera with respect to the global coordinate system, and also the quaternion representation of the rotation of the camera with respect to the global coordinate system axes. Finally, the last piece of information (“aggregate error”) is self-explanatory.
4.3 Hardware Implementation

The self-localization programs are run concurrently on a 64-bit ASUS laptop computer with a 1.3 GHz Core 2 Duo processor, with 4 GB of RAM. A monocular ICube NS4133BU USB camera was connected to the laptop through a 20 ft USB cable connection. The camera is CMOS-based with a resolution of $1280 \times 1024$ pixels and a frame rate of 25 fps; it is equipped with a C-Mount lens.
Chapter 5

Experiments

5.1 Purpose

The purpose of these experiments is to demonstrate the potential for error reduction in the proposed self-localization algorithm through the use of different error metrics to quantify the edge weights (and to a further extent, the overall localization uncertainty). Each error metric will be implemented, tested, and evaluated based on its ability to reduce global localization error.

5.2 Experimental Setup

The experiments for the proposed localization system were performed in a well-lit indoor environment. The camera was internally calibrated using the process outlined in Section 4.2.1. Markers (with distinct IDs) were used to represent the vertices of the graphs. The HDevelop and Python programs were both initialized and run concurrently to detect markers and perform map-building and localization operations. The useful results obtained were the camera’s position and orientation with respect to the global coordinate system. The system was evaluated in terms of the localization (i.e., positional) accuracy.
5.3 Error Metrics

A few different error metrics were used for quantifying the edge weights of the graph. When the edge connects two markers (as opposed to when it connects the camera and a marker), the function is applied to each marker individually and the two resulting figures are averaged to obtain the weight of the connecting edge. Conversely, when the edge is a connection between the camera and a marker, the function is applied to the marker and the result is halved (to stay consistent with the idea that each marker contributes only half of the total influence on the edge weight).

5.3.1 Reprojection Error Metric

The reprojection error is given by HDevelop when the pose of the marker contour is computed so it does not require implementation. It can be described as the average pixel error between the original marker contour and its reprojection (found using the estimated pose). This error can be directly attributed to the error inherent in the pose estimation.

![Figure 5.1: Visual description of the reprojection error (the pixel distance between the arrows).](image)

5.3.2 Blur Metric

In a blurred image, the edges appear to be more ambiguously defined, as can be seen in Figure 5.2. This results in a reduction of the quality of a pose estimation. This bluriness could be caused by, for example, an unclean camera sensor, ambient lighting effects, distance of an object of interest from the camera, and camera movement.
Figure 5.2: Progressively blurrier image captures of a marker. Note how the edges are less distinctively defined in the blurrier captures.

With the idea that the more blur that occurs in an image of a marker, the worse the quality of the edge detection, and ultimately the worse the pose estimation, a blur metric was chosen as a potential representation of the quality of the pose. The method used was introduced by Crete et al. [9]. As explained by the authors, when an image is compared to a blurred version\(^1\) of itself, it becomes harder to distinguish between the two as the blurriness of the original image goes up. Therefore, the intensity variations between neighbouring pixels (in both the \(x\) and \(y\) directions) are calculated in the original image and the blurred version, and compared with each other. In this way, if the difference between the intensity variations of the images is high, it can be concluded that the original image was sharp, and vice versa when the difference is low.

### 5.3.3 Error Ellipse Volume Metric

When HDevelop computes the pose estimation of a marker contour, it also returns a \(6 \times 6\) covariance matrix of the estimation. This matrix consists of the variances and covariances of the positional and rotational information with respect to each of the three axes (\(x\), \(y\), and \(z\)). The upper-left \(3 \times 3\) sub-matrix represents the positional information and the lower-right \(3 \times 3\) sub-matrix represents the rotational information, while the other two \(3 \times 3\) sub-matrices do not offer information of interest for the purposes of these experiments.

Since only the localization accuracy is being evaluated, only the sub-matrix with the positional information will be considered. The error ellipse volume metric is based on the information obtained from this sub-matrix. The ellipse itself represents a volume of uncertainty related to the marker’s origin position. Therefore, calculating the volume of this ellipse would give an indication of how uncertain the positional information is [26].

\(^1\)A low-pass filter is used to blur the image in the \(x\) and \(y\) directions.
The volume is calculated using the eigenvalues of the positional sub-matrix. The eigenvalues represent the squared lengths of the ellipse dimensions; hence, taking their square roots gives the actual dimensions of the ellipse [2]. Using the formula for the volume of an ellipse, $V = \frac{1}{3} \pi abc$ (where $a$, $b$, and $c$ are the ellipse dimensions), the volume of the error ellipse is found.

![Figure 5.3: An error ellipse representing the positional uncertainty of the marker origin.](image)

### 5.3.4 Perpendicular Distance Metric

The perpendicular distance metric is based on the work of Schweighofer and Pinz [27]. In their paper, they demonstrate that pose ambiguity of planar targets is affected by the change in position and/or orientation of the target in the scene. For instance, the pose ambiguity was found to increase with the distance between the camera and the target. Based on this, a function of the perpendicular distance between the camera and a marker can be used as a metric for the edge weights.

### 5.4 Experiment to Compare the Error Metrics

The following experiment was performed on a local scale (i.e., the reference marker on which the global coordinate system lies was always in the frame).
5.4.1 Procedure

1. A (reference) marker is attached to the wall.

2. The camera is traversed through the room (facing the marker throughout) while the marker detection (HDevelop) and self-localization (Python) algorithms are running. The traversal involves placing the camera in several different locations, varying in the $x$ and $z$ directions, such that the marker is always in the camera’s FOV. This is illustrated in Figure 5.5, where the black dots represent the different positions where the camera is placed.

3. From these known position, frames are captured, the marker is detected, and the self-localization results are recorded.

4. Because only one marker is in use, the marker graph consists of a single vertex with no edges; however, the localization graph still contains the edge connecting the camera and the marker. This edge is weighted based upon the chosen error metric. The weight of the edge varies with the different positions.

5. Each edge weight is compared with its respective position’s ground truth error\(^2\) (in meters) in a graph.

---

\(^2\)The ground truth error is measured manually.
Figure 5.5: An illustration of the different positions that the camera is placed and localization results are recorded for the local scale experiment. The grid starts at a distance of 1.116 m from the marker in the negative z direction, 0.313 m in the y direction and is centered at $x = 0$ m. The dimensions of the grid are $2.508 \times 0.456$ m with the spacing between the dots along the $x$ and $y$ directions being 0.228 m.
5.4.2 Results

Refer to Appendix D.1 for the raw (numerical) data of these results.

Reprojection Error Metric

![Graph of ground truth error vs. the reprojection error metric.](image1)

Figure 5.6: Graph of ground truth error vs. the reprojection error metric.

Blur Metric

![Graph of the ground truth error vs. the blur metric.](image2)

Figure 5.7: Graph of the ground truth error vs. the blur metric.
CHAPTER 5. EXPERIMENTS

Error Ellipse Volume Metric

![Graph of the ground truth error vs. the error ellipse volume metric.](image)

Figure 5.8: Graph of the ground truth error vs. the error ellipse volume metric.

Perpendicular Distance Metric

![Graph of the ground truth error vs. the perpendicular distance metric.](image)

Figure 5.9: Graph of the ground truth error vs. the perpendicular distance metric.
5.4.3 Analysis

Reprojection Error Metric

Referring to Figure 5.6, there appears to be little to no correlation between the ground truth error and the reprojection error. Fitting a line or curve to this data is not plausible.

Reprojection error originates from the estimation error of the internal camera parameters so it does not take into account the environmental factors (e.g., poor lighting, extreme contrast, proximity of the marker to the camera, etc.) that affect the quality of the image. Also, considering the simplicity of the reprojection error, this could explain why it is not encompassing enough to be an effective error metric.

Blur Metric

A counter-intuitive result was encountered with the implementation of the blur metric. As shown in Figure 5.7, as the blur metric increases, the ground truth error decreases. This, along with the weak correlation \( R^2 = 0.4677 \) for a quadratic fit, is enough to preclude this metric from being a viable representation of the ground truth error.

This may be the fault of this particular metric implementation, or it could be that blur is not a suitable indicator for pose estimation quality.

Error Ellipse Volume Metric

The result of the error ellipse volume metric implementation is shown in Figure 5.8. The best fit for the data was found to be a power function, \( y = 12476x^{0.4696} \), which yields a coefficient of determination \( R^2 = 0.7507 \). This appears to be the first reliable relation between the ground truth error and an error metric. One drawback of note is that the data dispersion greatly increases as the metric increases.

