Methods for Calibration, Registration,
and Change Detection in Robot Mapping Applications

by

Aravindhan K Krishnan

A Dissertation Presented in Partial Fulfillment
of the Requirements for the Degree
Doctor of Philosophy

Approved October 2016 by the
Graduate Supervisory Committee:

Srikanth Saripalli, Chair
Andrew Klesh
Jekan Thangavelautham
Pavan Turaga
Georgios Fainekos

ARIZONA STATE UNIVERSITY
December 2016
ABSTRACT

Multi-sensor fusion is a fundamental problem in Robot Perception. For a robot to operate in a real world environment, multiple sensors are often needed. Thus, fusing data from various sensors accurately is vital for robot perception. In the first part of this thesis, the problem of fusing information from a LIDAR, a color camera and a thermal camera to build RGB-Depth-Thermal (RGBDT) maps is investigated. An algorithm that solves a non-linear optimization problem to compute the relative pose between the cameras and the LIDAR is presented. The relative pose estimate is then used to find the color and thermal texture of each LIDAR point. Next, the various sources of error that can cause the mis-coloring of a LIDAR point after the cross-calibration are identified. Theoretical analyses of these errors reveal that the coloring errors due to noisy LIDAR points, errors in the estimation of the camera matrix, and errors in the estimation of translation between the sensors disappear with distance. But errors in the estimation of the rotation between the sensors causes the coloring error to increase with distance.

On a robot (vehicle) with multiple sensors, sensor fusion algorithms allow us to represent the data in the vehicle frame. But data acquired temporally in the vehicle frame needs to be registered in a global frame to obtain a map of the environment. Mapping techniques involving the Iterative Closest Point (ICP) algorithm and the Normal Distributions Transform (NDT) assume that a good initial estimate of the transformation between the 3D scans is available. This restricts the ability to stitch maps that were acquired at different times. Mapping can become flexible if maps that were acquired temporally can be merged later. To this end, the second part of this thesis focuses on developing an automated algorithm that fuses two maps by finding a congruent set of five points forming a pyramid.

Mapping has various application domains beyond Robot Navigation. The third
part of this thesis considers a unique application domain where the surface displace-
ments caused by an earthquake are to be recovered using pre- and post-earthquake
LIDAR data. A technique to recover the 3D surface displacements is developed and
the results are presented on real earthquake datasets: El Mayur Cucupa earthquake,
Mexico, 2010 and Fukushima earthquake, Japan, 2011.
ACKNOWLEDGMENTS

I thank my advisor Prof. Srikanth Saripalli for guiding me through my thesis for the past 5 years and providing me ample opportunities to learn and grow as a researcher.

I would like to thank all my labmates: Yucong Lin, Sai Vemprala, Shatadal Mishra, Andres Mora, Patrick McGarey, Ben Stinnett, Alexander Kafka, Phil Burbank, Anand Biradar, Prasanna Ghadage, Aniket Shirsat, and Stephanie Booth for spending their valuable time with me in various discussions and data collection runs. Special thanks to the folks from Spain: Jesús Pestana, Changhong Fu, Adrian Carrio, and Carlos Sampedro Pérez for the time spent on heated debates and discussions, as well as the numerous fun moments.

It was a pleasure to work with Gelogists Prof. Ramon Arrowsmith and Prof. Edwin Nissen (currently at Colorado School of Mines) at SESE, ASU. Projects with Prof. Jim Bell were a steep learning curve as well. These experiences diversified my perspective on Robotics Applications.

Beyond the lab, I am thankful to the friends I made at The Faithful City. The long walks and rants with Ranjit Christopher definitely made my grad life easier. Conversations with Sunil Deshpande were enjoyable as well.

I would like to thank the teachers of every single course I took at Arizona State University. Each course was different in it’s own way and it opened up new avenues for intellectual growth which I hope to nurture throughout my career.
I am grateful to my alma mater ‘International Institute of Information Technology (IIIT) Hyderabad’ for cultivating an interest in research and the Robotics Research Center for introducing me to the awesome fields of Robotics and Computer Vision.

Finally, none of these would have happened without the staunch support of my wife, Shilpa and the folks back home in India. A big ‘Thank You’ to all of them.
# TABLE OF CONTENTS

<table>
<thead>
<tr>
<th>LIST OF TABLES</th>
<th>viii</th>
</tr>
</thead>
<tbody>
<tr>
<td>LIST OF FIGURES</td>
<td>ix</td>
</tr>
</tbody>
</table>

### CHAPTER

1. INTRODUCTION
   1.1 Overview of Sensors Used
      1.1.1 LIDAR Scanners
      1.1.2 Monocular/Color Cameras
      1.1.3 Multi-spectral / Thermal Cameras
      1.1.4 IMU/GPS
   1.2 Multi-modal Sensing
   1.3 Registration
   1.4 Change Detection
   1.5 Contributions
   1.6 Publications

2. CROSS CALIBRATION OF RGB AND THERMAL CAMERAS WITH A LIDAR
   2.1 Related Work
   2.2 Method
      2.2.1 Calibration Target for the Thermal Camera
      2.2.2 Cross Calibration Setup
      2.2.3 Detecting Edges in the Image
      2.2.4 Detecting Edges in the LIDAR Point Cloud
      2.2.5 Generating All Points on the Circle
<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.2.6 Computing the Extrinsic Parameters Between the Cameras and the LIDAR</td>
<td>31</td>
</tr>
<tr>
<td>2.3 Results &amp; Discussion</td>
<td>34</td>
</tr>
<tr>
<td>2.3.1 Convexity of the Objective Function</td>
<td>34</td>
</tr>
<tr>
<td>2.3.2 Convergence Analysis</td>
<td>36</td>
</tr>
<tr>
<td>2.4 RGB-Depth-Thermal Mapping</td>
<td>37</td>
</tr>
<tr>
<td>2.5 Conclusions</td>
<td>38</td>
</tr>
<tr>
<td>3 COLORING ERROR AFTER CROSS-CALIBRATION: AN ERROR ANALYSIS</td>
<td>40</td>
</tr>
<tr>
<td>3.1 Noisy 3D points (Arising from the LIDAR)</td>
<td>40</td>
</tr>
<tr>
<td>3.2 Error in the Camera Calibration Parameters (K Matrix)</td>
<td>43</td>
</tr>
<tr>
<td>3.3 Error in the Extrinsic Parameters</td>
<td>45</td>
</tr>
<tr>
<td>3.3.1 Error in Rotation Components</td>
<td>45</td>
</tr>
<tr>
<td>3.3.2 Error in Translation Components</td>
<td>46</td>
</tr>
<tr>
<td>3.4 Coloring Error Analysis Setup</td>
<td>48</td>
</tr>
<tr>
<td>3.5 Comparison</td>
<td>48</td>
</tr>
<tr>
<td>3.5.1 Error Minimization</td>
<td>50</td>
</tr>
<tr>
<td>3.5.2 Information Maximization</td>
<td>50</td>
</tr>
<tr>
<td>3.6 Conclusions</td>
<td>52</td>
</tr>
<tr>
<td>4 INITIAL ALIGNMENT METHOD FOR POINT CLOUD REGISTRATION</td>
<td>54</td>
</tr>
<tr>
<td>4.1 Related Work</td>
<td>54</td>
</tr>
<tr>
<td>4.2 Approach</td>
<td>59</td>
</tr>
<tr>
<td>4.2.1 Workflow for Computing Initial Alignment</td>
<td>60</td>
</tr>
</tbody>
</table>
## Chapter 4.2: Finding a Congruent Base
- Finding a Congruent Base: Page 60
- Selecting the MAX Plane: Page 64
- Finding a Congruent Pyramid: Page 66
- Flatness Measure: Page 67
- Change in Flatness Measure Due to Sensor Noise: Page 68

## Chapter 4.3: Sufficiency Proofs
- Sufficiency Proofs: Page 70

## Chapter 4.4: Results
- Results: Page 71

## Chapter 4.5: Conclusions
- Conclusions: Page 77

## Chapter 5: Change Detection Using Airborne Lidar
- Problem Statement: Page 78
- Data Description: Page 79
- Approach: Page 80
- Finding the Right Window Size: Page 81
- Transformation Validation: Page 82
- Fault Analysis: Page 83
- Results: Page 84

## Chapter 6: Conclusion
- Summary of Contributions and Future Work: Page 91

## References
- References: Page 96
<table>
<thead>
<tr>
<th>Table</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.1 Convergence Analysis: This Table Reports the Errors Observed When the Optimizer was Initialized with Different Initial Estimates. We Observe that our Method Converges when the Initial Estimates are off by 15 degrees for rotation Parameters and by 1.25m for the Translation Parameters. The Maximum Time Taken for Convergence in these Experiments is 36 seconds</td>
<td>35</td>
</tr>
<tr>
<td>3.1 Coloring Errors at Various Depths for the RGB Camera</td>
<td>49</td>
</tr>
<tr>
<td>3.2 Coloring Errors at Various Depths for the Thermal Camera</td>
<td>49</td>
</tr>
<tr>
<td>4.1 4 sq km Dataset</td>
<td>72</td>
</tr>
<tr>
<td>4.2 Flat Topography Dataset</td>
<td>72</td>
</tr>
<tr>
<td>4.3 4 sq km Dataset</td>
<td>74</td>
</tr>
<tr>
<td>4.4 RMS Error for Varying Gaussian Noise</td>
<td>74</td>
</tr>
</tbody>
</table>
# LIST OF FIGURES

<table>
<thead>
<tr>
<th>Figure</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.1 Various LIDAR scanners</td>
<td>4</td>
</tr>
<tr>
<td>1.2 Monocular and Stereo cameras</td>
<td>5</td>
</tr>
<tr>
<td>1.3 Multi-spectral and Thermal cameras</td>
<td>6</td>
</tr>
<tr>
<td>1.4 Exposure vs Mean pixel value plot averaged over multiple images. It is noticeable that the plot is linear until an exposure time of 5 ms before entering a region of non-linearity</td>
<td>7</td>
</tr>
<tr>
<td>1.5 IMU and GPS</td>
<td>8</td>
</tr>
<tr>
<td>2.1 The Setup Consisting of a Hokuyo LIDAR, PointGrey Color Camera, and a FLIR Tau2 Thermal Camera</td>
<td>22</td>
</tr>
<tr>
<td>2.2 Calibration Target used for Thermal and RGB Camera Calibration</td>
<td>22</td>
</tr>
<tr>
<td>2.3 Calibration Target used for Thermal and RGB Camera Calibration</td>
<td>23</td>
</tr>
<tr>
<td>2.4 Detecting the Edges of the Circular Target in the Image</td>
<td>23</td>
</tr>
<tr>
<td>2.5 Detecting the edges of the circular target in the LIDAR point cloud</td>
<td>30</td>
</tr>
<tr>
<td>2.6 Distance Transform</td>
<td>32</td>
</tr>
<tr>
<td>2.7 Textured Point Clouds</td>
<td>33</td>
</tr>
<tr>
<td>2.8 (a) and (b) : The Variation of Parameters Roll, Pitch, Yaw, Tx, Ty, and Tz Around the Global Minimum. Roll, pitch, and Yaw were Varied by -30 Degrees to +30 Degrees Around the Global Minimum. Tx, Ty, and Tz were Varied by -3m to +3m Around the Global Minimum. The Y Axis Shows the Objective Function Values. One Can notice that the Function Appears Smooth and There are no Steep Local Minima</td>
<td>36</td>
</tr>
<tr>
<td>2.9 Mapping Pipeline</td>
<td>37</td>
</tr>
<tr>
<td>2.10 RGB-Depth-Thermal Mapping</td>
<td>39</td>
</tr>
<tr>
<td>3.1 Reprojection Error vs Depth of the 3D Point</td>
<td>42</td>
</tr>
<tr>
<td>Figure</td>
<td>Page</td>
</tr>
<tr>
<td>--------</td>
<td>------</td>
</tr>
<tr>
<td>3.2 Coloring Error Due to Error in Rotation</td>
<td>46</td>
</tr>
<tr>
<td>3.3 Coloring Error Analysis Setup. Errors can be Seen Along the Edges of the White Cardboard as the Depth Increases</td>
<td>47</td>
</tr>
<tr>
<td>3.4 Measuring the Colouring Error using Cloud Compare. Here the Colouring is Off by 10 cm as Seen in the Call-out</td>
<td>49</td>
</tr>
<tr>
<td>3.5 Intensity Distributions of the MI dataset - the LIDAR Intensities are Calibrated</td>
<td>52</td>
</tr>
<tr>
<td>3.6 Intensity Distributions of our Dataset - the LIDAR Intensities are NOT Calibrated</td>
<td>53</td>
</tr>
<tr>
<td>3.7 Intensity Distribution</td>
<td>53</td>
</tr>
<tr>
<td>4.1 Congruency Test for 4 Points, Based on the Affine Invariant Property that the Ratio of Lengths Does Not Change</td>
<td>55</td>
</tr>
<tr>
<td>4.2 Different Aerial Datasets</td>
<td>63</td>
</tr>
<tr>
<td>4.3 Choosing Planes for Finding Congruent Quadrilaterals, Shown in 2D. The Topography is Drawn in Blue, Red Lines are Cross Section of Planes from Which the Quadrilaterals are Chosen</td>
<td>64</td>
</tr>
<tr>
<td>4.4 Multiple Congruent Quadrilaterals on a Planar Surface</td>
<td>65</td>
</tr>
<tr>
<td>4.5 Finding Congruent Pyramids</td>
<td>68</td>
</tr>
</tbody>
</table>
4.6 The Top Row Shows the Various Residue Distributions Considered (Only 4 is shown for Brevity. 1000 Distributions were Considered in Total). The Bottom Row Shows the Different Noise Distributions Considered (4 in Total). The Noise is Modelled as a Gaussian Mixture. Each Residue Distribution is Convolved with Every Noise Distribution and the Difference in Flatness Measure is Computed. The Difference in Flatness Measure $\Delta H_i$ is Shown in the Last Row. One Can Notice that $\Delta H_i$ is not Significant to Mis-interpret a Flat Surface Otherwise.

4.7 Results on the 4 sq km Dataset

4.8 Results on a Flat Topography Dataset Spanning 0.24 sq kms

4.9 Urban Outdoor Dataset Spanning 120 x 330 m

4.10 Indoor Data of a Garage

4.11

5.1 Standard Deviation of Computed Displacements for Different Window Sizes

5.2 Standard Deviations by Window Number, for Different Window Sizes. (a) Shows the Plot for Window Sizes of 75 m (red), 50 m (green) and 25 m (blue). In (b), We also Plot Standard Deviations for Window Size of 10 m (purple), with an Enlarged $y$-axis Scale Such That the Standard Deviations for the 75 m, 50 m and 25 m Window Sizes are Barely Visible. There is a Huge Increase in Standard Deviation When the Window Size is Reduced from 25 m to 10 m, Suggesting that Window Splitting should be Stopped at 25 m.
<table>
<thead>
<tr>
<th>Figure</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>5.3</td>
<td>(a) Height Difference Map of the Mexico Earthquake, Before Global ICP, with $x$ and $y$ Coordinates in Meters. Height Changes Across the Fault are Clear. (b) Height difference Map After Global ICP, with Height Differences Reduced.</td>
</tr>
<tr>
<td>5.4</td>
<td>Window Split and Fault Detection</td>
</tr>
<tr>
<td>5.5</td>
<td>Displacement Vectors for Different Window Sizes. The Approximate Length of the Displacement Vectors is 2 m. Notice the Change in Vector Directions on Both Sides of the Fault. $X$ and $Y$ Values are in Meters.</td>
</tr>
<tr>
<td>5.6</td>
<td>Results for the Simulated Earthquake. Horizontal Displacements (Black Arrows) and Vertical Displacements (Coloured Circles) Can Clearly be Seen to Change Markedly Either Side of the Fault. $X$ and $Y$ Axes Show UTM Zone 11 Coordinates, in Meters.</td>
</tr>
<tr>
<td>5.7</td>
<td>Results for the Mexico Earthquake. The Thin Lines Show the Earthquake Surface Faulting, as Observed by Geologists, with E-facing Scarps in Green and W-facing Scarps in Blue. Again, the Horizontal and Vertical Displacements Clearly Change Markedly Across the Fault. $X$ and $Y$ Axes Show UTM Zone 11 Coordinates, in Meters.</td>
</tr>
<tr>
<td>5.8</td>
<td>Results for the Japan Earthquake. $X$ and $Y$ Axes Show UTM Zone 11 Coordinates, in Meters.</td>
</tr>
</tbody>
</table>
Chapter 1

INTRODUCTION

Perception is key to any autonomous robotic system. Perception allows robots to understand their environment and plan their behaviour to accomplish tasks. Perception technologies have enabled robots to move out of factories and operate in human environments. A notable example of perception technologies enabling real world robots are the self-driving cars. [40]. This progress has been due to various factors: i) Availability and affordability of new sensors ii) Substantial progress on some fundamental problems in robot perception iii) Tremendous increase in computing power.

