Efficient Object Localization and Scene Registration Using Contour-Enhanced Point Cloud Resampling Techniques

Contents

List of Figures and Tables

Chapter 1

Introduction

1.1 The Object Localization Problem

One of the most salient problems in the field of computer vision is object localization - the task of identifying, locating, and orienting an object within a cluttered scene. The human brain performs phenomenally at this task, and is able to perform whole-scene image processing with very high accuracy in under 150 milliseconds [1]. In comparison, object localization has proven to be difficult to mimic in computer systems with similar speed and accuracy. This difficulty can be attributed to a variety of factors, including viewing position, object rotation, scene complexity, occlusion, illumination and lighting, and object deformation [2].

1.1.1 2D Object Localization

Due to the availability and ease of use of 2D images, many techniques have been developed to tackle this problem within two-dimensional photographs. Some of these techniques include traditional classification techniques, such as local feature matching - finding correspondences between local features of two images containing the same object to create a global match [3] - and more complex machine learning-based efforts that aim to emulate the biological visual cortex to improve computer-based localization [4]. However, 2D localization approaches

are highly sensitive to illumination, shadows, occlusion, and pose [5]. Per contra, three-dimensional object localization is resistant to many of these drawbacks of 2D localization, and with recent advances in resolution, accuracy, and accessibility of three-dimensional depth sensors such as the Microsoft Kinect sensor [6] and Intel RealSense cameras [7], a substantial portion of recent work in computer vision has been on three-dimensional object localization.

1.1.2 3D Object Localization

The task of 3D object localization, can be defined as follows [8]: Given a model and a scene that contains this model, what is a transformation that, when applied to the model, causes it to match a subset of the scene? Due to the nature of three-dimensional data obtained from depth sensors, which are primarily point clouds - sets of points in a 3D coordinate system that represent the surfaces present in the scene - many of the problems with 2D localization, such as lighting, shadows, and viewing position, no longer need to be considered in 3D localization. However, 3D object localization comes with its own set of challenges. For example, data produced by the Microsoft Kinect tends to be noisy and sparse, especially with increasing distance from the sensor's position [9].

1.2 The Scene Registration Problem

One common extension of object localization is scene registration. In scene registration, instead of locating a pre-defined model within a 3D scene, two or more incomplete scene views taken from different sensor positions and orientations are mapped onto one another to create a composite scene cloud. An illustration of the difference between localization and registration can be seen in Figure 1.1. Scene registration is a key problem for vision-based autonomous robot systems, as many SLAM (Simultaneous Localization and Mapping) algorithms rely on stable methods to align sensor readings over time to create a map of the robot's environment [10].