The error ellipse, like the reprojection error, originates from the estimation errors of the internal calibration process. However, it does provide more information regarding the nature of the errors in the \( x, y, \) and \( z \) directions, rather than just a scalar quantity like the reprojection error.
Perpendicular Distance Metric

Promising results were obtained from the perpendicular distance metric, as shown in Figure 5.9. Once again, a power function was used as a curve of best fit, this time being \( y = 0.0068x^{2.6913} \) with a coefficient of determination \( R^2 = 0.839 \). Although some dispersion occurs as the metric increases, it is not as pronounced as that of the error ellipse volume metric.

This is an intuitive finding because the smaller the perpendicular distance, the closer the camera is to the marker, which results in a higher resolution image of the marker; therefore, the contour of the marker can be extracted with greater precision, leading to a more accurate pose estimation. However, it is important to note that ambient noise may vary with the position of the camera, causing a skew in the results.

Through comparison with the other tested error metrics, the perpendicular distance metric qualitatively and quantitatively produces the best results. For this reason, it will be used for further investigation in Section 5.5.

5.5 Experiment to Examine the Effect of Implementing an Error Metric on the Global Scale

The following experiment was performed on a global scale (i.e., the reference marker may not be in the frame but it can be reached through a path of other markers in the graph).

5.5.1 Procedure

1. The markers are attached to the wall in the configuration shown in Figure 5.11(a) (see Figure 5.12 for image captures of the configuration).

2. The camera is traversed through the room (facing the markers throughout) while the marker detection and self-localization algorithms are running. As in Section 5.4.1, the traversal involves placing the camera in several different locations, varying in the \( x \) and \( z \) directions. This is illustrated in Figure 5.10, where the black dots represent the different positions where the camera is placed.
3. A localization graph with reference \( R \) is built and updated throughout the traversal by the self-localization algorithm (refer to Figure 5.11(b)); the error metric is used to quantify the edge weights.

4. At the end of the traversal, the camera is positioned such that only marker \( Y \) is visible in its FOV.

5. From this known position, the localization accuracy is compared between the shortest path obtained from the algorithm and the 35 other possible (simple) paths between markers \( Y \) and \( R \).

Figure 5.10: An illustration of the different positions that the camera is placed and localization results are recorded for the global scale experiment. The grid starts at a distance of 0.888 m from the reference marker in the negative \( z \) direction, 0.313 m in the \( y \) direction and with its right edge at \( x = 0 \) m. The dimensions of the grid are 2.508 \( \times \) 0.456 m with the spacing between the dots along the \( x \) and \( y \) directions being 0.228 m.
**Perpendicular Error Metric Function**

As stated, the edge weight quantifier being used for this experiment is the perpendicular distance error metric. It is defined as a function, \( f(d) \), where \( d \) is the distance between the camera and a marker. Three types of functions were tested on the graph: linear \( (f(d) = d) \), quadratic \( (f(d) = d^2) \), and exponential \( (f(d) = 3^d - 1) \). Each function was applied to the 36 possible paths from \( Y \) to \( R \).
5.5.2 Results

Refer to Appendix D.2 for the raw (numerical) data of these results.

Linear Function of the Perpendicular Distance Metric

Figure 5.13: Graph of ground truth error vs. the aggregate error based on the linear function \( f(d) = d \) of the perpendicular distance metric.

Quadratic Function of the Perpendicular Distance Metric

Figure 5.14: Graph of ground truth error vs. the aggregate error based on the quadratic function \( f(d) = d^2 \) of the perpendicular distance metric.
Exponential Function of the Perpendicular Distance Metric

Figure 5.15: Graph of ground truth error vs. the aggregate error based on the exponential function \( f(d) = 3^d - 1 \) of the perpendicular distance metric.

Further experimentation was done by varying the constants of the three functions. An exceptional result was encountered by implementing the exponential function \( f(d) = 50^d - 1 \) as displayed in Figure 5.16.

Figure 5.16: Graph of ground truth error vs. the aggregate error based on the exponential function \( f(d) = 50^d - 1 \) of the perpendicular distance metric.
5.5.3 Analysis

In all three types, the shortest path corresponding to the minimum aggregate error (obtained from the proposed algorithm) yielded the lowest ground truth positional error compared to the 35 other paths (see comparatively large triangular data point in Figures 5.13, 5.14, and 5.15). Furthermore, it was observed that variations of the exponential function showed the best correlation between the aggregate error and the ground truth positional error. Figure 5.16 shows this in the form of a line of best fit with coefficient of determination $R^2 = 0.5473$, when using the exponential function $f(d) = 50^d - 1$ as an error metric. The bottom left-most data point (again, the large triangle) graphically shows how the minimum aggregate error corresponds to the minimum ground truth positional error of the 36 paths.

5.5.4 Limitations

The experiment was repeated using the same exponential error function, except the three lower markers were moved closer to the camera and tilted slightly horizontally. In this case, the minimum ground truth error corresponded to the third least aggregate error and there was a negative correlation (with coefficient of determination $R^2 = 0.2118$) between the aggregate and ground truth errors. This indicates that this particular error function is not suitable for all configurations of markers.
Chapter 6

Conclusion

6.1 Summary of Contributions

A proposed topological mapping approach is applied to a self-localization system for the purpose of reducing global error. The map is incrementally built and updated as the camera moves through the environment. A shortest path algorithm is applied to the map to find the path of least aggregate error based on an appropriate error metric. Experiments were done using error metrics based on reprojection error, blur, error ellipse volume, and the perpendicular distance between the camera and a marker. The results demonstrate the effectiveness of the system in reducing global error but emphasize the importance of deriving an appropriate error metric for the edge weights. To the best of the author’s knowledge, the topological graph approach has not been applied to a single-sensor self-localization system for the purpose of reducing global error.

6.2 Future Work

The proposed approach opens up possibilities for creating more suitable error metrics in future systems. Although the perpendicular distance yielded some favorable results, it was not robust in all marker configurations. More complex error metrics can be created by trying different, weighted, combinations of simpler error metrics. Furthermore, this system can be implemented onto an autonomous vehicle to examine the effects that velocity has on the quality of the image.
In the computer vision community, research is being done on accurately modeling and studying the effects of ambient noise in an image; these models can be studied and implemented into localization systems, such as the one proposed herein, to improve the accuracy of self-localization systems.
Appendix A

Glossary of Terms

**artificial landmark**
A custom designed object with distinct features that make it easier to distinguish from its surrounding environment. It is added to the scene rather than being a naturally occurring element of the environment.

**augmented reality**
A field of research in computer science that augments real world scenes/images with computer-generated objects. It requires that the pose of the camera is known with respect to the real world landmarks on which the virtual objects are to be superimposed.

**external (or extrinsic) calibration**
The process of determining the 3D Euclidean transformation between the camera coordinate system and the global coordinate system.

**FOV**
Field of view: The observable region of a camera at a particular position and orientation.

**global coordinate system**
The single coordinate system by which all objects are referenced (i.e. a 3D world coordinate system). In the context of this work, it is the coordinate system whose origin lies in the center of the reference marker, with its $x$- and $y$- axes parallel and $z$-axis perpendicular
global error
Error which arises from the accumulation of local error and affects pose estimations made with respect to a global coordinate system (which may not necessarily be in the image).

IDE
Integrated Development Environment: A programming environment that includes a code editor, a compiler, and a debugger.

internal (or intrinsic) calibration
The process of determining the internal parameters – such as focal length, pixel cell dimensions, optical center, etc. – of a camera.

local coordinate system
A coordinate system located in the current FOV of the camera. In the context of this work, it is a coordinate system whose origin lies in the center of a marker in the image (not necessarily the reference marker). Its $x$- and $y$- axes are parallel and its $z$-axis is perpendicular to the marker’s surface.

local error
Error which arises from noise in the image capture and affects pose estimations made with respect to coordinate systems in the image (i.e., a local coordinate system).

localization graph
Essentially, a marker graph that includes the camera as an additional node. Edges between the camera and the markers represent the existence of a relative pose between them. As in the marker graph, edges are weighted based on an error metric that suitably represents pose uncertainty.

marker
See artificial landmark.
**marker graph**
A representation of a set of markers as a weighted, undirected graph, where the markers are the nodes and the edges represent the existence of a relative pose between any two of them. Edge weights are quantified based on an error metric that suitably represents pose uncertainty.

**pose**
A pose $P_{\alpha\beta}$ is a rigid three dimensional Euclidean transformation from the coordinate system of object $\alpha$ to the coordinate system of object $\beta$.

**reference marker**
A marker in the scene that is chosen to represent the location of the global coordinate system (i.e., the origin of the global coordinate system is in the center of the marker).