Perception is achieved by mounting sensors that gather some data about the environment. Till date, a wide range of sensing modalities have been used in robotic systems. A survey of the various sensing technologies used in Robotics is presented in [29]. Back in the 1980s, vision was the primary sensing mechanism and it was used for many robot navigation applications [59]. The 1990s saw the increased use of LIDAR scanners primarily due to the fact that they became compact and more affordable. Moreover, the range data was accurate to a few centi-meters. The very first mobile robotic systems to be deployed in human environments relied heavily on LIDAR scanners [75]. Single sensor robotic systems were primarily used to solve the fundamental problems in Robot Perception such as Localization, Mapping, Place Recognition, and Simultaneous Localization and Mapping (SLAM). When solutions to these fundamental problems were known in the 2000s, there was a surge of multi-sensor robotic systems that gathered and operated on richer datasets to make these
solutions robust. This opened up new problems and possibilities for Robot Perception applications. It is now safe to say that every real world robot operating today has a multitude of sensors for perception, and the sensors used varies according to the application. As different sensors are being used, it is important to understand the individual characteristics of each sensor and their associated errors. It is also vital to understand the calibration algorithms that fuse the data from individual sensors and the inherent errors this ‘fusion’ brings.

1.1 Overview of Sensors Used

To get high fidelity data from multi-modal sensing systems it is important to understand the various errors and model/correct them. Here, we present a brief overview of some commonly used sensors in Robot Perception and the calibration required for each one of them. This is not a comprehensive list, but a brief overview of the current sensors being used.

1.1.1 LIDAR Scanners

LIDAR scanners operate based on the time of flight principle. They measure the distance to an object based on the time taken by emitted LIDAR beam to reach the receiver after being reflected by the object. LIDAR scanners became prevalent because the measurement data was available real time and the depth data was accurate to the order of a few centimeters. Some of the widely used LIDAR scanners are shown in Figure 1.1. The initial scanners used on mobile robots were the SICK LIDARs and these were wide used by many teams in the DARPA grand challenge [76]. The SICK LIDAR is a 2D scanner with an angular resolution of 0.25 degrees, and the maximum range being \(\sim 80\) m. The measurement resolution is 10 mm and the accuracy
is 35 mm. SICK LIDAR scanners can be used in outdoor environments (both during day and night). However, they are not reliable in rain, fog, and dust due to the scattering of the light beams. Hokuyo LIDAR scanners are lightweight 2D scanners and are typically used in indoor environments (range: 30 m, angular resolution: 0.25 degrees, accuracy: 50 mm). These 2D LIDAR scanners do not require any prior calibration procedure for mapping applications. The measurement error is usually incorporated into the sensor model while building a metric map. Reliable outdoor navigation requires a dense 3D model of the environment, so the use of 2D LIDAR scanners for outdoor navigation has been limited after 3D LIDAR scanners became prevalent. 3D mapping of outdoor/urban environments became popular with the advent of 3D LIDAR scanners like the Velodyne 64SE - which is a 64 beam LIDAR, that rotates between 5-20 times per second producing dense 360 degree scans of the environment in real time. The dense data has enabled mapping as well as tracking applications. However, using multiple beams introduces new challenges. For the 3D data to be accurate, the extrinsic parameters between the individual beams should be known accurately. So this requires a calibration procedure before LIDAR scanner can be operated.

Recently, there has been an interest in using LIDAR intensity images for localization in urban environments [83]. This requires that the intensity data of the LIDAR returns are calibrated. The intensity returns of the LIDAR may not represent the true albedo of the surface and a calibration procedure is necessary to correct the inaccuracies in the data before they can be used in any further processing.
1.1.2 Monocular/Color Cameras

Cameras are passive sensing systems that measure the ambient light in the scene. They operate very similar to the human eye, hence ‘vision only’ robotic systems have been the dream of Computer Vision purists. Consequently, Vision based navigation was the primary focus of robot perception research until LIDAR scanners became affordable in the 1990s. Initial research included motion estimation, photogrammetry and stereo, multiple view geometry and pattern classification [30]. Later, the fields of Computer Vision and Robotics diverged and Robot Vision research involved real time mapping using single monocular cameras [19]. SLAM systems using single cameras came into existence [22], [36], [72]. Concurrently, their have been attempts on long term mapping/localization without generating any 3D information from images [51], [50], [52]. All SLAM algorithms require that the cameras are geometrically calibrated. The most common method of calibration is to use checkerboard targets and compute the intrinsic parameters by finding the relationship between the checkerboard corners in the camera frame and the world frame. This is the only calibration step required in color cameras. In the case of stereo systems, the calibration pro-
procedure also involves finding the extrinsic parameters between the left and the right cameras in the stereo pair. This allows to construct 3D maps at a known scale. Stereo cameras have been of interest primarily because of their ability to construct maps to a known scale facilitating higher level navigation algorithms. Stereo based navigation systems have been deployed in the real world, a notable example being the Spirit and Opportunity rovers in the Mars missions [17].

1.1.3 Multi-spectral / Thermal Cameras

Multi-spectral cameras are used in a wide range of disciplines such as remote sensing, industrial quality control, medical imaging etc. In the context of Robotics, they are very useful for planetary exploration. This is because, multi-spectral cameras can be used to identify rock/mineral types based on the spectral response at different wavelengths, which can lead to goal oriented exploration for planetary rovers. For instance, the PanCam [9] on the Mars Exploration rovers is a mulit-spectral camera.

Identifying mineral types based on the spectral composition require an accurate spectral composition of the scene. For this purpose, the pixel to pixel sensitivities of the detector needs to be corrected – this calibration procedure is referred to as
‘flat field correction’. Sometimes, the detector response can become non-linear as the exposure times increase. This happens especially when the pixels reach their full well capacity \[37\]. At this point electrons overflow to adjacent pixels corrupt the signal measured at adjacent pixels, resulting in a non-linear response characteristic. This artifact in the data must be corrected when an accurate spectral composition of the scene is needed. This is done by identifying the maximum cut-off pixel value before the non-linearity (or pixels reaching their full well capacity) starts. The calibration procedure involves plotting the detector response vs the exposure time, and identifying the break in linearity. An example of this plot is shown in Figure 1.1.3.

Thermal cameras allow a non-contact temperature measurement of an object. This is particularly useful in the context of classification, for instance a pedestrian can easily be identified in the dark. To obtain the temperature of the source, the pixel intensity needs to be mapped to the temperature of the source. Radiometric
Figure 1.4: Exposure vs Mean pixel value plot averaged over multiple images. It is noticeable that the plot is linear until an exposure time of 5 ms before entering a region of non-linearity.

calibration involves finding this mapping. A temperature controlled black body source is viewed by the camera and the relationship between the radiometric value and the pixel values are then derived by fitting a parametric curve on the observed data \[80\].

1.1.4 IMU/GPS

Inertial Measure Unit (IMU) and the Global Positioning Unit (GPS) are sensors that measure the robot’s internal state such as the orientation and position. The
inertial measurement unit is composed of an orthogonal triad of accelerometers and gyroscopes. IMU calibration involves determining the biases in each direction and correcting them. In a sensor fusion system consisting of an IMU and a camera, the IMU-camera transform is required to represent the estimated poses in a global frame. An IMU-camera system is commonly used in quadcopters. GPS calibration involves determining the phase center variations of the GPS antenna. The phase center varies depending on the direction of the signal from the satellites. These variations can result in errors in the order of 10 cm. But this is insignificant in the context of robot mapping applications, so these errors are typically ignored.

1.2 Multi-modal Sensing

Each sensor described in the previous section has its own advantages and limitations. A robot that needs to operate in a real world environment clearly needs a combination of different sensors. And true autonomy can be achieved only by a combination of various sensors and their intelligent fusion. The fusion requires the development of calibration algorithms that can represent all the gathered data in a
global frame. To combine LIDAR and image data, the relative pose between the
LIDAR and the cameras must be known. To combine information from an IMU and
a camera, the extrinsics between them must be known. Thus, calibration is a fun-
damental problem in multi-modal sensing. The calibration setup and the algorithm
varies for each combination of sensors being used. The first part of this thesis focusses
on the calibration of a multi-modal sensing system consisting of a thermal camera, a
color camera, and a LIDAR. We present a unified setup and an algorithm that finds
the extrinsics between the cameras and the LIDAR. Calibration procedures do have
some inherent errors and understanding these errors help us in interpreting the fused
data. This thesis presents a setup to quantify the calibration errors w.r.t to the data
observed after the fusion.

1.3 Registration

To build a representation of the robot’s environment (i.e. a map), the gathered
data must be spatially aligned in a single global frame. Registration algorithms like
the Iterative Closest Point [10] and the Normal Distribution Transform [47] allow the
alignment of range scans in a single frame. These algorithms assume that a good
initial estimate of the transformation between successive scans is available. However,
this may not be available all the time. Moreover, mapping can be made flexible if maps
that are acquired temporally can be merged to form a single map. This is especially
useful when building large scale maps. Map merging can be viewed from a global
registration perspective where two 3D models (point clouds) are aligned by computing
the rigid body transformation between them. Map merging is equivalent to aligning
3D maps that were acquired temporally. Global registration is a challenging problem
because (i) there is a varying degree of overlap between the point clouds (ii) the
initial estimate of the rigid body transformation between the point clouds is usually
unknown. The second part of this dissertation provides a solution to this problem using a geometric approach that finds a congruent set of five points constituting a pyramid. The congruency tests are based on the following properties of a rigid body transformation: the distance between points do not change, the ratio of lengths do not change.

### 1.4 Change Detection

Mapping has many applications beyond robot autonomy. To name a few examples: photogrammetry, where maps of the surface of the earth are built using aerial photographs; archeology, where 3D maps of the artifacts are recorded to be preserved and monitored for detecting damages; geology, where soil erosion and other natural processes can be monitored and analyzed. Change detection is a very important application domain for mapping. Change detection applications range from remote sensing, disaster management, surveillance, medical imaging etc. In the pure sciences community, change detection applications help in understanding the relationships and interactions between human and natural phenomena. Changes are detected by repetitive data acquisition, followed by a processing of the sensed data. Typical change detection applications are forest or vegetation change, deforestation, forest fire, environmental change (flood monitoring, land-slide changes), crop monitoring etc. Recently, there has been a lot of interest in the Geological Sciences community to use map differencing techniques to quantify the surface displacements caused by an earthquake. The differencing of airborne LIDAR data of an earthquake site prior and post an earthquake reveals the vertical displacements and the altitude changes in the landscape caused by an earthquake. But identifying the 3D surface displacements can reveal the tectonic movements that caused the earthquake. To this end, the third part of the thesis describes a technique that recovers the 3 dimensional surface
displacements from pre- and post-earthquake LIDAR data.

1.5 Contributions

The contributions in this dissertation are follows

1. **Cross-Calibration**: A unified setup and an algorithm is presented to find the extrinsic parameters between a thermal camera, a color camera, and a LIDAR. The extrinsic parameters are estimated by formulating a non-linear optimization function that aligns the edges of a circular target in the LIDAR point cloud with the edges in the image. The distance transform of the edges in the image is used to compute the residual in the optimization function, which vastly improves the convergence basin of the optimizer.

2. **Analysis of Coloring errors after cross-calibration**: The various factors contributing to coloring/texturing error after cross-calibration are identified as (i) LIDAR noise (ii) error in intrinsic parameters of the camera (iii) error in the relative rotation and translation between the LIDAR and the cameras. The contribution of these errors to the coloring error was analyzed theoretically the following conclusions were made — (i) Errors due to LIDAR noise, intrinsic parameters of the camera, and the relative translation between the LIDAR and the camera decrease with depth and disappear after particular depth. (ii) Errors in the relative rotation between the LIDAR and the camera increase with distance. The solution to the depth at which the coloring errors disappear due to (i) is computed as the root of a second degree equation in depth. Additionally, a setup to quantify the coloring errors after cross-calibration is presented. This setup is then used to compare the results of our method to existing methods.
3. **Initial Alignment Method for Point Cloud Registration**: A initial alignment method to merge 3D maps (point clouds) is developed which finds congruent structures in the point clouds. The congruency test is based on the properties of a rigid body transformation. The algorithm has $O(n^2)$ complexity and even merges maps representing a flat topography.

4. **Change Detection Applied to Earthquake Scenarios**: A technique to recover the complete 3D surface displacements caused by an earthquake is developed. The technique is based on the Iterative Closest Point (ICP) algorithm and was able to recover surface displacements caused by the El Mayur Cucupa earthquake in Mexico (2010) and the Fukushima earthquake in Japan (2011).

1.6 Publications

1. “Cross-Calibration of RGB and Thermal Cameras with a LIDAR for RGB-Depth-Thermal Mapping”

   **Aravindhan K Krishnan** and Srikanth Saripalli

   In *Journal of Unmanned Systems. (Submitted)*

2. “RGB-Depth-Thermal Mapping of Outdoor Environments”  
   **Aravindhan K Krishnan** and Srikanth Saripalli


3. “Cross-Calibration of RGB and Thermal Cameras with a LIDAR”  
   **Aravindhan K Krishnan**, Benjamin Stinnett, and Srikanth Saripalli

4. “Coseismic fault zone deformation revealed with differential LiDAR: examples from Japanese Mw 7 intraplate earthquakes”
Edwin Nissen, Tadashi Maruyama, J. Ramon Arrowsmith, John R. Elliott, Aravindhan K Krishnan, Michael E. Oskin, and Srikanth Saripalli

5. “Three-dimensional coseismic surface displacements and rotations from pre- and post-earthquake Lidar point clouds”
Edwin Nissen, Aravindhan K Krishnan, Ramon Arrowsmith and Srikanth Saripalli.
In *Geophysical Research Letters (GRL)*, 2012

6. “Point Cloud Registration Using Congruent Pyramids”
Aravindhan K Krishnan, Srikanth Saripalli.

7. “NIR-CAM : Development of a Near Infrared Camera”
Aravindhan K Krishnan, Srikanth Saripalli, James F Bell.
In *IEEE International Symposium on Robotics and Sensor Environments (ROSE)* 2013

8. “3D change detection using low cost aerial imagery”
Aravindhan K Krishnan, Srikanth Saripalli, Edwin Nissen, Ramon Arrowsmith.
In *IEEE International Symposium on Safety, Security, and Rescue Robotics (SSRR)* 2013


13
Aravindhan K Krishnan, Edwin Nissen, Srikanth Saripalli and Ramon Arrowsmith.

In *International Symposium on Experimental Robotics (ISER) 2012*
Chapter 2

CROSS CALIBRATION OF RGB AND THERMAL CAMERAS WITH A LIDAR

Calibration refers to the operation that, under specified conditions establishes a relationship between quantity values along with measurement uncertainties [3]. Calibration improves the accuracy and the quality of the data acquired. Hence a good calibration is essential for any sensing system. In the context of Robotics, sensing modalities fall under two categories [71] — proprioceptive sensors and exteroceptive sensors — and the calibration procedures vary accordingly.

Proprioceptive sensors measure values internal to the robot such as motor velocity, robot heading etc. Examples of proprioceptive sensors include IMU, GPS, Magnetometer etc. Exteroceptive sensors obtain information about the robot’s external environment and these sensors can be ‘active’ or ‘passive’. Active sensors emit signal and measure the properties of the returns (e.g. LIDAR, TOF camera, and SONAR) whereas ‘passive’ sensors measure the ambient signal in the environment (e.g color and thermal cameras). IMU calibration [85] involves determining the biases and sensitivities of the orthogonal triad of the accelerometers and gyroscopes. Camera calibration involves correcting for the pixel to pixel sensitivities [70] and estimating the intrinsic parameters [86] such as focal length, center of the image, skew etc. Radiometric calibration [15] of thermal cameras map the pixel intensity values to the temperature of the source. The intensity calibration of a LIDAR maps the LIDAR intensity returns to the reflectance of the surface [42, 14]. In multi-beam LIDARs, calibration involves estimating the offsets between the individual beams [42]. These calibration procedures ensure that the data obtained from the sensors is of
Multi-modal sensing presents alternate ways to solve sensing problems in Robotics and in some cases simplifies the problems to be solved. For instance, a radiometrically calibrated thermal camera is capable of detecting pedestrians and other animals purely based on their body temperature, which is much simpler than training machine learning algorithms. More descriptive place signatures can be derived from multi-modal data which can potentially help place recognition and change detection in urban environments. In multi-modal systems, calibration is necessary to register the data obtained from the various sensing modalities as well as correcting errors in the individual sensors. For instance, in a LIDAR-camera system \cite{28, 77}, the relative pose between the LIDAR and the camera is essential to register the range data from the LIDAR with the color data from the camera. In a IMU-camera system, correcting the IMU biases is essential for accurate pose estimation and the IMU-camera transform is necessary to represent the estimated poses in a global frame \cite{35}. In this chapter, we present a multi-modal sensing system comprising of a color camera, a thermal camera, and a LIDAR. The calibration problem that we are trying to solve is the estimation of the relative poses between the RGB and thermal cameras and the LIDAR to produce RGB-Depth-Thermal (RGBDT) data.