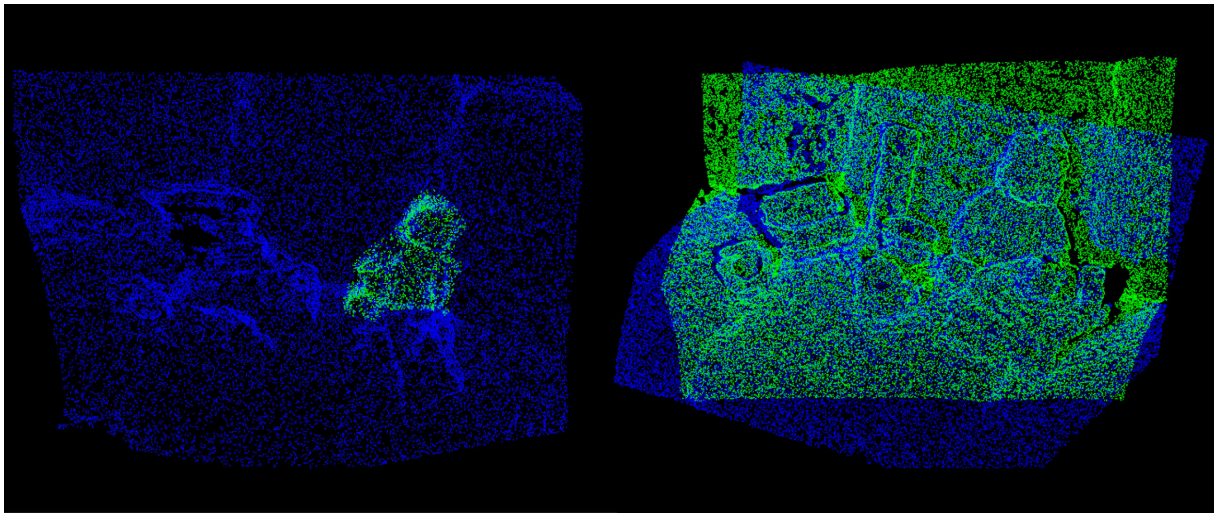


Figure 1.1: Illustration of 3D object localization (left) and scene registration (right). In localization, a pre-defined object (stuffed bear in green) is located within a scene (blue), while in registration, two partial scans of the same scene taken from different views are mapped to one another to create a composite point cloud.

1.3 Prior Work on 3D Localization and Registration

Initial efforts to work on 3D object localization and scene registration centered around voting methods, in which aspects of the scene cast votes in the parameter space of the solution. In Hough Voting [11], one of the more common voting methods, the solution space of transformations is broken down into various buckets. Each feature within the scene then casts a vote for the various possible object or scene transformations in this space. After voting concludes, the bucket with the most votes is considered as a solution to the problem. Other voting methods include Geometric Hashing [8], which is similar to Hough Voting, but votes on geometry instead of on object or scene pose. While voting methods tend to perform remarkably well in simple scenes or with exact data, adding occlusion, clutter, or even a minimal amount of sensor noise can lead to dramatically lowered performance [12].

As a result, more recent efforts in 3D object localization and scene registration have focused on correspondence-based methods. This class of solutions starts by creating local geometry descriptors for cloud points in both the model and the scene in which the model is being searched for. Many attempts to create local geometry descriptors have been informed

by 2D image-based feature descriptors. For example, Zaharescu et al. [13] created a 3D feature descriptor named MeshHOG that attempted to extend the Scale Invariant Feature Transform (SIFT) [3] algorithm into 3D space. Sehgal et al. [14] used a square root scaling technique to reduce three-dimensional point cloud data into two-dimensional space, allowing them to apply standard 2D feature descriptors to 3D data. Other groups have created three-dimensional harmonic shape context descriptors [15] and local multidimensional histograms of multiple features [16].

Once points within the model and scene clouds are assigned descriptors, matches between points in the model and scene are created by comparing these feature descriptors to each other. Next, an initial alignment between these clouds is created. One of the more commonly used techniques for initial alignment is the random sample consensus (RANSAC) algorithm [17], which first creates an initial estimate of a transformation from a few points of the data, and then iteratively improves this transformation to reduce the number of correspondence outliers in the transformation. Compared to voting methods, using RANSAC with feature descriptors is highly-resistant to both noise and occlusion [18], but only provides a rough estimation of the true alignment between model and scene, and is also expensive to compute if there are many candidate locations to test. Therefore, once an initial alignment is provided by RANSAC, another method, such as the Iterative Closest Points algorithm [19] is often used to improve on the initial alignment to minimize alignment error.

1.4 Current Challenges in 3D Object Localization

With recent improvements to sensor density, collected point cloud data tends to be very massive, with even inexpensive sensors like the Microsoft Kinect generating nine million points every second [9]. Unfortunately, increased point cloud density poses a problem for 3D object localization, as runtime complexities of all of the previously described algorithms are highly dependent on the number of points within a point cloud. For example, calculating

SIFT features is at least quadratic [20], while the RANSAC algorithm is approximately cubic [18] to the number of points in the point cloud. The original ICP algorithm ran quadratically to number of points in the cloud [21], and while kd-tree improvements [22] were able to bring this down to $O(n * log(n))$ and subsequent local search optimizations were able to make ICP run in linear time to number of points [23], every function in correspondence-based solutions to 3D object localization run at least linearly to the number of points in the point cloud. Thus, the runtime of object detection with modern sensors is deeply tied to the density of the generated point cloud.

Some effort has already been made to reduce point cloud density without affecting localization accuracy. For example, Gomes et al. was able to use foveated point cloud resampling to reduce localization runtime by 85 percent while only reducing accuracy by 8.4 percent [24]. In their method, after an object is initially located, the point cloud is resampled to remove points further away from the object's last-seen position. This limits the number of points that need to go through feature generation, RANSAC, and ICP to increase the speed of object detection. However, foveated resampling still requires an initial, computationally-intensive registration and can only function if an initial guess of object position is known, making it unsuitable for single-shot object detection in an unknown scene.