**reprojection error**
The average distance (in pixels) between a measured point and its reprojection (found using the estimated pose).

**scene**
The 3D world space.

**SLAM**
Simultaneous Localization and Mapping: A technique used by autonomous vehicles or robots to (1) build/update a map with/without a priori knowledge of the environment and (2) localize themselves within that map.

**socket**
A mechanism used by computer ports that allows communication between two applications.
Appendix B

HDevelop Program Source Code

The HDevelop source code consists of the main program, a function for finding the corners of a quadrilateral contour (called \texttt{get_quad_corners}), a function that determines if a contour is inside another contour (called \texttt{is_quad_inside}), and a function that calculates cross-ratio (called \texttt{calc_cross_ratio}).

B.1 Main Program

*************** parameters ***************
* image contrast (emphasize()) mask size
 emsize := 31
* variable for determining which type of threshold method to use
 thresstype := 0
* threshold for judging between black and white
 thresh_bw := 85
* side length of marker
 marklen := 0.143
* true cross-ratio of marker
 true_cr := 1.06921835
* threshold for the cross-ratio of a marker
 thresh_cr := 0.02
************** end of parameters ***************

dev_update_off ()
dev_close_window ()
APPENDIX B. HDENVELOP PROGRAM SOURCE CODE

* folder paths
normcaps_path: = 'C:/experiments/ims_of_exp_6_b/run1/
par_path: = 'C:/params/
* open socket and connect it to an accepting socket
open_socket_connect ('localhost', 5678, ['timeout', 'protocol'],
                     ['infinite', 'TCP'], osock)
* read camera’s intrinsic parameters
read_cam_par (par_path+'cube_intrinsic_param.cal', CamParIn)
* fix camera’s intrinsic parameters for radial distortion
change_radial_distortion_cam_par ('fixed', CamParIn, 0, CamParOut)
* create fitting window
read_image (Image, normcaps_path+'01.png')
dev_open_window_fit_image (Image, 0, 0, -1, -1, WindowHandle)

* loop through the set of images
for index := 1 to 40 by 1
  * read image
    read_image (Image, normcaps_path+index$'.02'+'.'png')
  * convert to gray image
    rgb1_to_gray (Image, Image)
  * enhance the contrast of the image
    emphasize (Image, Image, emsize, emsize, 1)
  * remove radial distortion
    change_radial_distortion_image (Image, Image, Image, CamParIn,
                                    CamParOut)
  * if pose(s) of marker(s) cannot be found, go to next frame
    try
      * segment image using one of the threshold methods
        bin_threshold (Image, Region)
      * separate the region into connected parts
        connection (Region, ConnectedRegions)
      * filter out non-marker regions by area
        select_shape (ConnectedRegions, SelectedRegions1, 'area',
                      'and', 150, 300000)
      * convert remaining regions to contours
        gen_contour_region_xld (SelectedRegions1, Contours1,
                                 'border_holes')
      * get dimensions of image
        get_image_size (Image, ImgWidth, ImgHeight)
* select only the contours within a certain range of length
select_shape_xld (Contours1, Contours2, 'contlength', 'and', \100, 2*ImgWidth + 2*ImgHeight)

* count the number of contours
count_obj (Contours2, NumContours)

* initialize marker contours and their cross-ratio errors
gen_empty_obj (MarkConts)
CRErrs := []

* loop to take care of contours depending on their cross-ratio
i := 1
while (i <= NumContours - 1)

  * find 4 corners of outer quadrilateral contour
  get_quad_corners (Contours2, i, ImgWidth, ImgHeight, \OuterRow, OuterCol, FoundOuter)

  * find 4 corners of inner quadrilateral contour
  get_quad_corners (Contours2, i+1, ImgWidth, ImgHeight, \InnerRow, InnerCol, FoundInner)

  * if either quadrilateral (outer/inner) not found, go to
  * next contour in loop
  if (not (FoundOuter and FoundInner))
    i := i + 1
    continue
  endif

  * check if inner quad. is inside outer quad.
  is_quad_inside (OuterRow, OuterCol, InnerRow, InnerCol, \QuadInside)

  * if not, go to next contour in loop
  if (not QuadInside)
    i := i + 1
    continue
  endif

  * assert that the inner and outer corners are sorted in
  * the same manner for proper cross-ratio calculation
  angle_ll (OuterRow[0], OuterCol[0], OuterRow[1], \OuterCol[1], InnerRow[0], InnerCol[0], \InnerRow[1], InnerCol[1], Angle)
  if (abs(Angle) > 0.0873)
    TempRow := InnerRow
    TempCol := InnerCol
  endif
if (Angle > 0)
    InnerRow[0] := TempRow[3]
    InnerRow[1] := TempRow[0]
    InnerCol[0] := TempCol[3]
    InnerCol[1] := TempCol[0]
else
    InnerRow[0] := TempRow[1]
    InnerRow[3] := TempRow[0]
    InnerCol[0] := TempCol[1]
    InnerCol[3] := TempCol[0]
endif
angle_ll (OuterRow[0], OuterCol[0], OuterRow[1], OuterCol[1],
          InnerRow[0], InnerCol[0], InnerRow[1], InnerCol[1], NewAngle)
if (abs(NewAngle) > 0.0873)
i := i + 1
continue
endif
* calculate the avg. cross-ratio errors of both diagonals
* of a marker
A_r0 := OuterRow[0]
A_c0 := OuterCol[0]
B_r0 := InnerRow[0]
B_c0 := InnerCol[0]
C_r0 := InnerRow[2]
C_c0 := InnerCol[2]
D_r0 := OuterRow[2]
D_c0 := OuterCol[2]
calc_cross_ratio (A_r0, A_c0, B_r0, B_c0, C_r0, C_c0,\
                  D_r0, D_c0, cr0)
APPENDIX B. HDEVELOP PROGRAM SOURCE CODE

A_r1 := OuterRow[1]
A_c1 := OuterCol[1]
B_r1 := InnerRow[1]
B_c1 := InnerCol[1]
C_r1 := InnerRow[3]
C_c1 := InnerCol[3]
D_r1 := OuterRow[3]
D_c1 := OuterCol[3]
calc_cross_ratio (A_r1, A_c1, B_r1, B_c1, C_r1, C_c1,\nD_r1, D_c1, cr1)
error_cr0 := abs(cr0 - true_cr)
error_cr1 := abs(cr1 - true_cr)
error_cr := (error_cr0 + error_cr1)/2
* if the avg. cross-ratio error exceeds the threshold,
* go to next contour
if (error_cr >= thresh_cr)
  i := i + 1
  continue
endif
* add the remaining contours to the tuple of viable
* markers
select_obj (Contours2, ObjSel, i)
concat_obj (MarkConts, ObjSel, MarkConts)
* store each frame marker's cross-ratio error in a tuple
CRErrs := [CRErrs, error_cr]
* skip the next (inner) contour, and go to the next outer
* one
  i := i + 2
endwhile
* if no desirable contours are detected toggle threshold method
* and go to next frame
count_obj (MarkConts, NumMarkConts)
if (NumMarkConts = 0)
  * if no contours found, toggle threshold method, send a
  * string notifying the python prog. of exception, and go
  * to next frame of vid capture when a response is received
  threshtype := threshtype xor 1
  send_data (osock, 'z', 'no markers found', [])
  continue
endif
* estimate the poses of the markers using the outer square
* borders of each marker
try
    get_rectangle_pose (MarkConts, CamParOut, marklen,\
        marklen, 'nonweighted', 2, PosesArray,\
        CovPoses, RPErrs)
* if pose estimation fails, take out the markers that caused
* the failure and only use the good ones
catch(Exception1)
    gen_empty_obj (MarkConts2)
    for i := 1 to NumMarkConts by 1
        try
            select_obj (MarkConts, ObjSel2, i)
            get_rectangle_pose (ObjSel2, CamParOut, marklen,\
                marklen, 'nonweighted', 2,\
                TempPA, TempCP, TempE)
            concat_obj (MarkConts2, ObjSel2, MarkConts2)
        catch(Exception2)
            continue
        endtry
    endfor
    count_obj (MarkConts2, NumMarkConts2)
    if (NumMarkConts2 > 0)
        get_rectangle_pose (MarkConts2, CamParOut, marklen,\
            marklen, 'nonweighted', 2,\
            PosesArray, CovPoses, RPErrs)
    else
        * if exception encountered, toggle threshold method,
        * send a string notifying the python prog. of exception,
        * and go to next frame of vid capture when a response
        * is received
        threshtype := threshtype xor 1
        send_data (osock, 'z', 'no markers found', [])
        continue
    endif
endtry
catch (Exception3)
    * if exception encountered, toggle threshold method, send a
* string notifying the python prog. of exception, and go to
* next frame of vid capture when a response is received