\cite{69, 34, 41} have demonstrated methods to combine data from RGB cameras and LIDAR to obtain RGBD data. But there has not been much work on fusing thermal data with LIDAR. It presents new applications like navigation and scene understanding in the dark. And to the best of the authors’ knowledge only Nuchter et al \cite{12} present a work that merges data from a LIDAR, thermal, and RGB cameras to create a RGB-Depth-Thermal (RGBDT) map. Recently, there has been work on obtaining
RGBDT data from a thermal camera and a Kinect [46], [81]. These approaches are applicable only for indoor environments. To obtain RGBDT data of outdoor environments, a couple of problems have to be solved: A) Computing the intrinsic parameters of the thermal camera. B) Computing the extrinsics between the thermal/RGB cameras and the LIDAR. As a solution to problem A, we describe our calibration target that looks like a checkerboard pattern in the thermal image. And for problem B, we present our extrinsic calibration method that aligns edges in the thermal/RGB images with the edges in the LIDAR. Since we are calibrating both a color camera and a thermal camera with a LIDAR, we propose a single setup that can be used for cross-calibration of both the thermal and RGB cameras.

Typically, cross-calibration algorithms define an objective function parameterized in terms of the extrinsic parameters which is then minimized or maximized. They fall under two categories (i) error minimization methods (ii) information maximization methods.

Error minimization methods either reduce the reprojection error when the LIDAR points are projected onto the image (or) minimize the alignment error of planes defined in the camera frame and the LIDAR frame. Information maximization methods represent the scene content in various sensing modalities as intensity distributions and maximize the correlation between them. Our method falls under the category of error minimization methods. We define a cost function and a calibration setup that
can be used for the cross-calibration of both RGB and thermal cameras.

2.1 Related Work

To merge data from a LIDAR and a camera, the extrinsic parameters (i.e. rotation and translation parameters) between the LIDAR and the camera should be known. Typically, finding the extrinsics is formulated as an optimization problem that computes the transformation (T) between the camera frame and the LIDAR frame. The first approaches to camera-LIDAR calibration used calibration targets similar to camera calibration. The calibration target is placed in the common field-of-view (FOV) of the camera and the LIDAR. The equation of the plane (in the LIDAR frame) containing the planar calibration target is computed after segmenting the LIDAR point cloud [78]. For a 2D LIDAR, a line is used instead of a plane [58]. The equation of the plane in the camera frame is computed from the extrinsics between the camera frame and the world frame (which is defined on the calibration target). The required transformation T is obtained by aligning the plane (or the line in [58]) in the LIDAR frame to the plane in the camera frame. Recently, a similar approach ‘single shot calibration’ has been presented in [25]. The toolbox that implements ‘single shot calibration’ also provides an approach to detect checkerboard corners automatically. The toolbox requires that many checkerboards are placed in the scene at once, rather than having to collect multiple scans / images of the checkerboard placed in different orientations. The above approaches rely on the plane equations of the calibration target obtained from the camera calibration to compute the cross-calibration parameters; hence are not suitable for online calibration.

Another approach is to find the pose of the camera in the LIDAR frame using the perspective-from-n-points (PnP) algorithm [69]. Corresponding points between the
camera image and the LIDAR point cloud are selected by the user using an intermediate Bearing Angle Image (derived from the point cloud).

Few other recent approaches to camera-LIDAR calibration assume that the camera is calibrated and the intrinsics of the camera are known. The LIDAR point cloud (with intensity values) is projected to the camera frame using the camera intrinsics resulting in an intensity image. The cross-calibration problem is then formulated as an optimization problem that finds the transformation (T) which aligns the LIDAR intensity image with the camera image.

Algorithms that use mutual information as the optimization function are presented in [57] and [74]. These methods rely on the intensity values from the LIDAR which may not be accurate. Pandey et al [57] calibrate the intensity returns for their Velodyne LIDAR before doing the cross-calibration. Mutual information methods are targetless and are suitable for online calibration. However, the mutual information metric is not very useful when a high resolution image (e.g. 3 MP image) is used with a sparse point cloud containing a few thousand points. The few thousand points, when projected back to the image, contribute only to a fraction of pixels in the intensity image and the other pixels have to be interpolated [53]. So, an averaging of information occurs before the mutual information is maximized. This can result in poor convergence.

An edge alignment approach is presented by Levinson et al [41] where the edges in the LIDAR point cloud are projected on to the image. The optimal parameter \( T_{opt} \) is determined from an initial estimate (T) by a brute force search in a discrete set of points \( \mathcal{P} \in [T-\Delta, T+\Delta] \) that projects the edges in the LiDAR data to the edges in
the image. The assumption here is that the edges in the point cloud which arise out of depth discontinuities have corresponding edges in the image. A very good initial estimate of the parameters is required for this solution to work as one can easily get stuck in a local minima with a local brute force search.

A calibration technique using a circular calibration target is presented in [7]. The plane containing the circle is segmented from the LIDAR point cloud and the pose of the circle is estimated in the LIDAR frame. The pose of the circle in the camera frame is estimated as described in [56]. The circles are then aligned using the ICP point-to-plane metric. Another method that uses circles as a calibration target is presented by Martin et al [79].

Thermal cameras are used in various applications such as surveillance, building inspection, gas detection, human detection etc, but are rarely used in the context of 3D mapping. A survey of the applications of thermal cameras can be found in [24]. 3D Mapping using stereo thermal cameras is described in [65] and monocular SLAM using a thermal camera is presented in [82]. Nuchter et al present their work on integrating thermal camera and a LIDAR in [12]. RGBDT data provides a rich set of applications to autonomous systems such as pedestrian detection, change detection, localization etc. And with the cost of thermal cameras going down in recent years, it presents a new mode of sensing for future robotic systems.

The first step to build a thermal 3D mapping system is to find the intrinsic parameters of the thermal camera. Various approaches have been tried to create a calibration target that appears like a checkerboard pattern in the thermal image. A black body radiation source with a cold metal grid placed over it can produce a hot
and cold pattern that resembles a calibration target. An LCD monitor being used as the black body radiation source is presented in [73]. In [4] a cut checkerboard was made and placed in front of a human subject and the heat radiation from the human body produced the background illumination to create a checkerboard pattern.

A method to create RGB-Depth-Thermal point cloud is presented in [27]. The calibration target here consists of a metal board embedded with an evenly spaced array of resistors connected to a 12 volt DC supply. On being powered, the resistor array produces a thermal gradient necessary to detect the corners. These corners are used for camera calibration. For the thermal-LIDAR cross calibration, the plane containing the calibration target is segmented. Next, the known dimensions of the calibration target are then used to interpolate the 3D co-ordinates of the resistors. The 2D-3D correspondence between the points are then used to compute the extrinsic parameters. A thermal stereo system is presented in [63]. For intrinsic calibration, a checkerboard pattern is heated with a flood lamp and the emissivity difference between the black and white regions provides the necessary thermal gradient to detect the corners. Another thermal stereo system is presented in [32]. The calibration target consists of a checkerboard pattern which is embedded with thin wires along the edges of the checkerboard squares. Current flowing through the wires causes the wires to heat and the wire intersections provide the checkerboard corners for thermal camera calibration. An RGBDT data stream of indoor environments is created in [46] by combining data from the Kinect sensor with a thermal camera. Their approach aligns the thermal image with the RGB image from the Kinect, producing RGBDT data. Since the depth information for a pixel is already available from the Kinect, their problem is reduced to an image alignment problem between the two cameras, without having to align the depth image. A similar hand held setup which creates
Figure 2.1: The Setup Consisting of a Hokuyo LIDAR, PointGrey Color Camera, and a FLIR Tau2 Thermal Camera

(a) The target made produced by cutting squares of black and white melamine in a laser cutter and gluing them alternatively on a board.

(b) Image of the calibration target in the thermal camera. The intensity contrast was good enough to detect the checkerboard corners.

Figure 2.2: Calibration Target used for Thermal and RGB Camera Calibration.

RGBDT maps of indoor environments is presented in [81].

2.2 Method

Our setup consists of a PointGrey CM3-U3-13S2M-CS color camera, FLIR Tau2 (long-wave-infrared, uncooled core) thermal camera, and a Hokuyo UTM-30LX LIDAR. To get 3D point clouds, the scanner is rotated using a Robotis Dynamixel PRO.
(a) Image of the calibration setup in the RGB camera. The setup in the thermal camera.
black cloth provides the necessary contrast to detect the circle. The wet (black) cloth provides
the necessary contrast to detect the circle.

Figure 2.3: Calibration Target used for Thermal and RGB Camera Calibration.

(a) Color image - Detected edge shown in red  (b) Thermal image - Detected edge shown in red

Figure 2.4: Detecting the Edges of the Circular Target in the Image
motor which gives \( \sim 12 \) encoder readings per degree. The cameras are rigidly mounted to the frame containing the LIDAR. An image of the system is shown in Figure 2.1.

To obtain a 3D point cloud, we integrate the encoder readings from the motor and the 2D scan from the LIDAR based on the time stamps of corresponding ROS topic messages. 2D scans in one tilt of the LIDAR are accumulated to create a point cloud. We assume that there are no errors in the encoder readings and the scene remains static during one tilt of the LIDAR (in the up-down tilt motion of the
motor). Our cross-calibration method assumes that the intrinsic parameters of the cameras are known. To compute the intrinsic parameters of the thermal camera, we designed a calibration target that looks like a checkerboard pattern in the thermal image. Initially, we describe our calibration target for the thermal camera. Later, we describe the setup for the extrinsic calibration and finally the algorithm for the cross-calibration.

2.2.1 Calibration Target for the Thermal Camera

The target was made by cutting squares of black and white melamine in a laser cutter and gluing them alternatively on a board. First, a black rectangular frame is cut and glued to the board – which acts as a border. The squares (2 × 2 inches) are then glued inside, starting from one of the corners, ensuring that they are placed tightly without any gaps in between. Because of the color contrast, the same target is used for RGB camera calibration too. When placed in the sun, the different albedo of the white and black squares creates the thermal gradient needed for camera calibration.

The images of the calibration target in the color camera and the thermal camera is shown in Figures 2.2(a) and 2.2(b) respectively. We use the RADOCC toolbox \([33]\) for camera calibration. As seen in 2.2(b), the intensity contrast in the thermal image was good enough for detecting the checkerboard corners. 40 images of the calibration target were obtained from various orientations and we achieved a reprojection error of 0.24 pixels.
Algorithm 2: Detecting edges of the circular target in the point cloud

**Input:** 1) Point Cloud $FR$ represented in cartesian as well as spherical co-ordinates 2) Clicked point $p$ 3) Plane fit threshold $\delta$ 4) Edge detection threshold $\theta$

**Output:** 3D edge points $E$ of the circular calibration target

1. $FR_F = SACPlaneSegmentation (FR)$
2. $FR_B = FR - FR_F$
   
   /* If the plane in the background is chosen as the foreground in the previous step */
3. if $p \in FR_B$ then
4.   Swap ($FR_B$, $FR_F$);
5. end

   /* Removing the noisy edge points */
6. $\mathcal{C} = RANSAC_{PlaneFit} (FR_F, \delta)$;
7. $\Phi = ComputeUniquePhiList (\mathcal{C})$;

   /* Sorting points by scan line */
8. $P = \emptyset$
9. foreach $\phi \in \Phi$ do
10.   $p_s = \text{FindPointsWithPhi} (\phi)$
11.   $p_s = \text{SortByTheta} (p_s)$
12.   $P = P \cup p_s$
13. end

   /* Detecting the edges of the circle */
14. $E = \emptyset$
15. foreach $p_s \in P$ do
16.   foreach $p_i^s \in p_s$ do
17.     if $|p_i^s.theta - p_i^{s+1}.theta| \geq \theta$ then
18.       $E = E \cup p_i^s$
19.       $E = E \cup p_i^{s+1}$
20.     end
21.   end
22. end

### 2.2.2 Cross Calibration Setup

The cross-calibration algorithm aligns the edges of a circular target in the LIDAR to its corresponding edges in the color and thermal images. Hence we designed a setup that facilitates the accurate detection of edges in the different sensing modalities. A circle of known radius was cut out from a white cardboard. The cardboard (which can be of any color) is then placed before a background of a contrasting color. We draped a wet black cloth over a garment rack and used it as a background. The black color produces a contrast for the color camera and the wetness of the cloth produces the intensity contrast in the thermal image. The edges of the circle were detected
Algorithm 3: Generating points on the 3D circle

Input : 1) Detected edge points \( \mathcal{E} \). 2) Radius of the circle \( r \). 3) Output circle resolution \( \Delta \theta \).

Output: 3D edge points \( \hat{\mathcal{E}} \) of the circular target at a resolution \( \Delta \theta \)

1. \( \Sigma = \text{ComputeCovariance (} \mathcal{E} \text{)} \);
2. \( \{ V_1, V_2, V_3 \} = \text{EigenVectors (} \Sigma \text{)} \)

\[
\hat{c} = \arg \min_c \sum [p_i - c]^T [p_i - c] - r^2 + n \cdot [p_i - c] \text{ where } p_i \in \mathcal{E}.
\]

3. \( \hat{c} \) is the normal vector corresponding to the smallest eigen value.

4. Points on the XY plane
   \[
   \hat{\mathcal{E}}_{2d} = \emptyset
   \]
   \[
   \text{for } (\theta = 0; \theta \leq 360; \theta += \Delta \theta) \text{ do}
   \]
   \[
   p_{2d} = [r \cos(\theta), r \sin(\theta), 0]^T
   \]
   \[
   \hat{\mathcal{E}}_{2d} = \hat{\mathcal{E}}_{2d} \cup p_{2d}
   \]
   \[
   \text{end}
   \]

5. Points on the 3D circle

6. \( \hat{\mathcal{E}} = \emptyset \)

7. \( \text{foreach } p_{2d} \in \hat{\mathcal{E}}_{2d} \text{ do} \)

8. \( p_{3d} = [p_{2d} \cdot (V_1), p_{2d} \cdot (V_2), p_{2d} \cdot (V_3)]^T + \hat{c} \)

9. \( \hat{\mathcal{E}} = \hat{\mathcal{E}} \cup p_{3d} \)

10. \( \text{end} \)

11. \( \hat{\mathcal{E}} = \hat{\mathcal{E}} \cup p_{3d} \)

12. \( \text{end} \)

13. \( \hat{\mathcal{E}} = \hat{\mathcal{E}} \cup p_{3d} \)

14. \( \text{end} \)

across all the sensing modalities and the cross-calibration algorithm computes the extrinsics that aligns the edges in the images with the edges in the LIDAR. Images of the calibration setup for the color and thermal cameras are shown in Figures 2.3(a) and 2.3(b) respectively.

2.2.3 Detecting Edges in the Image

We use a region growing algorithm to detect the edges of the circle in the color and thermal images. The user clicks a point within the circle and the clicked point is used as the seed point for the region growing segmentation. The region is grown by adding the neighboring pixels to the segment when the pixel intensities are similar. The boundary of the grown segment represents the edges of the circle in the image. The detailed work flow is described in Algorithm 1.

To begin with, our algorithm creates a binary image that is of the same size of
the input image. All pixels in the binary image are initialized to 1. A queue is used to keep track of the recently added pixels to the region. Initially, the seed point is pushed to the queue. The 8 neighbors of the seed point are compared with the seed point and are pushed to the queue if the intensities match within a given threshold. Whenever a point is pushed to the queue the corresponding pixel in the binary image is set to 0. After the neighbors of a pixel are compared for similar intensity values, the pixel is removed from the queue. This process is repeated until the queue becomes empty. The set of all pixels that were pushed to the queue represents the circular region and the corresponding pixels in the binary image have 0 intensity values.

A list of points \( L \) that were pushed to the queue is maintained. This list is used to detect the boundary of the estimated segment and the boundary corresponds to the edges of the circular target. For every point in \( L \), the corresponding neighbors in the binary image is considered. It is obvious that pixels that have at least one non-zero neighbor constitute the boundary of the circular region and these pixels correspond to the edges of the circular target. Figures 2.4(a) and 2.4(b) show the detected edges in color and thermal images respectively.