There has been other research into non-uniform resampling of point clouds. Chen et al., after representing a 3D point cloud as a graph, apply a high-pass Haar-like graph filter to point cloud data in order to filter out cloud points that had have low variability with respect to their neighbors [25]. By doing so, they selectively resample the cloud to preserve contours and edges of objects while removing points that offer little information relative to points in the area. Such contour-enhanced resampling techniques are able to reduce point cloud density while keeping points that are more likely to be useful to represent the object in the cloud. However, such work has not yet been applied to the 3D object localization or scene registration problems.

1.5 Thesis Goals

The primary goal of this thesis is to tackle the aforementioned challenge to object localization and scene registration - to decrease point cloud density to allow for faster computation of a solution to one of these problems without a corresponding loss of accuracy. To do this, I preprocess point clouds through contour-enhanced graph-based resampling before using the rest of the correspondence-based matching pipeline. I also perform a comparative analysis of using different depth sensors and point feature descriptors in generating solutions to the point cloud matching problem to determine which setup of hardware and algorithms is best for performing localization and registration.

Chapter 2

Materials and Methods

2.1 Point Cloud Acquisition

Due to its ubiquity and inexpensiveness, the Microsoft Kinect was used to collect all data for this thesis. Initial data was collected using the Kinect v2, which shipped with the Xbox One, as it has a higher depth camera resolution, wider field of view, and higher precision. However, being a time-of-flight depth sensor, the Kinect v2 performs poorly at depth discontinuities, making erroneous point estimates at object boundaries known as "flying pixels" [26]. An example of this flying pixel effect seen in Kinect v2 data can be seen in Figure 2.1.

Figure 2.1: A side-view comparison of similar scenes captured by the Kinect v1 (left) and Kinect v2 (right) at depth discontinuities. The Kinect v2 demonstrates significant flying pixel effects, while the Kinect v1 accurately trims the point cloud at discontinuities.

Furthermore, the Kinect v2's depth determination is affected by object colors - darker surfaces have a higher depth variability that is estimated to be four times as high as the depth variability of brighter surfaces [27]. As such, black surfaces, when imaged by the Kinect v2, tend to take on a distorted and warped appearance, as can be seen in Figure 2.2 below.

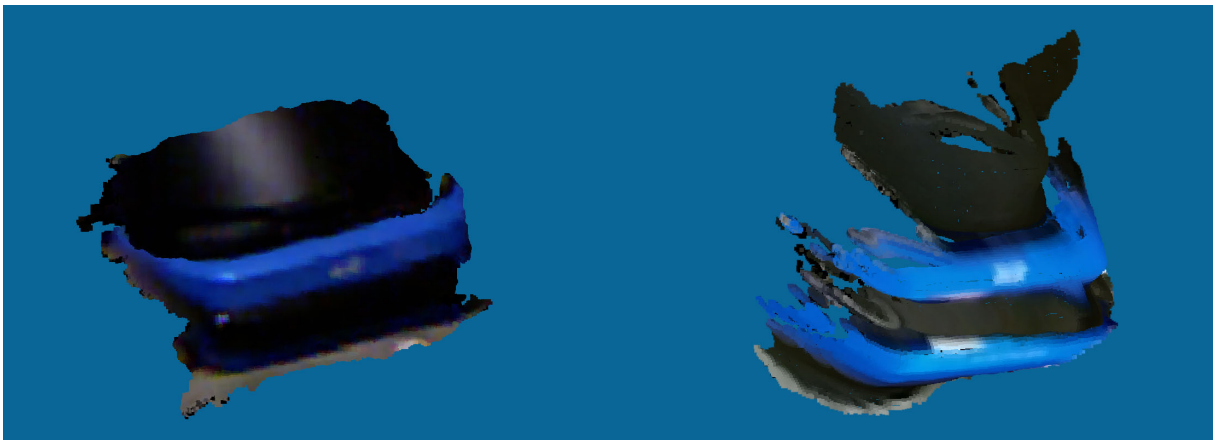


Figure 2.2: Comparison of a virtual reality headset captured by the Kinect v1 (left) and Kinect v2 (right). The Kinect v2 demonstrates poor depth mapping of the black headband, which appears uneven. Note, however, that the Kinect v2 provides improved details than the v1, showing the ridge between the two blue bands on the headset that isn't captured in the point cloud on the left.