thresh_type := thresh_type xor 1
send_data (osock, 'z', 'no markers found', [])
continue

end try
* initialize the number of markers in the cf (current frame)
num_markers_CF := |PosesArray|/7
* initialize all arrays to store the properties of the markers in
* the cf
tuple_gen_const (0, 0, MarkerIDs)
tuple_gen_const (0, 0, HomMatsArray)
tuple_gen_const (0, 0, PosesArray1)
* loop to go through every marker in the frame
for i := 0 to num_markers_CF−1 by 1
* select pose of current marker
tuple_select_range (PosesArray, 7*i, 7*i + 6, Pose)
* move origin to the upper left corner of the marker
set_origin_pose (Pose, −marklen/4, −marklen/4, 0, Pose)
* refine the image so the marker is displayed with no persp.
* distortion
image_to_world_plane (Image, ImageRefined, CamParOut, Pose, \1000*(marklen/2), 1000*(marklen/2), \"mm", \"none")
* enhance the contrast of the marker to make dark/light
* regions darker/lighter
scale_image_max (ImageRefined, ImageRefined)
* generate the regions for code cell sampling
gen_rectangle1 (RectTL, 7, 7, 16, 16)
gen_rectangle1 (RectTM, 7, 32, 16, 41)
gen_rectangle1 (RectTR, 7, 56, 16, 65)
gen_rectangle1 (RectML, 31, 7, 40, 16)
gen_rectangle1 (RectMM, 31, 32, 40, 41)
gen_rectangle1 (RectMR, 31, 56, 40, 65)
gen_rectangle1 (RectBL, 55, 7, 64, 16)
gen_rectangle1 (RectBM, 55, 32, 64, 41)
gen_rectangle1 (RectBR, 55, 56, 64, 65)
gen_empty_obj (RegionsSampling)
concat_obj (RectBR, RegionsSampling, RegionsSampling)
concat_obj (RectBM, RegionsSampling, RegionsSampling)
concat_obj (RectBL, RegionsSampling, RegionsSampling)
concat_obj (RectMR, RegionsSampling, RegionsSampling)
concat_obj (RectMM, RegionsSampling, RegionsSampling)
concat_obj (RectML, RegionsSampling, RegionsSampling)
concat_obj (RectTR, RegionsSampling, RegionsSampling)
concat_obj (RectTM, RegionsSampling, RegionsSampling)
concat_obj (RectTL, RegionsSampling, RegionsSampling)

* calculate the mean gray value in each cell of the marker
* code

intensity (RegionsSampling, ImageRefined, MeanGray, \
    DeviationGray)

* convert gray values of code to binary code
tuple_gen_const (9, 0, CodeTemp)
for j := 0 to |CodeTemp|−1 by 1
    if (MeanGray[j] < thresh_bw)
        CodeTemp[j] := 1
    endif
endfor

* display the image
dev_display (Image)

* bring coord. system origin back to center of marker
set_origin_pose (Pose, marklen/4, marklen/4, 0, Pose)

* convert marker’s pose to hom. 3d matrix so transformations
* can be applied
pose_to_hom_mat3d (Pose, HomMat3D)

* create an empty tuple for the true code of the marker
tuple_gen_const (9, 0, Code)

* rearrange the temp. code & transform the temp. coord.
* system to find the true code and coord. system orientation
if (CodeTemp[0] = 0)
    Code := CodeTemp
elseif (CodeTemp[2] = 0)
    for j := 0 to 2 by 1
        for k := 0 to 2 by 1
            Code[3*j+k] := CodeTemp[2+3*k−j]
        endfor
    endfor
hom_mat3d_rotate_local (HomMat3D, 1.57079, ‘z’, HomMat3D)
else if (CodeTemp[6] = 0)
    for j := 0 to 2 by 1
        for k := 0 to 2 by 1
            Code[3*j+k] := CodeTemp[6−3*k+j]
        endfor
    endfor
    hom_mat3d_rotate_local (HomMat3D, −1.57079, 'z', HomMat3D)
else
    tuple_inverse (CodeTemp, Code)
    hom_mat3d_rotate_local (HomMat3D, 3.14159, 'z', HomMat3D)
endif
* convert the transformed hom. 3d matrix back to a pose
hom_mat3d_to_pose (HomMat3D, Pose)
* convert the code from tuple to decimal, let the result be
* the marker ID
MarkerID := 0
for j := 0 to 8 by 1
    MarkerID := MarkerID + Code[8−j]*pow(2, j)
endfor
* append current marker’s ID and pose to arrays
tuple_concat (MarkerIDs, MarkerID, MarkerIDs)
tuple_concat (HomMatsArray, HomMat3D, HomMatsArray)
tuple_concat (PosesArray1, Pose, P o s e s A r r a y 1)
endfor
* display the coordinate systems of the markers
for i := 0 to NumMarkersCF−1 by 1
    Pose := P o s e s A r r a y 1[7*i:7*i+6]
    disp_3d_coord_system (WindowHandle, CamParOut, Pose, \marklen/2)
endfor
* convert marker IDs tuple to integers
tuple_int (MarkerIDs, MarkerIDs)
* convert marker IDs and hom mats tuples to arrays of strings
tuple_string (MarkerIDs, '.1d', MarkerIDsStrings)
tuple_string (HomMatsArray, '.8f', HomMatsStrings)
tuple_string (RPErrs, '.8f', RPErrsString)
* create error metric that depends on z distance and convert to
* array of strings
for i := 0 to NumMarkersCF−1 by 1
ZDistErrs[i] := HomMatsArray[11+12*i]
endfor
tuple_string (ZDistErrs, '.8f', ErrsString)
* create markerID/pose string to send through socket
* format: markerID[0]:pose[0]|error[0];....
outstring := '
for i := 0 to NumMarkersCF-1 by 1
    outstring := outstring + MarkerIDsStrings[i] + ':'
    for j := 0 to 11 by 1
        outstring := outstring + HomMatsStrings[12*i+j]
        if (j = 11 and i ≠ NumMarkersCF-1)
            outstring := outstring + '|' + ErrsString[i] + ';
        elseif (j = 11 and i = NumMarkersCF-1)
            outstring := outstring + '|' + ErrsString[i]
            break
        else
            outstring := outstring + ',',
        endif
    endfor
endfor
* send string (data) through socket, wait for notification
* from other end that the program has processed the data and
* is waiting for the next string
try
    send_data (osock, 'z65536', outstring, [])
catch(Exception4)
    break
endtry
endfor
* close socket
close_socket (osock)
B.2 Function: *get_quad_corners*

* FINDING THE 4 CORNERS OF A QUADRILATERAL

* select the desired contour
select_obj (Contours, ContSel, ContInd)
* make the contour convex
shape_trans_xld (ContSel, XLDTrans, 'convex')
* convert the contour to a polygon
gen_polygons_xld (XLDTrans, Polygon, 'ramer', 1)
* get polygon vertices, line lengths, and line angles
get_polygon_xld (Polygon, PolyRow, PolyCol, Len, Phi)

* if there are less than 4 lines, no quad was found, return
if (|Len| < 4)
    Found := 0
    return ()
endif

* sort the lines by length in descending order
tuple_sort_index (Len, LenSrtdInd)
tuple_inverse (LenSrtdInd, LenSrtdInd)