2.2.4 Detecting Edges in the LIDAR Point Cloud

The edge detection algorithm requires that the point cloud is represented in cartesian \((x, y, z)\) as well as spherical co-ordinates \((r, \theta, \phi)\). The range \(r\) and scan angle \(\theta\) are obtained from a single scan in the 2D laser scanner and the tilt angle of the motor \(\phi\) is obtained from the motor. Initially, the user clicks on a point \(P\) on the cardboard containing the (cut-out) circle. Points that are within a radius \(R\) of the clicked point are filtered and the circle edges are detected in the filtered segment. (\(R\) is chosen
sufficiently large that it covers the size of the cardboard containing the circle). The work flow is described in Algorithm 2.

Since the filtered region \( (\mathcal{FR}) \) represents points that are within a radius \( R \) of the clicked point, it also includes points that are not on the cardboard, i.e., points that are on the black cloth in the background. So the filtered region contains two planes — one representing the cardboard and another representing the cloth in the background. As a first step, we extract the plane containing the cardboard from the filtered region.

Sample Consensus (SAC) segmentation method is used to detect ‘a’ plane in the filtered region — this can be either the cardboard in the foreground \( (\mathcal{FR}_F) \) or the cloth in the background \( (\mathcal{FR}_B) \). The implementation from the Point Cloud Library [67] is used for the SAC segmentation. Given \( \mathcal{FR}_F \), one can compute \( \mathcal{FR}_B = \mathcal{FR} - \mathcal{FR}_F \). Similarly, \( \mathcal{FR}_F \) can be computed given \( \mathcal{FR}_B \) and \( \mathcal{FR} \). The region (among \( \mathcal{FR}_F \) and \( \mathcal{FR}_B \)) containing the clicked point \( P \) is the plane containing the circular target and is further considered for detecting the edges.

To filter out any noisy edge points in the region obtained in the previous step, we again fit a plane using RANSAC on this region with a plane fitting threshold of 2 cm. This filters out all the noisy edge points and we get a new set of points \( C \). The new region is shown in Figure 2.5(a).

Since we also maintain the spherical co-ordinate representation for the LIDAR points, we consider points in \( C \) scan line by scan line (a scan line represents all points \((r, \theta, \phi)\) for a particular \( \phi \)) and look for any discontinuities in \( \theta \) for adjacent points in a scan line. These discontinuities correspond to the background points that were filtered in the Sample Consensus step described earlier. And the points at the point
Figure 2.5: Detecting the edges of the circular target in the LIDAR point cloud

of discontinuity represent the edge points ($E$) of the circle. The detected edge points can be seen in Figure 2.5(b).

2.2.5 Generating All Points on the Circle

As seen in Figure 2.5(b), the edge detection method described in the previous section does not compute all edge points of the circle. To generate the remaining points on the circle, we estimate the parameters of the circle by doing a least squares fit on the detected edge points ($E$). The parameters of a circle are the radius ($r$) and the center ($c$). In our case $r$ is known, so we estimate the center using least squares. The least squares cost function that we minimize is given below, where $p_i \in E$ are the
detected edge points, $n$ is the normal to the plane containing the edge points and $c$ is the parameter to estimate.

$$f(c) = \sum_i [p_i - c]^T[p_i - c] - r^2 + n \cdot |p_i - c|$$  \hspace{1cm} \text{(2.1)}$$

The above function is convex, so minimizing this function gives a unique minima. We use Levenberg-Marquardt algorithm to minimize the cost function and obtain the center of the circle ‘c’. Part A ensures that the resulting center lies on the plane containing the edge points. ($n$ is the normal to the plane and $[p_i - c]$ is a vector on the plane containing the circle. So the dot product $n \cdot [p_i - c]$ should be 0).

To generate the points on the 3D circle, we first generate points ($E_{2d}$) on a 2D circle (XY plane) from the computed parameters. We then compute the orthogonal basis vectors ($V_1, V_2, V_3$) of the detected edge points ($E$). The basis vectors are given by the eigen vectors of the covariance matrix of $E$. The points in $E_{2d}$ are projected on to the orthogonal basis vectors $V_1, V_2, V_3$ to obtain the points on the 3D circle ($\hat{E}$). The procedure is described in Algorithm 3 and the result in shown in Figure 2.5(c).

2.2.6 Computing the Extrinsics Between the Cameras and the LIDAR

The circular target is placed in different orientations and places in the common field of view of the LIDAR and the cameras. The relative transformation is computed by aligning the edges of the circular target in the LIDAR point cloud to the edges of the target in the images. We solve the edge alignment problem as an optimization problem. The optimization function is shown below.
Here, the index \( j \) refers to the ‘image – point cloud’ pair containing the same view of the calibration target. \( E_j^i \) are the edge points of the circular target in the \( j^{th} \) point cloud. \( T \) is the parameter we are estimating. \( K \) is the camera matrix containing the intrinsic parameters. \( \mathcal{N}(\cdot) \) produces the normalized image co-ordinates. \( I_j^c \) is the closest edge point in the \( j^{th} \) image, when \( E_j^i \) is projected on to the camera.

The optimization function in Equation 2.2 is implemented as a distance transform of the edges computed on the image. For an edge image, the distance transform of a pixel gives the distance to the closest edge pixel. When the 3D point is projected to a pixel \((x,y)\) in the image, the corresponding distance transform at \((x,y)\) gives the distance to the closest edge pixel in the image. When the distance transform is computed only on the edges of the circular target, it is precisely the optimization function in Equation 2.2. An example distance transform is shown in Figure 2.6.

One can notice that the distance transform looks convex for most part as the distance to an edge pixel grows gradually towards the boundary of the image. We used the Levenberg-Marquardt (from Ceres solver \cite{12}) algorithm for optimization.
We used numerical differentiation to compute the gradients for minimizing Equation 2.2. The *central difference* method for computing the gradient of a function $f$ at a point $T$ is given by $\frac{f(T+\Delta)-f(T-\Delta)}{2\Delta}$, where $\Delta$ is usually chosen to be $\sim$1e-6. But the optimization function in Equation 2.2 is a distance transform and both $f(T)$ and $f(T+\Delta)$ result in the same pixel $x$ when $\Delta = 1e-6$, producing a zero gradient. This causes the optimization algorithm to terminate early.

To overcome this, we fit a plane on the distance transform values for a $10 \times 10$ window centered around each pixel $(T)$. The plane represents a parameterized continuous surface and we can obtain the objective function values even for small deviations $\Delta$ centered around a pixel $T$. With this surface approximation, the *central difference* method for computing gradients is possible and the optimizer converges. The plane is computed only on pixels onto which the LIDAR points are projected. The plane parameters (computed per pixel) are stored in a look-up table to avoid re-computation on future iterations of the optimization.
2.3 Results & Discussion

Images of a point cloud textured by our algorithm with color and thermal images are shown in Figures 2.7(a) and 2.7(b) respectively. In this section, we discuss various aspects of the cross-calibration problem. We begin with a discussion on the smoothness/convexity of the objective function in Equation 2.2. We then report the timing analysis of our solution. Later, we discuss the various factors that can result in the wrong coloring of a 3D point after cross-calibration. We discuss a setup that allows us to quantify the coloring errors and compare our solution to the method described in [57] which is based on maximizing mutual information. We hypothesize why mutual information methods do not work for relative pose estimation between a thermal camera and a LIDAR. Finally, we present some results on our RGB-Depth-Thermal mapping in outdoor environments during day and night.

2.3.1 Convexity of the Objective Function

The optimization function in Equation 2.2 is a non-linear function, as the transformation matrix $T$ includes sine and cosine terms. So we investigated the convexity of the function by varying the number of point cloud - image pairs used for optimization. Figures 2.8(a) and 2.8(b) show the variation of the objective function as we vary one extrinsic parameter (roll, pitch, yaw, tx, ty, and tz) while keeping others constant. It is noticeable that the function becomes smooth as the number of scans is increased. The smooth function ensures that the optimizer does not encounter discontinuous points on the objective function, thus helping convergence. We observed that $\geq 8$ scans were sufficient for the optimization.
<table>
<thead>
<tr>
<th>roll</th>
<th>pitch</th>
<th>yaw</th>
<th>tx (m)</th>
<th>ty (m)</th>
<th>tz (m)</th>
<th>Time taken (s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>15</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>17.436</td>
</tr>
<tr>
<td>10</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>19.455</td>
</tr>
<tr>
<td>5</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>20.077</td>
</tr>
<tr>
<td>0</td>
<td>15</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>22.545</td>
</tr>
<tr>
<td>0</td>
<td>10</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>19.946</td>
</tr>
<tr>
<td>0</td>
<td>5</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>17.063</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>15</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>22.305</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>10</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>18.75</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>5</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>15.204</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1.25</td>
<td>0</td>
<td>0</td>
<td>14.48</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0.5</td>
<td>0</td>
<td>0</td>
<td>21.458</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1.25</td>
<td>0</td>
<td>0</td>
<td>24.898</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0.5</td>
<td>0</td>
<td>14.13</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1.25</td>
<td>1.25</td>
<td>0</td>
<td>13.174</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1.25</td>
<td>24.898</td>
</tr>
</tbody>
</table>

**Table 2.1:** Convergence Analysis: This Table Reports the Errors Observed When the Optimizer was Initialized with Different Initial Estimates. We observe that our Method Converges when the Initial Estimates are off by 15 degrees for rotation Parameters and by 1.25m for the Translation Parameters. The Maximum Time Taken for Convergence in these Experiments is 36 seconds.
(a) 2 scans were used to create the above plot. The function is not smooth.

(b) 10 scans were used to create the above plot. The function becomes smooth as the number of scans are increased.

**Figure 2.8:** (a) and (b) : The Variation of Parameters Roll, Pitch, Yaw, Tx, Ty, and Tz Around the Global Minimum. Roll, pitch, and Yaw were Varied by -30 Degrees to +30 Degrees Around the Global Minimum. Tx, Ty, and Tz were Varied by -3m to +3m Around the Global Minimum. The Y Axis Shows the Objective Function Values. One Can notice that the Function Appears Smooth and There are no Steep Local Minima

### 2.3.2 Convergence Analysis

To test the convergence of the algorithm, we simulated a test dataset consisting of circles placed at different orientations. A camera with known intrinsics and extrinsics was defined and the 3D points were projected on to the image. The distance transform of the projected circles were then computed. The extrinsics were recovered from the optimization and compared to the ground truth transformation.

Table 2.1 presents the errors observed for various values of the initial estimates along with the runtime. 8 circles defined by 360 points each were used in these tests, so a total of 2880 points were used. We did not observe a huge difference in the runtime for various initial configurations, with the maximum time taken being ~36 seconds.
2.4 RGB-Depth-Thermal Mapping

Initially, we computed the extrinsics between the cameras and the LIDAR using the approach described in this chapter. Then, we mounted our setup on a cart and collected data for RGBDT mapping of outdoor environments. For each point cloud obtained from the LIDAR, we apply the computed extrinsic parameters and project them on to the RGB and thermal images. We find the corresponding RGB and thermal texture for each point in the point cloud. We then pass this textured point cloud to our 3D mapping pipeline (which is implemented using the Iterative Closest Point [10] algorithm) and produce RGBDT maps. The mapping pipeline is shown in Figure 2.9. One point cloud is generated every 600 milliseconds (time taken for
1 tilt of the LIDAR). The frame rates for the RGB and thermal cameras are 5 fps and 25 fps respectively. Figure 2.10 shows the RGB and thermal maps (containing ~1.2 million points) of the same environment during various times of the day. It is noticeable in the thermal maps that the pavement can be differentiated from the building and the trees during the day and the night. So thermal maps can potentially be used to segment roads for urban night driving. We are optimistic that there are interesting applications to look at as we gather more data of urban environments at night.

2.5 Conclusions

We presented an approach to cross-calibrate an RGB and a thermal camera with a LIDAR. Initially, we described a calibration target for the intrinsic calibration of the thermal camera. Next, we described our setup for the extrinsic calibration of the thermal and the RGB cameras with a LIDAR. We presented our calibration algorithm that uses the distance transform of the edges in the image as an objective function, which when minimized gives the extrinsics between the cameras and the LIDAR. Finally, we presented our results on the RGB-Depth-Thermal mapping of outdoor environments using our setup. In the future, we are interested in building maps at a larger scale by mounting the sensor suite on a car and investigate the applications of RGB-Depth-Thermal mapping for autonomous driving.
Figure 2.10: RGB-Depth-Thermal Mapping
COLORING ERROR AFTER CROSS-CALIBRATION : AN ERROR ANALYSIS

Cross-calibration refers to the procedure of finding the relative transformation between the various sensor frames. In the previous chapter, we introduced a method to find the relative transformation between a LIDAR, a thermal camera, and an RGB camera. An interesting question to answer after the cross-calibration procedure is — what factors result in the wrong coloring of a LIDAR point after cross-calibration? In this chapter, we identify the various sources of coloring errors and analytically derive the conditions under which these factors affect the coloring of a LIDAR point. Additionally, we discuss a setup that allows us to quantify the coloring errors.

There are three sources of coloring error after cross-calibration
1. Noisy 3D points (arising from the LIDAR)
2. Error in the camera calibration parameters (K matrix)
3. Error in the extrinsic parameters between the camera and the LIDAR

These errors manifest in coloring a 3D point after cross-calibration. We will analyze each source of error in this section.

3.1 Noisy 3D points (Arising from the LIDAR)

In the context of cross-calibration the reprojection error for a ‘single’ 3D point is only a discrete quantity (as it represents the distance to the closest edge pixel). So the noisy 3D point does not contribute to the reprojection error unless the error
is $\geq 1$. We will base our analysis on ‘When does the noisy 3D point introduce a reprojection error of 1?’.

The projection of a 3D point on an image is given by

$$
\begin{bmatrix}
\hat{x} \\
\hat{y} \\
\hat{z}
\end{bmatrix} =
\begin{bmatrix}
f_x & 0 & c_x \\
0 & f_y & c_y \\
0 & 0 & 1
\end{bmatrix}
\begin{bmatrix}
X \\
Y \\
Z
\end{bmatrix}
$$

(3.1)

Throughout our error analysis, we consider only the $x$ co-ordinate of the point on the image. $x^g$ refers to the ground truth value of the pixel, and $x^p$ refers to the projected point (with noise).

$$
x^g = f_x \frac{X}{Z} + c_x
$$

(3.2)

The projection of a noisy 3D point on the image is given by

$$
\begin{bmatrix}
\hat{x} \\
\hat{y} \\
\hat{z}
\end{bmatrix} =
\begin{bmatrix}
f_x & 0 & c_x \\
0 & f_y & c_y \\
0 & 0 & 1
\end{bmatrix}
\begin{bmatrix}
X + \Delta x \\
Y + \Delta y \\
Z + \Delta z
\end{bmatrix}
$$

(3.3)

$$
x^p = f_x \frac{X + \Delta x}{Z + \Delta z} + c_x
$$

(3.4)

The variation of $|x^g - x^p|$ w.r.t $Z$ is shown in Figure 3.1(a) It is noticeable that $|x^g - x^p|$ reduces with depth. This can be explained from the Equations 3.2 and 3.4

As $Z$ grows $Z + \Delta z \approx Z$, so

$$
|x^g - x^p| \approx |f_x \frac{X}{Z} + c_x - f_x \frac{X + \Delta x}{Z} - c_x| = f_x \frac{\Delta x}{Z}
$$

(3.5)

In the above equation, $f_x$ and $\Delta x$ are constant. As $Z$ increases, the ratio approaches 0, which explains the graph in Figure 3.1(a) The physical interpretation of this phenomenon is that the footprint of a pixel grows with depth, and for a given noise ($\Delta x$), the noisy 3D point lies within the pixel footprint beyond a particular depth. A
Variation of reprojection error with the depth of the 3D point; it decreases with depth. This curve gives the depth at which the reprojection error is 1.