Despite this, the Kinect v2 demonstrates a higher depth resolution than the Kinect v1, and is able to better capture small surface details on imaged objects. Furthermore, the color camera on the Kinect v2 provides much higher contrast and improved color reproduction as compared to the v1's color camera. The contour-enhanced resampling filter used to downsample clouds also has the added benefit being able to remove some of the flying pixels present in v2 clouds. Due to all this, both Kinect v1 and v2 point clouds were used in developing the algorithms presented below.

The point clouds themselves were captured using the Kinect Fusion Explorer [28], a open-source tool developed by Microsoft for 3D surface reconstruction of scenes that is distributed with the Kinect for Windows Developer Toolkit. The original Fusion Explorer only supports output to mesh-based data formats (STL, OBJ, and PLY files), so an extension to the Fusion

Explorer source was created that allows output in PCD (Point Cloud Data) file format by extracting points from the voxels generated by Kinect Fusion.

All further post-acquisition processing and analysis was done using Point Cloud Library [29], an open-source library used across academia and industry for point cloud-based research. All code was written using PCL version 1.8.1. Many of the figures presented in this paper were created by visualizing PCD files in CloudCompare [30], another open-source point cloud processing software that includes a suite of powerful visualization tools, while all graphs were created using the matplotlib and seaborn libraries for Python.

2.2 Transformation Estimation Pipeline

The approach to object localization and scene registration presented in this paper is modelled after the one described by Holz et al. in "Registration with the Point Cloud Library" [31]. An easily digestible flowchart of the process is provided in Figure 2.3.

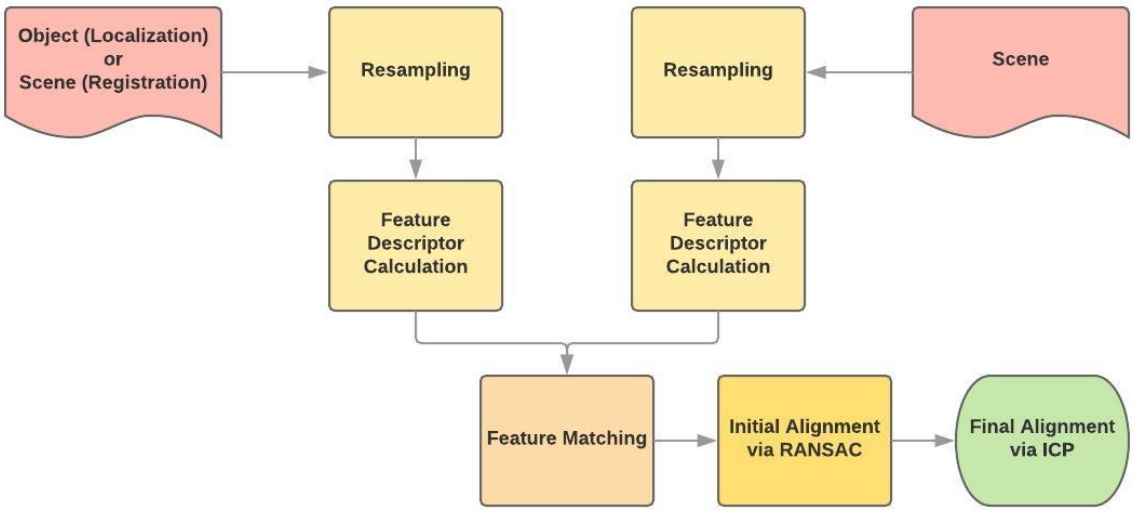


Figure 2.3: A brief overview of the steps of transformation estimation for object localization and scene registration.

After the two initial point clouds are loaded into memory, both clouds are resampled to reduce the number of points used in downstream calculations. Next, feature histograms are

calculated for each of the points in the downsampled clouds, which are then matched together to create a correspondence from each point in one cloud to its most similar point in the other. Random Sample Consensus (RANSAC) is next used to fit an initial transformation between the clouds that is finally refined more precisely through the Iterative Closest Point (ICP) algorithm.