* take care of lines defined by the image boundaries
for i := 0 to |PolyRow|−2 by 1
    row0 := PolyRow[i]
    row1 := PolyRow[i+1]
    col0 := PolyCol[i]
    col1 := PolyCol[i+1]
    on_boundary := (row0 <= 0.5 and row1 <= 0.5) or (
        row0 >= ImgHeight−0.5 and row1 >= ImgHeight−0.5)\n        or (col0 <= 0.5 and col1 <= 0.5) or
        (col0 >= ImgWidth−0.5 and col1 >= ImgWidth−0.5)
    * if one of the lines of the polygon is defined by a boundary
    if (on_boundary)
        * choice 1: return with no quad. found
        Found := 0
        return ()
        * choice 2: remove the boundary line and continue
for j := 0 to |LenSrtInd|−1 by 1
  if (LenSrtInd[j] = i)
    tuple_remove (LenSrtInd, j, LenSrtInd)
    break
  endif
endfor
endfor

uncomment if you want to see the sorted line lengths and angles (phi)
for i := 0 to |LenSrtInd|−1 by 1
  PhiSrtd[i] := Phi[LenSrtInd[i]]
endfor

get the 4 longest lines and sort them in order of occurrence
TopLenSrtInd := LenSrtInd[0:3]
tuple_sort (TopLenSrtInd, TopLenSrtInd)

loop to assert that no 2 of the 4 lines lie on the same side
of the quadrilateral (by calculating the angular difference
between consecutive lines)
while (1)
done := 1
  for i := 0 to 3 by 1
    if (i = 3)
      PhiDiff := abs(Phi[TopLenSrtInd[i]]−\
                      Phi[TopLenSrtInd[0]])
    else
      PhiDiff := abs(Phi[TopLenSrtInd[i]]−\
                      Phi[TopLenSrtInd[i+1]])
    endif
    if (PhiDiff <= 0.7854 or PhiDiff >= 2.3562)
      if (i = 3)
        if (Len[LenSrtInd[i]] < Len[LenSrtInd[0]])
          tuple_remove (LenSrtInd, i, LenSrtInd)
        else
          tuple_remove (LenSrtInd, 0, LenSrtInd)
        endif
      else
        tuple_remove (LenSrtInd, i, LenSrtInd)
      endif
    endif
  endforeach
while (1)
APPENDIX B. HDEVELOP PROGRAM SOURCE CODE

```plaintext
end if
else
    if (Len[LenSrtdInd[i]] < Len[LenSrtdInd[i+1]])
        tuple_remove (LenSrtdInd, i, LenSrtdInd)
    else
        tuple_remove (LenSrtdInd, i+1, LenSrtdInd)
    endif
endif
endif
done := 0
endif
if (done = 0)
    break
endif
done := 0
for i := 0 to 3 by 1
    RowA1 := PolyRow[TopLenSrtdInd[i]]
    ColA1 := PolyCol[TopLenSrtdInd[i]]
    RowA2 := PolyRow[TopLenSrtdInd[i]+1]
    ColA2 := PolyCol[TopLenSrtdInd[i]+1]
    if (i = 3)
        ...
    endif
else
    TopLenSrtdInd := LenSrtdInd[0:3]
    tuple_sort (TopLenSrtdInd, TopLenSrtdInd)
endif
endwhile

* if there are less than 4 lines, no quad was found, return
if (|LenSrtdInd| < 4)
    Found := 0
    return()
* otherwise, quad was found
else
    Found := 1
endif

* find the 4 corners of the quadrilateral by finding the intersection
* points of the 4 lines
for i := 0 to 3 by 1
    RowA1 := PolyRow[TopLenSrtdInd[i]]
    ColA1 := PolyCol[TopLenSrtdInd[i]]
    RowA2 := PolyRow[TopLenSrtdInd[i]+1]
    ColA2 := PolyCol[TopLenSrtdInd[i]+1]
    if (i = 3)
        ...
    endif
```

The above code snippet includes conditional logic and loops for processing line segments and determining the intersection points for quadrilateral corners. It also includes a check for the number of lines to determine whether a quad was found and to return or continue processing.
APPENDIX B. HDEVELOP PROGRAM SOURCE CODE

RowB1 := PolyRow[TopLenSrtdInd[0]]
ColB1 := PolyCol[TopLenSrtdInd[0]]
RowB2 := PolyRow[TopLenSrtdInd[0]+1]
ColB2 := PolyCol[TopLenSrtdInd[0]+1]

else
RowB1 := PolyRow[TopLenSrtdInd[i+1]]
ColB1 := PolyCol[TopLenSrtdInd[i+1]]
RowB2 := PolyRow[TopLenSrtdInd[i+1]+1]
ColB2 := PolyCol[TopLenSrtdInd[i+1]+1]
endif

intersection_l1 (RowA1, ColA1, RowA2, ColA2, RowB1, ColB1, RowB2, ColB2, RowInt, ColInt, _)

IntrsRow[i] := RowInt
IntrsCol[i] := ColInt

endfor

return ()
B.3 Function: *is_quad_inside*

* DETERMINING WHETHER A QUADRILATERAL IS INSIDE A CONVEX QUADRILATERAL
* see http://paulbourke.net/geometry/insidepoly/
* under "Solution 3 (2D)" for full explanation

* loop through every point
for j := 0 to 3 by 1
  * initially, contour and point are assumed to be outside
  QuadInside := 0
  PtInside := 0
  * the test point
  x := InnerCol[j]
  y := InnerRow[j]
  * loop through every line
for k := 0 to 3 by 1
  * the test line
  x_0 := OuterCol[k]
  y_0 := OuterRow[k]
  if (k = 3)
    x_1 := OuterCol[0]
    y_1 := OuterRow[0]
  else
    x_1 := OuterCol[k+1]
    y_1 := OuterRow[k+1]
  endif
  * the formula to determine whether pt in inside or outside the contour
  f := (y - y_0)*(x_1 - x_0) - (x - x_0)*(y_1 - y_0)
  * if pt not inside, exit line loop
  if (f >= 0)
    PtInside := 0
    break
  else
    PtInside := 1
  endif
endfor
* if pt not inside, contour is not inside, exit pt loop
* otherwise, contour is inside
if (not PtInside)
    QuadInside := 0
    break
else
    QuadInside := 1
    endif
endif
endfor

return ( )
B.4 Function: calc_cross_ratio

* CALCULATING CROSS RATIO OF 4 PTS ON A LINE

* calculating line lengths between points
distance_pp (A_r, A_c, C_r, C_c, AC)
distance_pp (B_r, B_c, C_r, C_c, BC)
distance_pp (A_r, A_c, D_r, D_c, AD)
distance_pp (B_r, B_c, D_r, D_c, BD)

* calculating cross ratio
CrossRatio := (AC/BC)/(AD/BD)

return ()
Appendix C

Python Program Source Code

```python
import socket
import argparse
from msvcrt import kbhit, getch
from sets import Set

from adolphus.geometry import Pose, Point, Rotation
from hypergraph.core import Edge, Graph
from hypergraph.path import dijkstra
from hypergraph.connectivity import connected

# importing libs for 3d plot
from mpl_toolkits.mplot3d import Axes3D
import matplotlib.pyplot as plt

# parse incoming string data from halcon into python dictionary
# format
def parse_from_halcon(hstring):
    """
    Convert tuple data in string format from HALCON into the camera ID and target pose.

    @param hstring: the string data from HALCON.
    @type hstring: C{str}
    @return: Camera ID and target pose.
    @rtype: C{str}, L{Pose}
```

78
APPENDIX C. PYTHON PROGRAM SOURCE CODE

```
frame_markers = {}
try:
    for pair in hstring.split('; '):
        pose = pair.split(': ')[1].split(',
        weight = float(pose[len(pose)-1].split('|')[0].pop())
        pose[len(pose)-1] = pose[len(pose)-1].split('|')[0]
        for i in range(len(pose)):
            pose[i] = float(pose[i])
        frame_markers.update({int(pair.split(': ')[0]):
            {"pose": pose, 'weight': weight}})
    for marker in frame_markers:
        trans_vec = [frame_markers[marker]['pose'][3], \
                     frame_markers[marker]['pose'][7], \
                     frame_markers[marker]['pose'][11]]
        rot_mat = [[] for x in range(3)]
        for i in range(3):
            rot_mat[i] = [frame_markers[marker]['pose'][4*i], \
                          frame_markers[marker]['pose'][4*i+1], \
                          frame_markers[marker]['pose'][4*i+2]]
        t = Point(trans_vec)
        r = Rotation.from_rotation_matrix(rot_mat)
        pose = Pose(t, r)
        frame_markers[marker]['pose'] = pose
except:
    pass
return frame_markers

def main():
    # parse command line arguments for reference marker id
    parser = argpase.ArgumentParser\
        (description='Choose reference marker. ')
    parser.add_argument('ref_id', type=int, \
        help='ID of the reference marker')
    ui = parser.parse_args()
    # set up network socket
    sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    sock.bind(('localhost', 5678))
```
sock.listen(20)
# accept incoming connection requests
channel, details = sock.accept()
# create a text file for poses
filename = 'voyager_poses.txt'
FILE = open(filename, 'w')
try:
    # initialize empty graph and the global relative poses
    # (GRPs)
    graph = Graph()
global_relposes = {}
    # data is initially not recorded to file
    rec = 0
    # variable to tell if new map was calculated
    newmap = 0
    # initialize global poses of markers
    gmarkposes = {}
    # x, y, and z points to be plotted in graph
    x_arr = []
y_arr = []
z_arr = []
    # loop to create graph and find edge poses
    while(True):
        # if a key is pressed
        if kbhit():
            # catch the key
            key = ord(getch())
            # 'q' is pressed: exit mapping/localization loop
            if key == 113:
                break
            # 'r' is pressed: toggle recording of data to file
            elif key == 114:
                rec = rec ^ 1
            # 's' is pressed: give update on registered and
            # unconnected markers
            elif key == 115:
                print 'STATUS:
                print len(graph.vertices),
                   'Registered marker(s):'
list(graph.vertices)
# if reference marker is registered
if ui.ref_id in graph.vertices:
    if 'prev' in locals():
        # find unconnected markers
        uc_marks = [um for um in prev
                     if um != ui.ref_id and
                     prev[um] == None]
        print len(uc_marks),
        'Unconnected marker(s):',
        uc_marks, '\n\n'
    else:
        uc_marks = [um for um in
                     graph.vertices if
                     um != ui.ref_id]
        print len(uc_marks),
        'Unconnected marker(s):',
        uc_marks, '\n\n'
else:
    print 'Ref marker is not registered.\n\n'
# parse string from halcon
frame_markers = {}
hsstring = channel.recv(65536)
if not hstring or hstring == 'no markers found':
    pass
else:
    frame_markers = parse_from_halcon(hstring)