Figure 3.1: Reprojection Error vs Depth of the 3D Point

Reprojection error is introduced when $|x^g - x^p| = 1$ and this happens at a particular depth $Z$ beyond which the reprojection error does not contribute to the coloring error. Now let us derive the depth at which the reprojection error is 1.

\[ |f_x \frac{X}{Z} + c_x - f_x \frac{X + \Delta x}{Z + \Delta z} - c_x| = 1 \]

\[ |f_x XZ + f_x X \Delta z - f_x XZ - f_x \Delta x Z| \frac{1}{(Z)(Z + \Delta z)} = 1 \]

\[ |f_x X \Delta z - f_x \Delta x Z| \frac{1}{Z^2 + Z \Delta z} = 1 \]

Assuming that $\Delta x = \Delta z$

\[ |(f_x X - f_x Z) \Delta z| \frac{1}{Z^2 + Z \Delta z} = 1 \quad (3.6) \]

\[ Z^2 + Z \Delta z = f_x X \Delta z - f_x Z \Delta z \]

\[ Z^2 + Z(\Delta z + f_x \Delta z) - (f_x X \Delta z) = 0 \quad (3.7) \]

which is a quadratic equation in $Z$, where the quantities within the braces are known ($\Delta z$ modelled from LIDAR noise characteristics, $f_x$ known from camera calibration, and $X$ is an arbitrary point chosen such that it is projected on to the image, for e.g. the center of the image). The solution to Equation $3.7$ gives the depth beyond which
the reprojection error is \( \leq 1 \) and hence there is no coloring error. As an example, we considered \( f_x = 1335.054 \) (focal length of our color camera), and \( \Delta z = 0.02 \) (2 cm noise for our LIDAR). We obtained \( Z = 4.303249 \) as the root of Equation 3.7. The quadratic function in Equation 3.7 for the considered values is plotted in Figure 3.1(b). In case of multiple roots to the quadratic equation, the largest root is chosen (negative roots are ignored as depth is a non-negative quantity, and the largest root gives the depth beyond which the reprojection error is always \( \leq 1 \)). Similarly, we obtained \( Z = 3.512751 \) for our thermal camera \( (f_x = 417.20215, \Delta z = 0.02) \).

3.2 Error in the Camera Calibration Parameters (K Matrix)

The reprojection error in camera calibration translates to errors in the intrinsic parameters \( f_x, f_y, c_x, \) and \( c_y \) as well as the extrinsic parameters (between the camera frame and the world frame) \( R \) and \( T \). However, the extrinsic parameters \( R \) and \( T \) will not be used in the cross-calibration. So we consider only the errors in the intrinsic parameters when analysing cross-calibration errors. We can assume that we have a perfect \( R \) and \( T \) and all errors are in the intrinsic parameters. This way, we are considering only the worst case error in the intrinsic parameters for any further error analysis. In this section, we consider the contribution of both the LIDAR noise and the camera calibration error towards the cross-calibration. The projection of a 3D point on an image is given by

\[
\begin{bmatrix}
\hat{x} \\
\hat{y} \\
\hat{z}
\end{bmatrix} =
\begin{bmatrix}
f_x & 0 & c_x \\
0 & f_y & c_y \\
0 & 0 & 1
\end{bmatrix}
\begin{bmatrix}
X \\
Y \\
Z
\end{bmatrix}
\]

(3.8)

\[
x^g = f_x \frac{X}{Z} + c_x
\]

(3.9)
The projection of a noisy 3D point on the image with errors in intrinsic parameters and LIDAR noise is given by

\[
\begin{bmatrix}
\hat{x} \\
\hat{y} \\
\hat{z}
\end{bmatrix} =
\begin{bmatrix}
f_x + \Delta f_x & 0 & c_x + \Delta c_x \\
0 & f_y + \Delta f_y & c_y + \Delta c_y \\
0 & 0 & 1
\end{bmatrix}
\begin{bmatrix}
X + \Delta x \\
Y + \Delta y \\
Z + \Delta z
\end{bmatrix}
\]

(3.10)

\[x^p = (f_x + \Delta f_x)(\frac{X + \Delta x}{Z + \Delta z}) + (c_x + \Delta c_x)\]

(3.11)

A reprojection error is introduced when \(|x^g - x^p| = 1\). We proceed similar to the previous section and arrive at the following equation.

\[Z^2 + Z(f_x \Delta x + \Delta f_x \Delta x + \Delta z) - (\Delta z f_x X + \Delta z \Delta f_x X) = 0\]

(3.12)

Again, this equation provide a constraint on \(Z\) at which the reprojection error for cross-calibration is 1. Things to notice are

1. All terms within the braces are known. \(\Delta f_x\) is given by the output of the camera calibration toolbox. \(X\) is chosen such that the point lies within the image (e.g. the center of the image) and the error analysis is based the chosen value of \(X\).

2. \(\Delta f_x\) provides an additional constraint on \(Z\) similar to the LIDAR noise.

3. Intuitively, one would expect that the solution for \(Z\) would be greater than the one obtained in the previous section.

As an example, we considered \(f_x = 1335.054\) (focal length of our camera), and \(\Delta z = 0.02\) (2 cm noise for our LIDAR), and \(\Delta f_x = 2\) (uncertainty reported by the camera calibration toolbox). We obtained \(Z = 4.304037\) as the root of Equation 3.12. It is noticeable that the \(Z\) obtained here (4.304037) is greater than the \(Z\) obtained in the previous section (4.303249). Similarly, we obtained \(Z = 3.524249\) for our thermal camera (\(\Delta z = 0.02\) and \(\Delta f_x = 6\)).
3.3 Error in the Extrinsic Parameters

For simplicity, let us analyse the error in rotation components and the translation components separately.

### 3.3.1 Error in Rotation Components

When there is a rotation between the LIDAR frame and the camera frame, the projected point on the image is given by

\[
\begin{bmatrix}
\hat{x} \\
\hat{y} \\
\hat{z}
\end{bmatrix} =
\begin{bmatrix}
f_x & 0 & c_x \\
0 & f_y & c_y \\
0 & 0 & 1
\end{bmatrix}
\begin{bmatrix}
X \\
Y \\
Z
\end{bmatrix}
\]  
\tag{3.13}

Now, let us consider only the error in yaw. The projected point on the image when there is no error in yaw is given by

\[
\begin{bmatrix}
\hat{x} \\
\hat{y} \\
\hat{z}
\end{bmatrix} =
\begin{bmatrix}
f_x & 0 & c_x \\
0 & f_y & c_y \\
0 & 0 & 1
\end{bmatrix}
\begin{bmatrix}
\cos(\theta) \\
\sin(\theta) \\
0
\end{bmatrix}
\begin{bmatrix}
X \\
Y \\
Z
\end{bmatrix}
\]  
\tag{3.14}

The point on the image when there is error ($\Delta\theta$) in yaw is given by

\[
\begin{bmatrix}
\hat{x} \\
\hat{y} \\
\hat{z}
\end{bmatrix} =
\begin{bmatrix}
f_x & 0 & c_x \\
0 & f_y & c_y \\
0 & 0 & 1
\end{bmatrix}
\begin{bmatrix}
\cos(\theta + \Delta\theta) \\
\sin(\theta + \Delta\theta) \\
0
\end{bmatrix}
\begin{bmatrix}
X \\
Y \\
Z
\end{bmatrix}
\]  
\tag{3.15}

\[
x^p = f_x \frac{X \cos(\theta + \Delta\theta) - Y \sin(\theta + \Delta\theta)}{Z} + c_x
\]  
\tag{3.16}
In the context of cross-calibration, the reprojection error for a 3D point is only a discrete quantity. So, we will analyze each source of error in this section.

Answering this question is important because any error in the extrinsic parameters will contribute to the reprojection error (for cross-calibration) unless the error is corrected. The mean of the reprojection errors is 0. For example, let’s say that we are considering 10 points and two 3D points have a reprojection error of 1 and the rest have 0. These errors manifest in the reprojection error during cross-calibration.

There are three sources of error in cross-calibration:

1. Error in the camera calibration parameters (K matrix)
2. Error in the translation components
3. Optimization error (reflects in the extrinsic parameters)

When there is a translation between the LIDAR frame and the camera frame, the projected point on the image is given by

$$\begin{bmatrix} \hat{x} \\ \hat{y} \\ \hat{z} \end{bmatrix} = \begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \end{bmatrix} + \begin{bmatrix} t_x \\ t_y \\ t_z \end{bmatrix}$$

The reprojection error is given by $|x^p - x^g|$. Making the assumption that $\theta = 0$, we get

$$x^p - x^g = \frac{f_x X \cos(\Delta \theta) - Z Y \sin(\Delta \theta) - f_x X}{Z}$$

(3.17)

In the above equation, $Z$ is both in the numerator and the denominator. So, reprojection error and the depth are not inversely related in this case. Figure 3.2 explains the situation.

### 3.3.2 Error in Translation Components

When there is a translation between the LIDAR frame and the camera frame, the projected point on the image is given by

$$x^g = \frac{f_x (X + t_x) + c_x (Z + t_z)}{Z + t_z}$$

(3.19)

When there is an error in translation, the projected point on the image is given by

$$\begin{bmatrix} \hat{x} \\ \hat{y} \\ \hat{z} \end{bmatrix} = \begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \end{bmatrix} + \begin{bmatrix} t_x + \Delta t_x \\ t_y + \Delta t_y \\ t_z + \Delta t_z \end{bmatrix}$$

(3.20)

$$x^p = \frac{f_y (X + t_x + \Delta t_x) + c_x (Z + t_z + \Delta t_z)}{Z + t_z + \Delta t_z}$$

(3.21)
(a) At 5 m, coloring errors barely noticeable (b) 7.5 m, coloring errors can be seen (c) 10 m, coloring errors increase

**Figure 3.3:** Coloring Error Analysis Setup. Errors can be Seen Along the Edges of the White Cardboard as the Depth Increases

As $Z$ grows, $Z + t_z + ∆t_z ≈ Z + t_z$. So

$$x^p ≈ \frac{f_x(X + t_x + ∆t_x) + c_x(Z + t_z)}{Z + t_z}$$

(3.22)

The reprojection error is given by

$$x^p - x^g = \frac{f_x ∆t_x}{Z + t_z}$$

(3.23)

The numerator is constant in the above equation and the denominator grows with depth. So as $Z$ increases, the error in coloring due to translation error reduces.

To summarise our analysis of coloring errors

- Errors in LIDAR points and the estimation of the camera matrix do not affect the coloring of a 3D point beyond particular depth.

- Errors in the estimation of the relative translation between the camera and the LIDAR do not affect the coloring of a 3D point beyond a particular depth.

- Errors in the estimation of the relative rotation between the camera and the LIDAR increases the coloring error with depth.
3.4 Coloring Error Analysis Setup

As discussed previously, coloring errors increase with depth if the estimation of the extrinsic (rotation) parameters are inaccurate. To evaluate this for our sensor suite, we created an experimental setup consisting of a white card board placed before a wet black cloth. The setup was placed at various depths and the laser scans of the setup were colored using the corresponding images. The setup was placed beyond 5 m in our tests so that the coloring error due to the error in LIDAR points and the error in camera matrix are completely ruled out. As derived previously, the depth at which these errors disappear are 4.304 m and 3.524 m for the color camera and the thermal camera respectively. Coloring errors were observed at the edges of the white card board. Any wrong coloring along the edges of the white card board is due to the inaccuracies in the estimation of the extrinsic parameters. Figure 3.3 shows the coloring of the setup placed at various depths. The coloring errors are noticeable at 5 m and it increases at 7.5 m – suggesting an error in the estimation of the rotation parameters. We measured the distance along the edges in the horizontal direction (x) and the vertical direction (y) that were mis-colored using Cloud Compare [26] (a 3D visualization tool) as shown in Figure 3.4. The errors we measured for the color and thermal cameras are reported in Table 3.1 and Table 3.2 respectively.

3.5 Comparison

Cross-calibration methods fall under two categories — error minimization methods and information maximization methods. Error minimization methods reduce the reprojection error when a LIDAR point is projected on to the corresponding image point (or) minimize the alignment error of planes defined in the camera frame and the LIDAR frame. Information maximization methods maximized the correlation
<table>
<thead>
<tr>
<th>Depth (in m)</th>
<th>Errors in cm</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Our Algorithm</td>
<td>Single Shot calibration</td>
<td></td>
</tr>
<tr>
<td></td>
<td>x</td>
<td>y</td>
<td>x</td>
</tr>
<tr>
<td>5</td>
<td>0</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>7.5</td>
<td>1</td>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>10</td>
<td>3</td>
<td>6</td>
<td>5</td>
</tr>
<tr>
<td>12</td>
<td>5</td>
<td>8</td>
<td>5</td>
</tr>
<tr>
<td>15</td>
<td>6</td>
<td>10</td>
<td>9</td>
</tr>
</tbody>
</table>

**Table 3.1:** Coloring Errors at Various Depths for the RGB Camera

<table>
<thead>
<tr>
<th>Depth (in m)</th>
<th>Errors in cm</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>x</td>
<td>y</td>
</tr>
<tr>
<td>5</td>
<td>2</td>
<td>4</td>
</tr>
<tr>
<td>7.5</td>
<td>3</td>
<td>6</td>
</tr>
<tr>
<td>10</td>
<td>3</td>
<td>7</td>
</tr>
<tr>
<td>12</td>
<td>4</td>
<td>10</td>
</tr>
<tr>
<td>15</td>
<td>5</td>
<td>13</td>
</tr>
</tbody>
</table>

**Table 3.2:** Coloring Errors at Various Depths for the Thermal Camera

**Figure 3.4:** Measuring the Colouring Error using Cloud Compare. Here the Coloring is Off by 10 cm as Seen in the Call-out
between the scene content observed in the different sensing modalities. We compare our method against two algorithms, one from each approach.

3.5.1 Error Minimization

We chose the single shot calibration toolbox [25] which minimizes the alignment error of planes defined in the camera frame and the LIDAR frame. As seen in Table 3.1, the coloring errors are marginally higher than our method, but the difference is not significant. However, the toolbox requires that multiple checkerboard patterns are placed in the scene and creating such targets for a thermal camera is a tedious task. Additionally, extracting all planes (containing the calibration targets) from the LIDAR point cloud reliably is a difficult task as there could be similar planes in the scene. Choosing the wrong set of planes will affect the final calibration results. We manually removed some planes in the scene (using Cloud Compare [26]) to get this toolbox to work and obtain the reported results.

3.5.2 Information Maximization

We compared the results of our algorithm to the method described in [57] which is based on maximizing mutual information between the intensity distributions of the LIDAR and the RGB images. The source code and a sample dataset are publicly available. Similar to their sample dataset, we collected 10 indoor scans and 10 outdoor scans using our setup. The extrinsic parameters were obtained and errors were compared using our error analysis setup. We observed a 20 cm error at 5 m and the errors increase hence forth.

We investigated the causes for the high errors and we realized that the mutual information method requires the intensity values of the LIDAR to be calibrated. The
authors in [57] use the calibration procedure described in [42] to calibrate the intensity values of their Velodyne LIDAR. This calibration method relies on the fact that the same surface patch is hit by different LIDAR beams (out of the 64 beams of Velodyne) and the average intensity of all those LIDAR returns represents the true surface albedo of that patch. A 64×256 lookup table is built corresponding to the 256 intensity levels for each beam. The intensity distribution of the LIDAR and camera data in their dataset (henceforth referred as MI dataset) is shown in Figure 3.5 and the corresponding distributions of our dataset is shown in Figure 3.6. It is hard to evaluate how similar the distributions are visually. However, it is safe to say that (i) the distributions in Figure 3.5 have a single peak and the distribution tapers down on either side of the peak (ii) the peaks are shifted. No such pattern is visible in Figure 3.6. To quantitatively compare the similarities of the distributions, we use the information gain metric and we observed that the information gain for distributions in Figure 3.5 is 4.82 which was higher (4.31) than the information gain for distributions in Figure 3.6.

Also, the intensity calibration method in [42] is specific for multi-beam LIDARs and is yet to be verified for a single beam LIDAR like the Hokuyo. Further, it is unclear on how the inaccuracies in the intensity calibration affects the cross-calibration algorithm.

Maximizing the mutual information between intensity distributions of the LIDAR and the thermal images (as implemented in [57]) also resulted in high errors similar to the RGB images discussed above. We hypothesize that this is because thermal cameras measure emitted radiation in the far-infrared (7µ to 14µ) whereas LIDAR intensities correspond to the reflectivity in the near infrared (∼1µ). Figure 3.7 shows
the intensity distributions of a scene in the LIDAR scan and the thermal image. In the LIDAR histogram multiple peaks are observable at 61 and between 200 and 255. However, in the thermal histogram, only two peaks are visible – one at 40 and the other at 255. It is obvious that the distributions look dissimilar and information maximization methods do not work in such cases. (However, in the case of color cameras the measured reflectivity in the visible spectrum (0.4µ – 0.7µ) is closer to the near infra red spectrum (0.7µ – 1µ) and the reflectivity values are assumed to have a strong correlation, making mutual information metric based on intensity distribution useful). Different mutual information metrics such as maximizing edge correlations could yield better results, but it is a case of further research.