UPDATE MAP (IF NECESSARY)

# update the set of vertices in the graph with the
# local markers
graph.vertices.update(frame_markers)
# find the local relative poses (LRPs) of the
# markers w.r.t. one another and use them ...
# ... to update the GRP, the graph, and possibly...
# the shortest path
local_repose = {}
restricted_e = []  # Set([]),
for marker in frame_markers:
    marker_a = frame_markers[marker]
    for other in frame_markers:
        if Set([marker, other]) in restricted_e:
            continue
        try:
            try:
                assert other != marker
                assert (marker, other) not in \local_repose
            except AssertionError:
                continue
            marker_b = frame_markers[other]
            pose_ab = marker_a['pose'] -\ marker_b['pose']
            pose_ab = -pose_ab
            weight_ab = (marker_a['weight'] +\ marker_b['weight'])/2.0
            weight_ab = weight_ab
            local_repose = [(marker, other)] =\
                {'pose': pose_ab, \'weight': weight_ab}
            local_repose = [(other, marker)] =\
                {'pose': pose_ab, \'weight': weight_ab}

# if the edge joining the two markers
# does not exist in neither the GRPs nor
# the graph, add it to both from the LRP,
# and calculate shortest paths (prev)
if (marker, other) not in global_repose:
    global_repose = [(marker, other)] =\local_repose
    [(marker, \other)]
    global_repose = [(other, marker)] =\
local_relposes
[(other,\n  marker)]

graph.add_edge(Edge((marker, other)),\n  local_relposes[[(marker,\n    other)]\n
  ['weight'])

# if reference is registered,
# calculate shortest path
if ui.ref_id in graph.vertices:
  prev = dijkstra(graph, ui.ref_id)
  newmap = 1
  # however, if it already exists and the
  # weight of the new one is less than the
  # existing one, replace the existing ones
  # in the GRPs and graph with the new one,
  # and calculate shortest paths (prev)
else:
  if weight_ab < global_relposes\n[(marker, other)]['weight']:
    global_relposes[(marker,\n      other)] =\n    local_relposes\n[(marker, other)]
    global_relposes[(other,\n      marker)] =\n    local_relposes\n[(other, marker)]

graph.weights\n[Edge((marker,\n    other))]]\n  = weight_ab

# if reference is registered,
# calculate shortest path
if ui.ref_id in graph.vertices:
  prev = dijkstra(graph,\n    ui.ref_id)

  newmap = 1

except:
pass
# if a new map was calculated
if newmap == 1:
    # make it the old map
    newmap = 0
    # empty old gmarkposes (if not already empty)
    gmarkposes = {}
    # find the global pose (and its associated
    # aggregate weight) of each marker in the map
    # w.r.t. the reference marker
    for marker in prev:
        pose_comp = Pose()
        agg_weight = 0.0
        curr_i = marker
        prev_i = prev[curr_i]
        # skip markers that are disconnected from
        # the reference
        if not prev_i and curr_i != ui.ref_id:
            continue
        # loop until reference is reached
        while prev_i:
            edge_i = (curr_i, prev_i)
            pose_comp += global_relposes[edge_i][
                'pose'
            ]
            agg_weight += global_relposes[edge_i][
                'weight'
            ]
            curr_i = prev_i
            prev_i = prev[curr_i]
        # store the global pose and weight of the
        # marker
        gmarkposes[marker] = {'pose': pose_comp,
            'weight': agg_weight}

LOCALIZE CAMERA

# find the camera pose that yields the minimum
# aggregate weight of all markers in the frame
bestmarker = None
bestpose = None
minweight = float('inf')
for marker in frame_markers:
    if marker in gmarkposes:
        marker_i = frame_markers[marker]
        gcampose = -marker_i['pose'] +
                   gmarkposes[marker]['pose']
        gcamweight = marker_i['weight'] +
                   gmarkposes[marker]['weight']
        if gcamweight < minweight:
            bestmarker = marker
            bestpose = gcampose
            minweight = gcamweight
    # case of only reference beings seen
    elif marker == ui.ref_id and
          len(frame_markers) == 1:
        marker_i = frame_markers[marker]
        bestmarker = marker
        bestpose = -marker_i['pose']
        minweight = marker_i['weight']
    else:
        pass
    # if pose is available, print it
if (bestpose):
    print '__________________________'
    print 'through marker: ', bestmarker
    print 'pose: ', bestpose
    print 'aggregate error: ', minweight
    print '__________________________'
    # if record toggle is on and pose is available, # record the poses to file
if rec == 1:
    if (bestpose):
        temp = str(bestpose.T).split(',
bestpose_str = temp[0][1:] + '\t' +
                  temp[1][1:] + '\t' +
                  temp[2][1:-1]
        FILE.write(bestpose_str+'\n')
```python
x_arr.append(float(temp[0][1:]))
y_arr.append(float(temp[1][1:]))
z_arr.append(float(temp[2][1:]-1))

# write the graph information to file
FILE.write('\n−−−−−− GRAPh −−−−−−\n\n\n')
FILE.write(str(graph)+'\n\n')
# write title of section (marker poses) to file
FILE.write('−−−−−− MARKER POSES −−−−−−\n\n\n')
# see if prev exists
if 'prev' in locals():
    # loop through markers in prev
    for marker in prev:
        try:
            # write global pose to file
            FILE.write(str(marker)+':'+str(gmarkposes[marker])+\'
\n')
            except:
                pass
            FILE.write('−−−−−−−−−−−−\n\n\n')
        # print prev #
        FILE.write('−−−− PREV −−−−\n\n\n')
        FILE.write(str(prev)+'\n\n')
finally:
    # plot the poses
    fig = plt.figure()
    ax = fig.add_subplot(111, projection='3d')
ax.scatter(x_arr, y_arr, z_arr, c='r', marker='o')
ax.set_xlabel('x-axis')
ax.set_ylabel('y-axis')
ax.set_zlabel('z-axis')
ax.set_xlim(0, 2)
ax.set_ylim(-2.4, 1)
ax.set_zlim(-3, 0.5)
plt.show()
# close text doc, channel, and socket
FILE.close()
channel.close()
sock.close()
print '−−−− CONNECTION CLOSED (PYTHON) −−−'
if __name__ == '__main__':
    main()
Appendix D

Data Tables

D.1 Ground Truth Error Vs. Error Metrics

Table D.1 shows the raw data for the plotted points in Figures 5.6, 5.7, 5.8, and 5.9. The first four columns show the data for the graphs’ respective horizontal axes, and the last column shows the ground truth error, which is the data for the vertical axes of the graphs.

Table D.1: Raw data for the Ground Truth Error Vs. Error Metric figures.

<table>
<thead>
<tr>
<th>Reprojection Error (pixels)</th>
<th>Blur</th>
<th>Error Ellipse Volume</th>
<th>Perpendicular Distance (m)</th>
<th>Ground Truth Error (m)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.244016</td>
<td>0.306483</td>
<td>$5.77219 \times 10^{-14}$</td>
<td>1.1108</td>
<td>0.01606518</td>
</tr>
<tr>
<td>0.260119</td>
<td>0.306405</td>
<td>$1.92482 \times 10^{-13}$</td>
<td>1.3469</td>
<td>0.022000227</td>
</tr>
<tr>
<td>0.340079</td>
<td>0.335202</td>
<td>$1.0223 \times 10^{-12}$</td>
<td>1.5965</td>
<td>0.027271047</td>
</tr>
<tr>
<td>0.259009</td>
<td>0.306475</td>
<td>$1.04955 \times 10^{-12}$</td>
<td>1.8303</td>
<td>0.057713863</td>
</tr>
<tr>
<td>0.283232</td>
<td>0.291644</td>
<td>$2.31581 \times 10^{-12}$</td>
<td>2.0472</td>
<td>0.043059261</td>
</tr>
<tr>
<td>0.329948</td>
<td>0.269679</td>
<td>$6.70566 \times 10^{-12}$</td>
<td>2.2893</td>
<td>0.049538874</td>
</tr>
<tr>
<td>0.302278</td>
<td>0.249769</td>
<td>$8.93862 \times 10^{-12}$</td>
<td>2.4909</td>
<td>0.122064819</td>
</tr>
<tr>
<td>0.323113</td>
<td>0.249188</td>
<td>$2.01045 \times 10^{-11}$</td>
<td>2.7399</td>
<td>0.194264304</td>
</tr>
</tbody>
</table>

continued on next page
<table>
<thead>
<tr>
<th>Reprojection Error (pixels)</th>
<th>Blur</th>
<th>Error Ellipse Volume</th>
<th>Perpendicular Distance (m)</th>
<th>Ground Truth Error (m)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.32487</td>
<td>0.264439</td>
<td>2.69782×10⁻¹¹</td>
<td>2.9677</td>
<td>0.233775405</td>
</tr>
<tr>
<td>0.171654</td>
<td>0.251639</td>
<td>7.50742×10⁻¹²</td>
<td>3.1822</td>
<td>0.337270944</td>
</tr>
<tr>
<td>0.212416</td>
<td>0.230506</td>
<td>1.65697×10⁻¹¹</td>
<td>3.4069</td>
<td>0.082846726</td>
</tr>
<tr>
<td>0.311882</td>
<td>0.312959</td>
<td>1.40736×10⁻¹¹</td>
<td>1.1208</td>
<td>0.006969935</td>
</tr>
<tr>
<td>0.284189</td>
<td>0.322527</td>
<td>3.03018×10⁻¹¹</td>
<td>1.3479</td>
<td>0.006941902</td>
</tr>
<tr>
<td>0.302746</td>
<td>0.337914</td>
<td>9.01716×10⁻¹³</td>
<td>1.585</td>
<td>0.021995681</td>
</tr>
<tr>
<td>0.325286</td>
<td>0.343823</td>
<td>2.28463×10⁻¹²</td>
<td>1.8286</td>
<td>0.035316852</td>
</tr>
<tr>
<td>0.254273</td>
<td>0.348979</td>
<td>2.1908×10⁻¹²</td>
<td>2.0429</td>
<td>0.060194435</td>
</tr>
<tr>
<td>0.348365</td>
<td>0.338313</td>
<td>9.11666×10⁻¹²</td>
<td>2.2944</td>
<td>0.046139137</td>
</tr>
<tr>
<td>0.295449</td>
<td>0.29857</td>
<td>1.04177×10⁻¹¹</td>
<td>2.5236</td>
<td>0.088027269</td>
</tr>
<tr>
<td>0.294031</td>
<td>0.289779</td>
<td>1.57167×10⁻¹¹</td>
<td>2.73</td>
<td>0.049334065</td>
</tr>
<tr>
<td>0.19543</td>
<td>0.337261</td>
<td>8.1721×10⁻¹²</td>
<td>3.0407</td>
<td>0.108720099</td>
</tr>
<tr>
<td>0.219394</td>
<td>0.298317</td>
<td>1.65133×10⁻¹¹</td>
<td>3.2173</td>
<td>0.204737026</td>
</tr>
<tr>
<td>0.292719</td>
<td>0.260186</td>
<td>5.60393×10⁻¹¹</td>
<td>3.4469</td>
<td>0.239343393</td>
</tr>
<tr>
<td>0.268857</td>
<td>0.317864</td>
<td>1.00731×10⁻¹³</td>
<td>1.1193</td>
<td>0.007971198</td>
</tr>
<tr>
<td>0.318447</td>
<td>0.323292</td>
<td>4.46251×10⁻¹³</td>
<td>1.3503</td>
<td>0.015279725</td>
</tr>
<tr>
<td>0.300811</td>
<td>0.347937</td>
<td>8.68693×10⁻¹³</td>
<td>1.5867</td>
<td>0.035079481</td>
</tr>
<tr>
<td>0.278228</td>
<td>0.342601</td>
<td>1.53966×10⁻¹¹</td>
<td>1.8253</td>
<td>0.026024027</td>
</tr>
<tr>
<td>0.285014</td>
<td>0.340733</td>
<td>3.12612×10⁻¹²</td>
<td>2.0588</td>
<td>0.031618349</td>
</tr>
<tr>
<td>0.300666</td>
<td>0.315367</td>
<td>6.43864×10⁻¹²</td>
<td>2.2979</td>
<td>0.046619524</td>
</tr>
<tr>
<td>0.331851</td>
<td>0.287771</td>
<td>1.46994×10⁻¹¹</td>
<td>2.5296</td>
<td>0.04709501</td>
</tr>
<tr>
<td>0.361852</td>
<td>0.271315</td>
<td>2.89333×10⁻¹¹</td>
<td>2.7473</td>
<td>0.124941346</td>
</tr>
<tr>
<td>0.326163</td>
<td>0.279405</td>
<td>3.38809×10⁻¹¹</td>
<td>2.9701</td>
<td>0.097628377</td>
</tr>
<tr>
<td>0.208325</td>
<td>0.271163</td>
<td>1.37819×10⁻¹¹</td>
<td>3.2523</td>
<td>0.126860908</td>
</tr>
<tr>
<td>0.373258</td>
<td>0.23397</td>
<td>1.09025×10⁻¹⁰</td>
<td>3.4666</td>
<td>0.210356982</td>
</tr>
</tbody>
</table>
D.2  Ground Truth Error Vs. Aggregate Errors of Perpendicular Distance Functions

Table D.2 shows the raw data for the plotted points in Figures 5.13, 5.14, 5.15, and 5.16. As in Appendix D.1, the first four columns show the data for the graphs’ respective horizontal axes, and the last column shows the ground truth error, which is the data for the vertical axes of the graphs.

Table D.2: Raw data for the Ground Truth Error Vs. Aggregate Error figures.

<table>
<thead>
<tr>
<th>Linear Function $f(d) = d$</th>
<th>Quadratic Function $f(d) = d^2$</th>
<th>Exponential Function $f(d) = 3^d - 1$</th>
<th>Exponential Function $f(d) = 50^d - 1$</th>
<th>Ground Truth Error (m)</th>
</tr>
</thead>
<tbody>
<tr>
<td>5.6849801</td>
<td>6.46381978</td>
<td>12.43636785</td>
<td>422.2730179</td>
<td>0.02907989</td>
</tr>
<tr>
<td>7.276336945</td>
<td>9.00375022</td>
<td>17.20162228</td>
<td>940.3695198</td>
<td>0.056874335</td>
</tr>
<tr>
<td>6.80960303</td>
<td>7.72859652</td>
<td>14.87655053</td>
<td>502.6873058</td>