![Intensity histograms](image)

(a) Intensity histogram of the LIDAR data   (b) Intensity histogram of the color images

**Figure 3.5:** Intensity Distributions of the MI dataset - the LIDAR Intensities are Calibrated

### 3.6 Conclusions

In this chapter, we analyzed the coloring error of a 3D points due to LIDAR measurement errors, errors in the camera matrix K, and errors in the extrinsic parameters (after cross-calibration). We analyzed the problem theoretically and came to the following conclusions

1. Coloring errors increase due to errors in relative rotation between the LIDAR
2. Coloring errors due to errors in LIDAR points, camera calibration errors, and relative translation between the LIDAR and camera decrease with depth and disappear after a particular depth.

We later introduced a setup that allows us to quantify the coloring errors after cross-calibration. We compared the results of our method to the existing methods and hypothesized why other methods failed during the cross-calibration of a thermal camera with a LIDAR.
INITIAL ALIGNMENT METHOD FOR POINT CLOUD REGISTRATION

In the last two decades, laser range scanners have been widely used in surveying and robot navigation applications. Consequently, there are a multitude of algorithms that build 3D maps from the point cloud data. An integral part of these algorithms is registration — an iterative optimization method that aligns two point clouds by estimating the relative transformation between them. Registration algorithms assume that an initial guess for the relative transformation is available. However, there are cases where one needs to compute the initial alignment. An example would be when the estimate of the initial guess is very noisy. Another example would be global registration where two existing 3D maps have to be aligned. In such cases, an algorithmic method to compute initial alignment is needed. This chapter presents a method for initial alignment for pairwise registration of point clouds. The method finds congruent pyramids in the two point clouds by using the invariant properties of a rigid body transformation: the ratio of lengths is preserved, the euclidean distance between points is preserved. Corresponding corners of the congruent pyramids are used to find a closed-form solution for initial alignment. The alignment is refined further using the Iterative Closest Point (ICP) algorithm. We validate the method on four datasets which include airborne LIDAR, terrestrial LIDAR, and indoor and present the results.

4.1 Related Work

Point Cloud registration has received a lot of attention in the last decade due to the ubiquitous use of 3D sensors. 3D sensors are used in a wide range of application
domains like mapping mines, underwater mapping, urban mapping, environmental monitoring etc. A fundamental requirement in these 3D sensing systems is to align consecutive scans to obtain a consistent representation of the environment. 3D sensors have resulted in abundant data and there has been an increasing demand to develop robust and efficient algorithms to register the data. This is evident from the fact that around 400 papers have been published in Robotics journals and conferences [60] in the last two decades.

![Diagram](image)

\[
\begin{align*}
\text{r1} &= \frac{|ae|}{|ab|} \\
\text{r2} &= \frac{|ce|}{|cd|} \\
\text{r1'} &= \frac{|a'e'|}{|a'b'|} \\
\text{r2'} &= \frac{|c'e'|}{|c'd'|} \\
\text{r1} &= \text{r1'} \text{ and } \text{r2} = \text{r2'} \text{ if } \{a,b,c,d,e\} \text{ and } \{a',b',c',d',e'\} \text{ are congruent}
\end{align*}
\]

**Figure 4.1:** Congruency Test for 4 Points, Based on the Affine Invariant Property that the Ratio of Lengths Does Not Change

Most registration methods are based on the Iterative Closest Point (ICP) algorithm [10], that iteratively reduces the root mean square (RMS) error between two point clouds (source and target). ICP is popular because of its speed and simplicity. In every iteration, ICP computes a closed-form solution for the transformation after computing the data association (i.e. correspondence) between points. Methods described in [8], [31] are commonly used to find the closed-form solution. The correspondence is computed using the nearest neighbour heuristic which is implemented efficiently using a kd-tree data structure. The process of computing the correspon-
The simplicity of ICP has resulted in its many variants, which are best summarized in [66]. Despite its popularity, ICP has convergence issues. ICP doesn’t converge in the absence of a good initial guess. An approximate correspondence between points is vital for ICP to converge. Nearest neighbour heuristic for correspondence fails when there is a big initial offset between point clouds, affecting ICP’s convergence. To illustrate this, let’s consider points on a straight line as the source and the target is the (same) line rotated by 90°. In this scenario there is one unique point in the source which is the closest (i.e the nearest neighbour) to all points in the target, which obviously is a wrong correspondence. The closed-form solution computed using such wrong correspondences causes ICP to diverge.

A comparison framework and open source library for ICP variants is presented in [60]. The authors acknowledge that the convergence of ICP depends on the initial pose and rely on inertial-measurement or odometry to obtain an initial guess. The authors propose sampled perturbations from the zero mean 6D multivariate Gaussian distribution to get a reasonable approximation of the initial guess. The following methods are suggested to ensure correct data association in [62], (i) attaching descriptors to remove disambiguation (ii) applying ICP algorithm fast enough to limit the magnitude of changes. However, descriptors vary with respect to data density – which makes it inappropriate for registering data sets of different point cloud densities. Applying ICP fast enough is suited for scan matching but not useful in global registration scenarios (typical examples of a global registration occur in terrestrial and airborne LIDAR datasets).

An alternate to ICP is the Normal Distribution Transform (NDT) [47]. NDT is a
surface matching method that approximates one point cloud (target) as a set of local probability density functions representing the local shape. NDT finds the transformation parameters that maximizes the likelihood that the points in the source scan lie on the target surface. NDT has a wider convergence basin than ICP \cite{icp} and is robust to sensor noise. Unlike ICP, NDT doesn’t have a closed-form solution. Numerical optimization methods like Gauss-Newton or Levenberg-Marquardt are used to solve the optimization problem. This makes the technique susceptible to local minima; hence using a good initial guess becomes important. One can use inputs from a odometer or an inertial-measurement unit to provide the initial pose. But, as mentioned earlier, this is not feasible in global registration scenarios; hence developing an algorithmic method for an initial guess becomes important.

A spectral method for registration is described in \cite{spectral}. Spectral methods for registration are relatively new to the Robotics community and are not studied as extensively as ICP or NDT. Hence, the role of initial guess is not well understood in spectral registration methods.

The problem of finding an initial guess is sparsely tackled in the Robotics community. Availability of odometers and inertial measurement sensors deems this problem less important. However, this problem cannot be ignored for applications like change detection, where registering two 3D maps is important to find the changes that have occurred in an environment. This is very typical of terrestrial scanning and airborne mapping applications.

One approach to initial alignment is based on finding similar features in the two point clouds \cite{feature}. Keypoints are extracted and matched using feature descriptors.
Matching keypoints are then used in a RANSAC framework to find a closed-form solution for initial alignment. The performance of this method relies on robust matching of keypoints. However keypoint correspondence by matching descriptors is not robust when there are differences in the point cloud densities.

Higher level features like planes, cylinders, and spheres were matched in [20, 16, 23, 61]. This approach can work only on highly structured environments and will fail in the absence of higher level features. Also, there is an additional cost of segmenting the scans. A similar method for robot mapping is presented in [21].

Another approach to initial alignment is to find congruent structures in the point clouds and compute a closed-form solution from the corresponding points of the congruent structures. A method to find congruent quadrilaterals is presented in [6]. The congruency test is based on the affine invariant property that the ratio of lengths is preserved. The method presented here resembles this congruency test. Another congruency technique is described in [84] where 3D SIFT keypoints and descriptors are used to identify $N$ congruent points. The congruency test is based on finding a similarity in the distribution of keypoints by using local descriptors and point pair relations. Local descriptors are not invariant to point cloud densities. Hence this approach can fail on datasets with varying densities. Moreover, congruency tests based on point pair relations are computationally expensive.

An initial alignment method based on overlapping cubes in the octree representations of the point clouds is presented in [54]. The initial alignment is computed by searching discrete points in the domain $[-T_{\text{max}}, T_{\text{max}}]$ (where $T$ is the transformation) that results in the best possible overlap of the octrees. This is similar to a brute force
search and the search space increases exponentially in the number of parameters in $T$.

Otherwise, the literature on initial alignment methods is limited. Point cloud registration softwares like CloudCompare [26] and Meshlab [18] allow users to select the corresponding points between point clouds to compute an initial alignment, which is then refined using ICP. A similar semi-automated approach that enables users to select corresponding regions in aerial datasets is presented in [39].

In this chapter, we tackle the problem of finding an initial alignment for point cloud registration algorithms. We present a method that is invariant to the magnitude of the initial offsets between the two point clouds. We approach the problem geometrically by finding congruent pyramids in the point clouds and use the congruent vertices of the pyramid to find a closed-form solution for initial alignment. We also explain our choice of the pyramid as the geometric structure. We validate our method on multiple datasets.

4.2 Approach

We define the terminology and assumptions before describing the approach to find congruent pyramids.

**Definition:** A pyramid is a structure with a polygonal base and an apex. We choose a quadrilateral as the polygonal base.

**Assumption:** The two point clouds are named as ‘source’ (with points $p_i$) and ‘target’ (with points $q_i$). They are related by a rigid body transformation (i.e. rotation ($R$) and a translation ($T$)). Let $\Phi_s = \{p_i\}, \Phi_t = \{q_i\}$
4.2.1 Workflow for Computing Initial Alignment

The workflow for computing the initial alignment is as follows

1. Initially a plane (called the MAX plane) is selected in the source — this is the base plane of the pyramid.

2. 4 points are randomly selected on this plane — these are the four corners of the base of the pyramid.

3. A point not in the MAX plane is selected — this forms the apex of the pyramid.

4. The congruent pyramid in the target (consisting of 5 points congruent to the points selected above) is computed by the method described in Sections 4.2.2 through 4.2.4

5. A closed-form solution for initial alignment is then computed from the corresponding corners of the congruent pyramids using SVD [8].

The initial alignment is further refined using ICP. Any fine registration method can be used, ICP is just our choice. The main contribution of this work lies in the workflow for computing the initial alignment. Fine registration is implemented for completeness.

The workflow is elaborated below in a non-linear fashion for convenience.

4.2.2 Finding a Congruent Base

The 4PCS algorithm [6] is used to find 4 congruent points comprising the quadrilateral base of the pyramid. Let a, b, c, d be the four corners of the quadrilateral base (in the source point cloud), and e be the intersection of the diagonal line segments
Since the ratio of lengths is preserved in a rigid body transformation the ratios $\frac{||ae||}{||ab||}$ and $\frac{||be||}{||bd||}$ remain unchanged. This property is used to find the congruent base $a', b', c', d'$ in the target point cloud. The test for congruency is explained in Figure 4.1.

Three points $a, b,$ and $c$ are randomly selected from the source points $p_i$, and a plane $P$ is fit on these 3 points. The set $\psi$ of points that lie on the plane $P$ is determined such that $|P . p_i| < \epsilon$. A point randomly chosen from $\psi$ acts as the fourth corner $d$ of the quadrilateral base $abcd$. The intersection of the line segments $ab$ and $cd$ is determined as $e$. The ratios $r_1 = \frac{||ae||}{||ab||}$ and $r_2 = \frac{||be||}{||bd||}$ are then computed. The objective is to find points $a', b', c', d'$ in the target such that the ratios $r_1$ and $r_2$ are similar. This is a test for congruency between $abcd$ and $a'b'c'd'$.

\textit{Note:} Since the ratios $r_1$ and $r_2$ are known, we can write

$$e = a + (b - a) \ r_1.$$ Also $$e = c + (d - c) \ r_2.$$ (4.1)

To compute congruent points in the target, the distance between every pair of points in $\Phi_i$ is computed. A list of point pairs $< a'_i b'_i >$ is computed such that $||a'_i b'_i|| - ||ab|| < \epsilon$. A similar list of point pairs $< c'_i d'_i >$ is computed such that $||c'_i d'_i|| - ||cd|| < \epsilon$. A set of intersection points $e^1_i$ is derived from $< a'_i b'_i >$ and the ratio $r_1$.

$$e^1_i = a'_i + (b'_i - a'_i) \ r_1.$$ (4.2)

Another set of intersections $e^2_i$ is derived from $< c'_i d'_i >$ and the ratio $r_2$.

$$e^2_i = c'_i + (d'_i - c'_i) \ r_2.$$ (4.3)
From Equation 4.1 it is clear that for every $|e_1^i - e_2^i| < \epsilon$ the corresponding point pairs $a'_ib'_ic'_id'_i$ are congruent to $abcd$. All $e_1^i$ are stored in a kdtree and each $e_2^i$ is used as a search query for computational efficiency. We observe that in Equation 4.2 swapping $a'_i$ and $b'_i$ would result in a different intersection $e_1^i$. Similarly swapping $c'_i$ and $d'_i$ in Equation 4.3 would result in different $e_2^i$. Hence another kdtree of intersections ($e_1^i$) is maintained, where $e_1^i$ are obtained after swapping $a'_i$ and $b'_i$. And the $e_2^i$ (obtained before and after swapping) are used as search queries in both kdtrees. This ensures that the congruency test is invariant to the order in which points are selected.

The closed-form solution for initial alignment is computed from the congruent points using SVD \[8\]. Let $T$ define the mapping between congruent quadrilaterals ($Q$) and their corresponding closed-form solution ($T$). So $T(Q) = T$ and $T^{-1}(T) = Q$. Multiple quadrilaterals $Q_k$ are chosen in the source and their corresponding congruent quadrilaterals $\{Q^c_k\}$ (note: there can be multiple congruent quadrilaterals) in the target are found for each $Q_k$. Choosing $k = 1$ is not appropriate as selecting only one quad can result in a false positive solution, i.e. the congruent quadrilaterals may not result in the right alignment. The closed-form solutions $\{T^c_k\}$ are derived for each congruent quadrilateral pair $Q_k \leftrightarrow \{Q^c_k\}$. The transformation with the least RMS error for each $Q_k$ is computed as

$$T_k = \arg\min_{T_k} \sum_i ||\hat{p}_i - T^c_k * \hat{q}_i||$$  \hspace{1cm} (4.4)

where $\hat{p}_i$ and $\hat{q}_i$ are closest point correspondences between the source and the target respectively. $\hat{Q}_k = T^{-1}(T_k)$ is the congruent quadrilateral that yields the least RMS error among all $Q^c_k$ for a particular $Q_k$.

\[1\]It is important to note that there can be multiple $a'_ib'_ic'_id'_i$ congruent to $a_ib_ic_id_i$
The initial alignment is then computed as

\[ T = \underset{T_k}{\text{argmin}} \sum_i \|\hat{p}_i - T_k \ast \hat{q}_i\| \]  \hspace{1cm} (4.5)

\( \hat{Q} = T^{-1}(T) \) is the congruent quadrilateral that yields the least RMS error among all \( \hat{Q}_k \). \( T \) is chosen as the solution for initial alignment.

The 4PCS algorithm fails in large aerial topographic datasets (spanning 4 sq km shown in Figure 4.2(a)). The reason is explained pictorially in 2D in Figure 4.3(a). This can be easily extended to 3D. There are multiple congruent quadrilaterals on the topography when an arbitrary plane is chosen. This is shown as matching line segments in 2D in Figure 4.3(a). This results in an increase in “c” in Equation 4.4. With an increasing “c”, Equation 4.4 becomes computationally expensive as \( p_i \leftrightarrow q_i \) correspondence evaluations are required for each “c”. Moreover, in partially overlapping datasets, the number of quadrilateral selections (\( Q_k \)) in the source should be increased to avoid false positive solutions that yield low RMS error. And increasing “k” contributes to computational burden in Equation 4.5.

(a) Aerial dataset with varying elevation differences spanning 4 sq km
(b) Aerial dataset with flat topography

Figure 4.2: Different Aerial Datasets
(a) Quadrilaterals on an arbitrary plane. There are multiple congruent quadrilaterals in the target dataset. If a plane that cuts across the data — in the top figure — if an arbitrary plane is chosen, there are more unique quadrilateral matches than an arbitrary plane chosen in the source; like the red plane shown in the bottom figure. Matching congruent quadrilaterals can be found in various planes in the target shown as multiple red planes.

**Figure 4.3:** Choosing Planes for Finding Congruent Quadrilaterals, Shown in 2D. The Topography is Drawn in Blue, Red Lines are Cross Section of Planes from Which the Quadrilaterals are Chosen.

To avoid false positive solutions and to overcome the computational burden, a constraint is imposed in the quadrilateral selection procedure as described below.

### 4.2.3 Selecting the MAX Plane

In the previous section, three points were selected at random and the fourth point was chosen in the plane containing the three points. In our proposed algorithm, we invert this procedure by identifying a plane initially, followed by selecting four points on this plane. The algorithm finds a plane that cuts through the entire dataset and selects quadrilaterals on this plane; we call this plane as the MAX plane. MAX plane is determined by a RANSAC procedure, where the plane containing the most
number of points in the point cloud is selected. The difference between selecting quadrilaterals on an arbitrary plane and selecting quadrilaterals on the MAX plane is explained pictorially in Figures 4.3(a) and 4.3(b).

Selecting quadrilaterals on the MAX plane ensures that the corners of quadrilaterals are spread across the entire dataset. This helps in filtering false positive matches in the target, thereby overcoming scenarios explained in Figure 4.3(a). This constraint significantly reduces “c” in Equation 4.4. Additionally, unique quadrilateral matches result in fewer quadrilateral selections $Q_k$ (reducing $k$ in Equation 4.5). Hence this constraint serves two purposes (i) to reduce false positive solutions (ii) to reduce computational burden.

Figure 4.4: Multiple Congruent Quadrilaterals on a Planar Surface

While this approach works on topographies with varying elevation differences, it fails on datasets with ‘flat’ topographies, i.e. without much elevation differences (the mathematical definition of ‘flat’ topography is described in the next section). One such dataset is shown in Figure 4.2(b). On ‘flat’ regions, the MAX plane is the ground plane and there are multiple congruent quadrilaterals on this plane (shown in Figure 4.4), similar to the problem mentioned in Section 4.2.2. We overcome this problem by searching for congruent structures in 3D instead of congruent structures in a 2D plane.
Let $\Psi_p$ be the set of points on the MAX plane. The key to find congruent structures in ‘flat’ datasets are the few points in the set $\Psi_{np} = \Phi_s \setminus \Psi_p$ that are not on the MAX plane. Our approach is to use $m_i \in \Psi_{np}$ as an additional constraint for congruency, thus moving from 2D quadrilaterals on a plane to 3D pyramids. We choose the quadrilateral base of the pyramid from $\Psi_p$ and the apex of the pyramid from $\Psi_{np}$. Though this solution was adopted to register ‘flat’ regions, it is still generic and can be applied to any indoor / outdoor dataset (as shown in Section 5.7).

### 4.2.4 Finding a Congruent Pyramid

The motivation for searching congruent pyramids is to exploit $m_i \in \Psi_{np}$ to reduce the search space of congruent structures, and to enhance the generality of the solution. As mentioned in Section 5.3, the pyramid we consider consists of a quadrilateral base and an apex. This allows us to borrow the method described in Section 4.2.2 to find the congruent base of the pyramid.

A pyramid $\Delta = \{a, b, c, d, f\}$ is chosen such that $\{a, b, c, d\} \in \Psi_p$, $f \in \Psi_{np}$. The pyramid consists of the base plane $\Delta_b$ and 4 base-apex planes (passing through an edge of the base and the apex) $\Delta_{ba}^1, \Delta_{ba}^2, \Delta_{ba}^3, \Delta_{ba}^4$. The apex $f$ is the intersection of the planes $\Delta_{ba}^1, \Delta_{ba}^2, \Delta_{ba}^3, \Delta_{ba}^4$. Since 3 planes intersect at a point, it is possible to find the apex $f$ with any three base-apex planes.

The equations of base-apex planes are found using the two edge points from the base and the apex. Since the congruent quadrilateral between the source and the target is already known, the transformation $T$ between them is computed from corresponding corners using SVD. Now $T$ is applied to the base planes to derive new base-
apex planes $\hat{\Delta}_{ba}^1, \hat{\Delta}_{ba}^2, \hat{\Delta}_{ba}^3, \hat{\Delta}_{ba}^4$ in the target. The apex of the new pyramid $f''$ is the intersection of any of the three base-apex planes $\hat{\Delta}_{ba}^1, \hat{\Delta}_{ba}^2, \hat{\Delta}_{ba}^3, \hat{\Delta}_{ba}^4$. Since euclidean distances are preserved in a rigid body transformation, an $f' \in \Phi_t \ s.t \ |f' - f''| < \epsilon$ completes the congruent pyramid $\hat{\Delta}$ to the source pyramid $\Delta$. The entire method is described in Figure 4.5.

When $f' \ s.t \ |f' - f''| < \epsilon = \emptyset$, a congruent pyramid does not exist, thus discarding false positive quadrilaterals. This extension is particularly useful in datasets like Figure 4.2(b) where considering the very few points above the ground plane helps discard a lot of false positives and improve the mechanism to find congruent structures.

It is trivial to notice that finding congruent pyramids to compute initial alignment is invariant to the magnitude of the initial offset.

### 4.2.5 Flatness Measure

We derive a metric to define flatness of a 3D dataset. An intuitive measure of flatness is the deviation of points from a planar (flat) surface. This intuition is modeled as an entropy measure. To reiterate, $\Psi_p$ are points on the $MAX$ plane and $\Psi_{np}$ are points not on the $MAX$ plane. The perpendicular distances $d_i$ of points $m_i \in \Psi_{np}$ to the $MAX$ plane is computed. A histogram of distances $H$ is derived from $d_i$ with a bin interval $\delta$. $H$ is normalized by $N = |\Psi_{np}|$ yielding a probability distribution of perpendicular distances $P$. The entropy of $P$ defines spread of the distribution which in turn defines the deviation of $m_i \in \Psi_{np}$ from the $MAX$ plane.

To interpret the flatness measure numerically, let us consider a perfectly flat surface. In this case, the flat surface is the $MAX$ plane. All perpendicular distances

67
are 0 and fall under the same bin in the histogram $H$. Thus the probability mass function $P$ has probability 1 for one bin and 0 for all other bins. It is easy to see that the entropy of this distribution is 0. (The definition of entropy is $\sum_i -p_i \log p_i$). So a flatness measure close to 0 is an indication of a flat surface. The entropy measures of the datasets shown in Figures 4.2(a), 4.2(b) are 3.735 and 0.619 respectively. One can notice that the dataset in Figure 4.2(b) is flat for the most part and its flatness measure is 0.619 (which is much lesser than the dataset in Figure 4.2(a)).

![Figure 4.5: Finding Congruent Pyramids](image)

**4.2.6 Change in Flatness Measure Due to Sensor Noise**

In this section, we argue that the sensor noise doesn’t have a major impact on the flatness measure. The flatness measure changes only by an $\epsilon$ which doesn’t change our understanding of whether a surface is flat or not. In the previous section, flatness measure is defined as the entropy of probability distribution $D$ of perpendicular distances from $MAX$ plane. Let $\hat{D}$ be the probability distribution of perpendicular distances after adding noise. We argue that $D \approx \hat{D}$, hence the flatness measure has only an $\epsilon$ change with sensor noise.
Let $\delta_{bw}$ be the bin width, and $d$ be the perpendicular distance considered. The bin number for $d$ is given by $B = d/\delta_{bw}$, and the residue $R = (d \mod \delta_{bw})$. For a given point cloud, let $\Gamma$ be the probability mass function of residues and

$$
\sum_{i=0}^{\delta_{bw}} P(R = i) = 1 \tag{4.6}
$$

On adding noise with a distribution $\mathcal{N}$, the residue distribution becomes $\mathcal{N} \ast \Gamma$ and

$$
\int \mathcal{N} \ast \Gamma = 1 \tag{4.7}
$$

Let $n_d$ be the sensor noise sampled from $\mathcal{N}$. A bin switch occurs when

$$
(R + n_d) > \delta_{bw} \quad \text{or} \quad (R + n_d) < 0. \tag{4.8}
$$

The noise distribution $\mathcal{N}$ causes bin switches (Equation 4.8) and alters the residue distribution $\Gamma$ and its entropy $H_{\Gamma}$. The resulting residue distribution is given by $\mathcal{N} \ast \Gamma$ and its entropy is $H_{\mathcal{N} \ast \Gamma}$. For a particular residue distribution $\Gamma$, we run simulations of different noise distributions $\mathcal{N}_i$ and compute $H_{\mathcal{N}_i \ast \Gamma}$. The difference in entropy $\Delta H_i$ is given by $H_{\Gamma} - H_{\mathcal{N}_i \ast \Gamma}$. This procedure is repeated for different residue distributions $\Gamma$ in a monte-carlo simulation, assuming that all residue distributions are equally likely. The expected value of the entropy change is

$$
H_{\epsilon} = E[\Delta H_i]
$$

We modelled the noise as a mixture of Gaussians. The residue and noise distribution for various simulations and their corresponding $\Delta H_i$ values are shown in Figure 4.6. The $H_{\epsilon}$ we obtained after all simulations is -0.1344.
Figure 4.6: The Top Row Shows the Various Residue Distributions Considered (Only 4 is shown for Brevity. 1000 Distributions were Considered in Total). The Bottom Row Shows the Different Noise Distributions Considered (4 in Total). The Noise is Modelled as a Gaussian Mixture. Each Residue Distribution is Convolved with Every Noise Distribution and the Difference in Flatness Measure is Computed. The Difference in Flatness Measure $\Delta H_i$ is Shown in the Last Row. One Can Notice that $\Delta H_i$ is not Significant to Mis-interpret a Flat Surface Otherwise

4.3 Sufficiency Proofs

In this section we present lemmas to support our choice of (i) a pyramid as the 3D structure for our congruency test and (ii) a quadrilateral base for the pyramid congruency test.

*Lemma*: Pyramid congruency test is a sufficient test for 3D congruency. All other 3D structures subsume the pyramid.

*Proof*: A 3D structure consists of 3 or more points on a plane and atleast 1 point not on the plane. Let $\Phi_p$ represent points on the plane and $\Phi_{np}$ represent points
not on the plane. A pyramid base $\Delta_b$ can be formed from $p_i \in \Phi_p$ for any $i \geq 4$. When $|\Phi_{np}| = 1$, $q \in \Phi_{np}$ forms the unique apex to the pyramid base $\Delta_b$. When $|\Phi_{np}| > 1$, every $q_i \in \Phi_{np}$ can be used as an apex to the pyramid base $\Delta_b$ resulting in $|\Phi_{np}|$ pyramids $\Delta^k$. Thus a 3D congruency test on $\Phi_p \cup \Phi_{np}$ can be broken down into multiple pyramid congruency tests on $\Delta^i_b \times \Phi_{np}$, where $\Delta^i_j$ are pyramid bases that can be formed in $\Phi_p$ consisting of 3 or more points. Hence a pyramid congruency test is a sufficient test for 3D congruency.

Lemma: In a pyramid congruency test, using a quadrilateral base is sufficient. Any other polygonal base $\Delta_p$, where $|\Delta_p| \geq 4$ subsumes a quadrilateral base.

Proof: Let $n = |\Delta_p|$. Thus $\binom{n}{4}$ quadrilaterals can be formed from $n$ points, resulting in $\binom{n}{4}$ pyramids with quadrilateral bases $\Delta_q$. Hence a pyramid base $\Delta_p$ with $n = 4$ is sufficient.

4.4 Results

To validate the generality of this method, we choose datasets that represent rural landscapes, an urban setting with buildings and trees, and a garage with shelves and tools. The 4PCS algorithm failed on all these datasets. We show results on 4 datasets — 2 airborne LIDAR, one terrestrial LIDAR, and one indoor. We also show one dataset where our method fails.

Congruency tests are run on keypoints computed on the source and the target, instead of the entire dataset. Using keypoints is a way of sampling the data, to ensure that the congruency tests are fast. To reduce the time in computing keypoints, 100,000 points are sampled in the source and the target and keypoints are computed on this downsampled dataset. Any keypoint can be used, as the method depends on finding congruent pyramids from keypoints and not on the local properties of the
keypoints. We used ISS keypoints \[87\]. For fine registration, ICP with a point to plane metric and linear least squares approximation \[45\] was used. We used implementations from Point Cloud library (PCL) \[68\] for ISS keypoints and ICP.
Initially we present results on the airborne LIDAR data shown in Figures 4.2(a) and 4.2(b). This dataset of Figure 4.2(a) contains 4.7 million points spanning 4 sq kms. To simulate different data collection runs, the data is clipped to obtain the source and the target with partial overlap. The source covers ~ 1.2 sq km and the target covers ~ 1.5 sq km. The partial overlap ratio is 0.65. The target is then transformed with a large initial offset and gaussian noise is added to simulate sensor noise. The registration results are shown in Figure 4.7. The variation of RMS error w.r.t the gaussian noise is given in Table 4.1.

Since it is common to have varying densities across data collection runs, we evaluated the performance of this method with varying densities between the source and
Table 4.3: 4 sq km Dataset

<table>
<thead>
<tr>
<th>% of points removed</th>
<th>RMS</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>0.4078</td>
</tr>
<tr>
<td>20</td>
<td>0.4077</td>
</tr>
<tr>
<td>30</td>
<td>0.4070</td>
</tr>
<tr>
<td>40</td>
<td>0.4065</td>
</tr>
</tbody>
</table>

Table 4.4: RMS Error for Varying Gaussian Noise

<table>
<thead>
<tr>
<th>% of points removed</th>
<th>RMS</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>0.0821</td>
</tr>
<tr>
<td>20</td>
<td>0.1752</td>
</tr>
<tr>
<td>30</td>
<td>0.2560</td>
</tr>
<tr>
<td>40</td>
<td>0.3689</td>
</tr>
</tbody>
</table>

the target. This scenario is simulated by removing points in the source and registering with the target. The RMS error after registration for varying densities is shown in Table 4.3.

Later we evaluated this method on the flat topography shown in Figure 4.2(b). This dataset contains 2.6 million points spanning 0.85 sq km. For this dataset, the data is split to get subsuming data instead of a partial overlap. The source spans \( \sim 0.12 \) sq km and the target covers \( \sim 0.24 \) sq km. Subsuming datasets are more challenging for flat topographies as there are multiple ways in which two planar surfaces can fit. Hence, the points that are not on the MAX plane are critical in finding the right alignment. The results are shown in Figure 4.8. One can notice that the topography is flat, except for a few shrubs on the surface. The points on these shrubs are not on the MAX plane and form the apexes of pyramids, which are crucial for
congruency tests. Again, the variation of RMS error w.r.t the gaussian noise is given in Table 4.2. Here the results are bad beyond $\sigma = 1$. Our hypothesis is that the points with high noise dominate the actual signal (points on the shrubs) resulting in ambiguous matches for the apex of the pyramids. We then proceeded to test with varying data densities. The RMS error after registration for varying densities is shown in Table 4.4.

Later, we tested our method on an urban dataset \cite{2}. Figure 4.9(a) shows a closeup view of this dataset where structured buildings and unstructured trees can be seen. This data contains 500,000 points and spans $\sim 120m \times 330m$. The result is shown in Figure 4.9.
Finally, we tested on an indoor dataset — LIDAR data of a garage [1]. This dataset contains 3 million points. The registration result is shown in Figure 4.10.

One dataset where our method failed is the LIDAR data containing dense vegetation [61]. An image obtained during data collection is shown in Figure 4.11(a). The point cloud is shown in Figure 4.11(b). Finding a unique MAX plane is difficult in such datasets and consequently the search for uniquely congruent structures fail. Adding more constraints to reduce the search space for congruent structures is a possible future direction.

One target application for robust global registration for aerial mapping is to detect
changes in landscape that have occurred over a period of time. One such work is presented in [38], where changes in earthquake zones before and after an earthquake are detected. The idea is to use structure from motion techniques to build 3D models of the terrain from images collected using our helicopter platform. 3D models across time are then compared to detect changes in the landscape.

4.5 Conclusions

In this chapter, we presented an initial alignment method to register two point clouds. The method is based on the properties of a rigid body transformation. We showed the performance of this method on airborne LIDAR, terrestrial and indoor datasets. Results show that the method can be applied to various indoor and outdoor datasets. We then presented a discussion on the sufficiency test for 3D congruency. Additionally we define a metric to quantify the ‘flatness’ of a surface. The variation of the flatness measure with respect to sensor noise is discussed with experimental results. We conclude by mentioning the future research directions.
This chapter presents a new application domain for robot mapping — detecting changes caused by an earthquake using Airborne LIDAR data. The recent explosion in sub-meter resolution airborne LIDAR data raises the possibility of mapping detailed changes to Earth's topography. We present a new method that determines three-dimensional (3D) coseismic surface displacements and rotations from differencing pre- and post-earthquake airborne LIDAR point clouds using the Iterative Closest Point (ICP) algorithm. Tested on simulated earthquake displacements added to real LIDAR data along the San Andreas Fault, the method reproduces the input deformation for a grid size of $\sim 50$ m with horizontal and vertical accuracies of $\sim 20$ cm and $\sim 4$ cm, values that mimic errors in the original spot height measurements. Later, we applied this technique to the real earthquake datasets: El Mayur Cucupa Earthquake, Mexico (2010) and Fukushima earthquake, Japan (2011), and present the results.