<td>0.02907989</td>
</tr>
<tr>
<td>6.140643075</td>
<td>7.71394966</td>
<td>14.71934233</td>
<td>856.3517294</td>
<td>0.056874335</td>
</tr>
<tr>
<td>7.937007075</td>
<td>8.99968102</td>
<td>17.32735352</td>
<td>584.0206241</td>
<td>0.046049213</td>
</tr>
<tr>
<td>7.26804712</td>
<td>8.98503416</td>
<td>17.17014532</td>
<td>937.6850477</td>
<td>0.073738525</td>
</tr>
<tr>
<td>6.81257823</td>
<td>7.73530399</td>
<td>14.88782744</td>
<td>503.6447083</td>
<td>0.044411035</td>
</tr>
<tr>
<td>7.93720116</td>
<td>9.00008073</td>
<td>17.32801012</td>
<td>584.0589962</td>
<td>0.044411035</td>
</tr>
<tr>
<td>7.268241205</td>
<td>8.98543387</td>
<td>17.17080192</td>
<td>937.7234198</td>
<td>0.078420533</td>
</tr>
<tr>
<td>7.933971875</td>
<td>8.99277466</td>
<td>17.31571619</td>
<td>583.0032874</td>
<td>0.061494227</td>
</tr>
<tr>
<td>6.800092055</td>
<td>7.70709121</td>
<td>14.84036927</td>
<td>499.586965</td>
<td>0.061494227</td>
</tr>
<tr>
<td>6.1311321</td>
<td>7.69244435</td>
<td>14.68316107</td>
<td>853.2511201</td>
<td>0.094263089</td>
</tr>
<tr>
<td>9.29345545</td>
<td>10.84169583</td>
<td>20.77047704</td>
<td>787.0299209</td>
<td>0.077509032</td>
</tr>
<tr>
<td>8.159465725</td>
<td>9.556012375</td>
<td>18.29513012</td>
<td>703.6133301</td>
<td>0.077509032</td>
</tr>
<tr>
<td>7.49050577</td>
<td>9.541365515</td>
<td>18.13792192</td>
<td>1057.277754</td>
<td>0.109422895</td>
</tr>
<tr>
<td>8.1719519</td>
<td>9.584225155</td>
<td>18.34258828</td>
<td>707.6713418</td>
<td>0.060363648</td>
</tr>
<tr>
<td>9.29657483</td>
<td>10.8490019</td>
<td>20.78277096</td>
<td>788.0856297</td>
<td>0.060363648</td>
</tr>
</tbody>
</table>

continued on next page
## APPENDIX D. DATA TABLES

### APPENDIX D. DATA TABLES

<table>
<thead>
<tr>
<th>Linear Function</th>
<th>Quadratic Function</th>
<th>Exponential Function</th>
<th>Exponential Function</th>
<th>Ground Truth Error (m)</th>
</tr>
</thead>
<tbody>
<tr>
<td>$f(d) = d$</td>
<td>$f(d) = d^2$</td>
<td>$f(d) = 3^d - 1$</td>
<td>$f(d) = 50^d - 1$</td>
<td></td>
</tr>
<tr>
<td>8.627614875</td>
<td>10.83435504</td>
<td>20.62556276</td>
<td>1141.750053</td>
<td>0.093118741</td>
</tr>
<tr>
<td>6.811205045</td>
<td>7.73217614</td>
<td>14.88255553</td>
<td>503.1822507</td>
<td>0.0327324</td>
</tr>
<tr>
<td>8.40256189</td>
<td>10.27210658</td>
<td>19.64780996</td>
<td>1021.278753</td>
<td>0.065619052</td>
</tr>
<tr>
<td>7.935827975</td>
<td>8.99695288</td>
<td>17.32273821</td>
<td>583.5965386</td>
<td>0.0327324</td>
</tr>
<tr>
<td>7.26686802</td>
<td>8.98230602</td>
<td>17.16553001</td>
<td>937.2609621</td>
<td>0.065619052</td>
</tr>
<tr>
<td>9.06323202</td>
<td>10.26803738</td>
<td>19.77354119</td>
<td>664.9298569</td>
<td>0.049499899</td>
</tr>
<tr>
<td>8.394272065</td>
<td>10.25339052</td>
<td>19.61633299</td>
<td>1018.59428</td>
<td>0.081265798</td>
</tr>
<tr>
<td>7.938803175</td>
<td>9.00366035</td>
<td>17.33401511</td>
<td>584.5539411</td>
<td>0.05348439</td>
</tr>
<tr>
<td>9.063426105</td>
<td>10.26843709</td>
<td>19.77419779</td>
<td>664.968229</td>
<td>0.05348439</td>
</tr>
<tr>
<td>8.39446615</td>
<td>10.25379023</td>
<td>19.61698959</td>
<td>1018.632653</td>
<td>0.088740633</td>
</tr>
<tr>
<td>9.06019682</td>
<td>10.26113102</td>
<td>19.76190387</td>
<td>663.9125202</td>
<td>0.06911447</td>
</tr>
<tr>
<td>7.926317</td>
<td>8.97544757</td>
<td>17.28655695</td>
<td>580.4959293</td>
<td>0.06911447</td>
</tr>
<tr>
<td>7.257357045</td>
<td>8.96080071</td>
<td>17.12934875</td>
<td>934.1603529</td>
<td>0.103438097</td>
</tr>
<tr>
<td>8.15662967</td>
<td>9.549572035</td>
<td>18.28428324</td>
<td>702.6709413</td>
<td>0.084257819</td>
</tr>
<tr>
<td>7.02274985</td>
<td>8.263888585</td>
<td>15.80893632</td>
<td>619.2543504</td>
<td>0.084257819</td>
</tr>
<tr>
<td>6.353789895</td>
<td>8.249241725</td>
<td>15.65172812</td>
<td>972.918774</td>
<td>0.117804244</td>
</tr>
<tr>
<td>7.035236025</td>
<td>8.292101365</td>
<td>15.85639449</td>
<td>623.3123622</td>
<td>0.068015292</td>
</tr>
<tr>
<td>8.159858955</td>
<td>9.556878105</td>
<td>18.29657717</td>
<td>703.7266501</td>
<td>0.068015292</td>
</tr>
<tr>
<td>7.490899</td>
<td>9.542231245</td>
<td>18.13936897</td>
<td>1057.391074</td>
<td>0.102298534</td>
</tr>
</tbody>
</table>
Appendix E

Letters of Permission to Use Copyrighted Material
E.1 Permission to Use Conference Paper

Use of my paper's material in a Master's thesis

From: Chun-Yi Su <chunyi.su@gmail.com>
To: shaya@uwindsor.ca
Cc: general@icira2012.org

Hi Karam,

It’s allowed for this purpose.

Regards,

Chun-Yi Su

From: Karam Shaya <shaya@uwindsor.ca>
Date: Fri, Sep 7, 2012 at 12:34 PM
Subject: Use of my paper's material in a Master's thesis
To: general@icira2012.org

To Whom It May Concern:

I am completing a Master’s thesis at the University of Windsor tentatively entitled “An Error-Reducing Self-Localization Algorithm with Online Map-Building Capabilities.” I would like your permission to include in my thesis the material of my own paper (K. Shaya, A. Marinac, J. L. A. Herrera, and X. Chen, “A Self-Localization System with Global Error Reduction and Online Map-Building Capabilities”) that was accepted for publication in the ICRA 2012 Conference Proceedings. This will include the use of text and figures from the paper.

My thesis will either be deposited to the University of Windsor Leddy library or to the University of Windsor’s online theses and dissertations repository (http://winspace.uwindsor.ca), in which case it will be available in full-text on the internet for reference, study and / or copy.

I will also be granting Library and Archives Canada and ProQuest/UMI a non-exclusive license to reproduce, loan, distribute, or sell single copies of my thesis by any means and in any form or format. These rights will in no way restrict reproduction of the material in any other form by you or by others authorized by you.

Please confirm by email that these arrangements meet with your approval. Thank you very much for your attention to this matter.

Sincerely,

Karam Shaya
APPENDIX E. LETTERS OF PERMISSION TO USE COPYRIGHTED MATERIAL 94

E.2 Permission to Use Dr. Boufama’s Lecture Notes

Requesting permission to use your class notes in thesis

Karam Shaya
Karam Shaya <kshaya@uwindsor.ca>

Tue, Sep 11, 2012 at 11:17 AM

Yes, you have my permission

Thanks

Bouhafems

Dr. Bouhafems Boufama, Phone: (519) 253-3000 ext. 3796
Professor E-mail: boufama@uwindsor.ca
School of Computer Science University of Windsor
Windsor, ON, Canada N9B 3P4

Dear Dr. Boufama,

I am completing a Master’s thesis at the University of Windsor tentatively entitled “An Error-Reducing Self-Localization Algorithm with Online Map-Building Capabilities.” I would like your permission to include in my thesis the material of your lecture notes from your Visual Processing (03-60-551, Winter 2013) class. This will include worded versions of your text along with figures.

My thesis will either be deposited to the University of Windsor Leddy Library or to the University of Windsor’s online theses and dissertations repository (http://winpace.uwindsor.ca), in which case it will be available in full-text on the internet for reference, study and/or copy.

I will also be granting Library and Archives Canada and ProQuest/UMI a non-exclusive license to reproduce, loan, distribute, or sell single copies of my thesis by any means and in any form or format. These rights will in no way restrict republication of the material in any other form by you or by others authorized by you.

Please confirm by email that these arrangements meet with your approval. Thank you very much for your attention to this matter.

Sincerely,

Karam Shaya
References


REFERENCES


Vita Auctoris

Karam Shaya was born in Mosul, Iraq in 1988. In 2006, he graduated from Holy Names High School in Windsor, Ontario. He attended the University of Windsor in Windsor, Ontario, where he obtained his B.A.Sc. in Electrical and Computer Engineering in 2010. He is currently a candidate for the Master of Applied Science degree in Electrical and Computer Engineering at the University of Windsor.