5.1 Problem Statement

The problem can be formulated as follows. Given pre- and post-earthquake LIDAR point clouds (each containing a scattered distribution of points), find the 3-dimensional displacement (with rotation and translation components) that has best shifted the post-earthquake point cloud from its pre-earthquake equivalent. These shifts will vary spatially, depending on the distance to the fault, the sense and magnitude of slip and secondary effects such as landsliding. For this reason, the area must be divided into separate windows and the best local transformation identified for each one. To complicate matters, post-event windows which contain surface faulting will
not be related by a rigid body transformation to their pre-event equivalents.

A few things must be considered in this problem statement. Firstly, how do we decide upon an appropriate window size for splitting the data? Secondly, without any prior knowledge, how do we identify whether a particular window contains the fault, or lies away from the fault and has been shifted?

5.2 Data Description

We began our experiments using a synthetic earthquake dataset, before moving on to real earthquake displacements. The synthetic post-earthquake dataset was generated by adding displacements of known magnitude and sense to a real point cloud (the ‘target cloud’), to be tested against another, unaltered point cloud representing the pre-earthquake ground surface (the ‘source cloud’). This way, we were able to identify an approach which best reproduced the known input displacements. We used publicly available “B4” LIDAR data [11] covering a ∼2 × 2 km section of the San Andreas Fault (SAF) near Coachella, CA. Our synthetic fault strikes North-West through the center of the target cloud, close to the real surface trace of the SAF. To simulate a vertical, right-lateral rupture, we displaced points North-East of the fault 2 m towards the South-East, and displaced points South-West of the fault 2 m towards the North-West. To evaluate our ability to detect vertical motions, we also raised points on the North-East of the fault by 1 m. After investigating the synthetic case, we go on to test the method using real pre- and post-earthquake data from part of El Mayor-Cucapah earthquake rupture in Mexico [55] and Fukushima earthquake in Japan [44].
5.3 Approach

Initially, we chose an arbitrary window size (100 × 100 m). For each of these windows in the source (pre-earthquake) cloud, the corresponding window in the target (post-earthquake) cloud is identified based on x and y coordinates. This target window is then enlarged (e.g. by 10%) such that the displacements that we are trying to quantify are fully accommodated. Next, we computed the rigid body transformation between the source and target windows using the ICP algorithm. ICP operates by finding the corresponding point \( q_i \) in the target cloud for every point \( p_i \) in the source cloud, and determines the rigid body transformation that minimizes the distances between these points. It is an iterative process where the correspondences and the errors are computed at every iteration and the rigid body transformation is applied.
to the source cloud repeatedly until it aligns with the target cloud. With the point to plane error metric, the objective is to minimize the distance between the source point \((p_i)\) and the tangent plane at the corresponding target point \((q_i)\). The error metric can be written as follows

\[
E = \sum_i \| (\phi p_i - q_i) \cdot n_i \|^2
\]  

(5.1)

where \(\phi\) is the rigid body transformation that minimizes the error metric and \(n_i\) is the normal to the tangent plane at \(q_i\). The transformation matrix consists of a translation component and a rotation component. \(\phi = T(t_x, t_y, t_z) \cdot R(\alpha, \beta, \gamma)\). A linear approximation can be made to the rotation matrix where \(\theta \approx 0\) and the new transformation matrix is of the form below.

\[
\phi = \begin{pmatrix}
1 & -\gamma & \beta & t_x \\
\gamma & 1 & -\alpha & t_y \\
-\beta & \alpha & 1 & t_z \\
0 & 0 & 0 & 1
\end{pmatrix}
\]

(5.2)

5.4 Finding the Right Window Size

Finding the correct window size to compute the rigid body transformation is an important factor in recovering the ground truth displacements. We began evaluating the effect of window sizes in recovering the ground truth displacements on the synthetic earthquake dataset. The input displacements on either side of the synthetic fault were \([1.41 \ 1.41 \ 1]^T\) and \([-1.41 \ -1.41 \ 0]^T\) respectively. Next, we computed the surface displacements by varying the widow sizes from 100 m to 15 m. The standard deviation of the output displacements computed on all windows are plotted in Figure 5.1. Clearly, the standard deviation increases as the window size is reduced, and the mean of the surface displacements also shifts from the input displacements.
To identify the right window size for a dataset, we began by choosing an arbitrary window size in the source cloud (e.g. 200 m $\times$ 200 m). This window is split into four smaller windows of equal size and the rigid body transformation is computed on every child window. The transformation is validated after each split (explained in section 5.5) and the associated error computed. Based on the differences in error after consecutive splits, we decide whether further splitting is necessary. We verified experimentally that we cannot have small errors for very small window sizes ($\sim$10 m) given the point cloud densities and input displacements. An analysis of this error indicates when to stop splitting.

5.5 Transformation Validation

We validate the transformations by randomly choosing $N$ points per iteration in the transformed source window ($\phi p_i$) and finding the closest point in the target win-
The error for the $k^{th}$ iteration is computed as $E_k = \sum_i \| \phi p_i - q_i \|^2$ and the standard deviation of this error is calculated over $k$ iterations. For a good alignment the standard deviation should be minimal. Figure 5.2 shows the standard deviation of errors for different window sizes. It can be seen that the standard deviation increases gradually as the windows become smaller (part a of the figure shows plots for window sizes of 75 m, 50 m and 25 m). However, at a particular point (for our data, a window size of 10 m) the standard deviation jumps markedly, as shown in part b (note the difference in $y$-axis scales between a and b). If this happens, it is because the computed transformation for that window is wrong. To discard these invalid transformations, we use a thresholding based on the change in standard deviation as a stopping criteria for window splitting (whereby the standard deviation should not exceed $1/m$ times that of the previous step).

5.6 Fault Analysis

Here, we are interested in filtering the windows containing the surface rupture (or the fault). After running ICP using a good window size, each window is then considered for a fault analysis. The curvature of the local surface is computed at every point in the transformed source windows (obtained by applying the computed transformation on the source window i.e. $\phi p_i$) and target windows ($q_i$) and the curvature distribution is estimated by assigning the curvature computed at each point to different bins of an histogram (ranging from $\text{max-curvature}$ to $\text{min-curvature}$) and then computing the probability mass function from this histogram. If there is no rigid body transformation (in case of windows containing the fault) the source and target curvature distributions will not be the same. An information theoretic measure is used to detect this inconsistency in the curvature distributions. The information gain
between the transformed source cloud \(X\) and the target cloud \(Y\) is given by

\[
I(X; Y) = H(X) + H(Y) - H(X,Y)
\]

where \(H\) is the entropy of the curvature distribution. \(H(X,Y)\) is computed on the curvature distribution of the merged clouds \(X\) and \(Y\). When the right window size is used on regions related by a rigid body transformation, the information gain should be maximum. If the estimated transformation is sub-optimal (i.e. if ICP converges to a local minima) or if the considered region is not related by a rigid body transformation (in the case of windows containing faulting) the information gain should be minimal. Hence thresholding based on information gain highlights which windows contain the fault, along with a few false positives where ICP results may be different from the ground truth. It is important to choose the right window size. If a window containing the fault is too large, then points lying away from the fault will dominate the curvature distribution and the fault detection mechanism will be affected.

5.7 Results

Figure 5.3 shows a simple height differencing of the raw Mexico earthquake data, with clear positive height changes West of the fault and negative changes East of the fault. After a global registration, these height differences are reduced with similar height changes on both sides, as seen by the red shading in Figure 5.3b. This is because ICP has minimized the least square error over the entire point cloud, including both those regions that contain the fault and those that are displaced. The alignment occurring as a result of this least square minimization is not sensitive to the local displacements that we are trying to quantify, and hence a global registration is not suitable for this problem.
Figure 5.4(a) shows the data split up into multiple, randomly coloured windows with the thick black line showing the synthetic fault line, either side of which artificial displacements were added (as described in section 5.2).

Figures 5.5(a) and 5.5(b) show the displacement vectors (~2m in length) obtained for different window sizes for the synthetic earthquake dataset. The change in the direction of the displacement vectors either side of the fault (shown by the red line) are obvious. However, the displacement vectors for windows along the fault are inconsistent. These are windows that are not related by a rigid body transformation and ICP finds the transformation that minimizes the least squares error. Reducing window sizes beyond this point did not satisfy our transformation validation criteria and hence further splitting of windows was stopped.

Figure 5.4(b) shows the results of our fault detection method, which filters out windows based on information gain as described in section 5.3. Compared to figure 5.4(a), only those windows which fall below the information gain threshold are now shown, including a North-West trending sequence of windows along the fault. In addition, there are a few false positives, mostly along the edges where window splitting has left few data points in one of the datasets. We hypothesise that ICP converges to a local minima in these windows.

Figure 5.6 shows the displacements calculated for the synthetic earthquake overlaid on the actual topography (we used a DEM derived from publicly available “B4” LIDAR data). Black arrows are horizontal displacements and coloured circles denote vertical displacements. The differences in these displacements are clear on either side of the fault.
Finally results on the real earthquake datasets can be seen in Figures 5.7 and 5.8. Figure 5.7 shows results for the El Mayur Cucupa earthquake, Mexico and Figure 5.8 shows results for the Fukuhisma earthquake, Japan.

**Figure 5.3:** (a) Height Difference Map of the Mexico Earthquake, Before Global ICP, with $x$ and $y$ Coordinates in Meters. Height Changes Across the Fault are Clear. (b) Height difference Map After Global ICP, with Height Differences Reduced.
(a) Top view of data split into multiple windows, the thick line shows the line along which the fault was defined. (b) Top view of windows containing the fault, there are a few false positives - these are places where ICP converges to a locally optimal solution.

**Figure 5.4:** Window Split and Fault Detection

---

(a) Window size 200 m  
(b) Window size 100 m

**Figure 5.5:** Displacement Vectors for Different Window Sizes. The Approximate Length of the Displacement Vectors is 2 m. Notice the Change in Vector Directions on Both Sides of the Fault. X and Y Values are in Meters.
Figure 5.6: Results for the Simulated Earthquake. Horizontal Displacements (Black Arrows) and Vertical Displacements (Coloured Circles) Can Clearly be Seen to Change Markedly Either Side of the Fault. X and Y Axes Show UTM Zone 11 Coordinates, in Meters.
Figure 5.7: Results for the Mexico Earthquake. The Thin Lines Show the Earthquake Surface Faulting, as Observed by Geologists, with E-facing Scarps in Green and W-facing Scarps in Blue. Again, the Horizontal and Vertical Displacements Clearly Change Markedly Across the Fault. X and Y Axes Show UTM Zone 11 Coordinates, in Meters.
**Figure 5.8:** Results for the Japan Earthquake. $X$ and $Y$ Axes Show UTM Zone 11 Coordinates, in Meters.
Advances in Robot Perception in the past two decades have enabled many real world robotic applications. Solving some fundamental problems in Robot Perception such as mapping, localization, calibration, and sensor fusion has been the key for such advancements. This thesis presented solutions to two such fundamental problems in Robot Perception: Cross-Calibration of a multi-sensor system and Global Registration of two 3D maps. Solving these problems enables a wide range of real world applications – change detection being one of them. This thesis introduced one application of Change Detection, where the pre- and post- earthquake maps of a place are used to identify the surface displacements caused by an earthquake.

6.1 Summary of Contributions and Future Work

In the first part of this thesis we tackled the problem of creating RGB-Depth-Thermal data from a sensor suite consisting of an RGB camera, a thermal camera, and a LIDAR. The contributions were i) Producing a calibration target for the thermal camera that is easy to make ii) A unified setup for cross-calibrating both RGB and Thermal cameras with a LIDAR iii) Identifying various sources of coloring error and their contributions towards the coloring of a 3D point iv) Presenting an error-analysis setup to quantify the coloring errors v) Benchmarking our work against two benchmark papers in cross-calibration using our error-analysis setup.

The limitation of the described method is that it runs offline and requires user
involvement for calibration. This approach is not desirable for repeat calibration scenarios – especially for systems that are deployed frequently. It is definitely useful to have an algorithm that works online as well. As future work, it is possible to extend the same approach for online calibration. The crux of our algorithm depends on aligning the edges observed in the cameras and the LIDAR. By using circles of a known radius, we simplified the edge detection problem. It is easy to see that the same algorithm can be applied to online calibration if the circle edges are replaced with edges computed on the entire scene. However, the objective function (described in Section 2.3.1, Figure 2.8) will look very different and it might not result in a global minima. To overcome that, we could use circle edges to get the initial estimate of the extrinsics and progressively refine these initial estimates for online calibration by using the edges computed on the entire scene. This is particularly useful on systems that are deployed on a regular basis where constant vibrations in the system can result in the extrinsic parameters to wiggle over time. Having a system that can identify and correct these wiggles is practical.

Using a calibration target made of black and white melamine has its limitations as well. We assume that the weather is warm to allow the calibration target to heat up and produce the necessary thermal contrast. This restricts this approach to work only under warm weather conditions. Additional ways such as using a heat gun can be considered to warm the calibration target. Or, a different calibration target that works in all-weather conditions can be considered.

It is also interesting to find application domains for RGB-Depth-Thermal mapping. An obvious application is to identify the changes in the temperature profile of a building over time, that can give clues about the stress levels at different structures.
in the building. In the context of Autonomous Driving, it would be interesting to see if the place signatures derived using the thermal data would help place recognition / localization at night and visually challenging environments (because of Perceptual Aliasing). A radiometrically calibrated thermal camera can also be used for person classification (based on body temperature) and these can be used in the context of autonomous driving as well.

Thermal-LIDAR mapping provides a new dimension to 3D mapping. However, thermal cameras are not without limitations. Infrared energy does not travel as far in heavy atmospheric conditions. So in case of heavy fog or rainy conditions, thermal cameras cannot see very far. For e.g., the detection range of a human being for the FLIR Tau 2 thermal camera is ∼50 m and in the case of heavy fog or rain, this distance reduces. Thus, heavy atmospheric conditions limit thermal-LIDAR mapping and its application to urban driving at night is limited.

The second part of the thesis tackled the Global Registration problem, i.e., aligning two 3D maps when the initial estimate of the transformation between them is unknown. In the absence of initial estimates, fine alignment methods converge to a local minima. Since we known that a closed-form exact solution exists when four correspondences are known, we tackle the global registration problem by finding these ‘four’ correspondences. Finding the right correspondences is a combinatorial optimization problem as there are nP4 possible correspondences for n points in the target point cloud. We presented a method that makes this combinatorial search problem quadratic in the number of points. The method finds these corresponding points by searching for congruent structures in the two point clouds. The congruent structure used here is a pyramid consisting of a quadrilateral base and an apex. This method
uses the properties of a rigid body transformation that the ratio of lengths do not change. Using this approach, we were able to successfully register aerial maps of topographies which look largely flat with very little features to match between them. We presented the results of this algorithm on aerial as well as urban and indoor datasets.

The pyramid search algorithm we proposed is quadratic in the number of points. This is still a big number as the number of points increase. It would be interesting to extract higher level features like ‘planes’ and do a combinatorial search on matching these higher level constructs. Thus ‘n’ would refer to the number of higher level constructs and not points - which could significantly cut down the search space.

Change Detection is a very important application domain for 3D mapping. Producing 3D maps of an environment repeatedly and asking the question ‘What has changed in this environment is the last 3 days?’ is meaningful. We presented a Change Detection application specific to earthquake scenarios. We computed the vector fields caused by the earthquakes and the magnitude of those vectors correspond to the stress levels on the earthquake fault. We presented the results of our approach on a simulated earthquake to begin with and later presented results on two real earthquake datasets: El Mayur Cucupa earthquake (in Mexico, 2010) and the Fukushima earthquake (in Japan, 2011). An interesting research direction to pursue would be computing the subtle surface movements along the fault before the earthquake happens and use it to measure the stress along the fault. This can potentially give a clue on when the surface rupture would happen in the future.

There are many more applications for Change Detection. For instance, many
questions related to work at a construction site can be answered from 3D maps (acquired by drones) over time. Some examples would be: What is the volume of the rubble that was created by demolishing this building? What is the volume of construction that happened in the last 2 weeks? How many trucks of soil are needed to fill this hole at this place? All these questions require a volumetric understanding of the environment and the changes that have happened over time. And there are many more exciting applications to be discovered.
REFERENCES

[1] Point Cloud Library Data Repository — 3DVSL dataset

[2] Point Cloud Library Data Repository — ISPRS dataset


[12] Dorit Borrmann, Hassan Afzal, Jan Elseberg, and Andreas Nüchter. Mutual


[23] Marek Franaszek, Geraldine S Cheok, and Christoph Witzgall. Fast automatic


[68] Radu Bogdan Rusu and Steve Cousins. 3d is here: Point cloud library (pcl). In ICRA, 2011.

[69] Davide Scaramuzza, Ahad Harati, and Roland Siegwart. Extrinsic self calibration