2.3 Point Cloud Downsampling Techniques

In this paper, I compare three different resampling algorithms - uniform random resampling, voxel-grid resampling, and graph-based contour-enhanced resampling. The first two are in common use today as downsampling techniques, and are built into PCL as the RandomSample and VoxelGrid classes. Contour-enhanced downsampling, on the other hand, is a very recent technique (published in 2017 [25]), and needed implementation from scratch. I present an overview of these three methods below.

2.3.1 Uniform Random Resampling

In uniform random resampling, each point within the cloud has an equal probability p of being included in the final sample. As a result, differences in density between areas of the cloud are preserved, and when using sensors like the kinect (which have decreasing cloud density as distance from the sensor increases), objects far away have a lower point density than nearby objects. Uniform random sampling also preserves the location of points selected for inclusion, so noise in the initial scan is not smoothed out during the resampling process. An example of downsampling using the uniform random method at different values of p can be seen in Figure 2.4.

Figure 2.4: The original scan (left); Uniform random downsample with $p = 0.25$ (center); Uniform random downsample with $p = 0.05$ (right).

2.3.2 Voxel-Grid Resampling

In voxel-grid resampling [32], a grid with side length l is created over the point cloud. Each of these grid voxels then generates a new point at the position and color centroid of all of the original cloud points within that voxel. Voxel-grid resampling fixes the two issues with uniform random resampling mentioned above. First, as the downsampling grid is regular over the original point cloud, the density of the sampled cloud will be uniform throughout instead of less dense with increasing distance from the camera. Second, minor local noise variations in the point cloud will be smoothed out as each group of points within the voxel is resampled to the centroid of that point group. An example of downsampling using the voxel-grid method is provided in Figure 2.5. Note how the voxel-grid resamples appear much less noisy when compared to the clouds generated using uniform random resampling.

Figure 2.5: The original scan (left); Voxel-grid downsample with $l = 2.5mm$, preserving 24.1% of points (center); Voxel-grid downsample with $l = 6mm$, preserving 4.8% of points (right).

2.3.3 Contour-Enhanced Resampling

The final resampling technique is contour-enhanced graph-based resampling as presented by Chen et al. [25]. In this method, each point x_i in the graph is assigned a response $f_i(X) = \|(h(A)X)_i\|_2^2$, where $h(A)$ is a graph filter. In the Haar-like [33] filter presented here, $h(A) = I - A$, where A is the graph shift operator $A = D^{-1}W$, with W being an adjacency matrix between points in the point cloud with edge weights equal to $W_{i,j} = e^{-\|x_i - x_j\|_2^2/\sigma^2}$ and D being a diagonal matrix where $D_{i,i}$ is the sum of every element in the i^{th} row of W. To speed response calculation time, a k-d tree is built over the point cloud and the weight $W_{i,j}$ is assumed to be zero for any points more than k nearest neighbors away from the point under evaluation. With k-d three optimizations, the overall runtime for point response calculation is $O(nk \cdot log(n))$, where n is the number of points in the cloud and k is the number of neighbors for which weights are calculated. Both k and σ are parameters to the resampling algorithm. Once responses are calculated for every point in the graph, resampled points are chosen via weighted random selection with point selection weight equivalent to the response $f_i(x)$ of the point. A demonstration of contour-enhanced downsampling can be seen in Figure 2.6.

Figure 2.6: The original scan (left); Contour-enhanced downsample preserving 26.9% of points (center); Contour-enhanced downsample preserving 4.97% of points (right). In these images, k is equal to 10 and σ^2 is 0.0005.

Additionally, both color-aware and color-blind versions of the contour-enhanced filter were implemented. In the color-blind variant, the distance between two points x_i and x_j in the cloud is the standard Euclidean distance in 3D space between two points. In the color-

aware variant, the red, green, and blue channels each independently act as three additional Euclidean dimensions with range from 0 to 1 based on the color value (stored as 8-bit color from 0 to 255). The differential effects of the color-aware and color-blind filters can be seen in Figure 2.7. Particularly, note the text on the shoebox at the back-right of the scene. The text is preserved strongly in the color-aware filter, while the color-blind filter does not sample the text as intensely, as when not provided with color information, the filter sees the text simply as part of a plane.

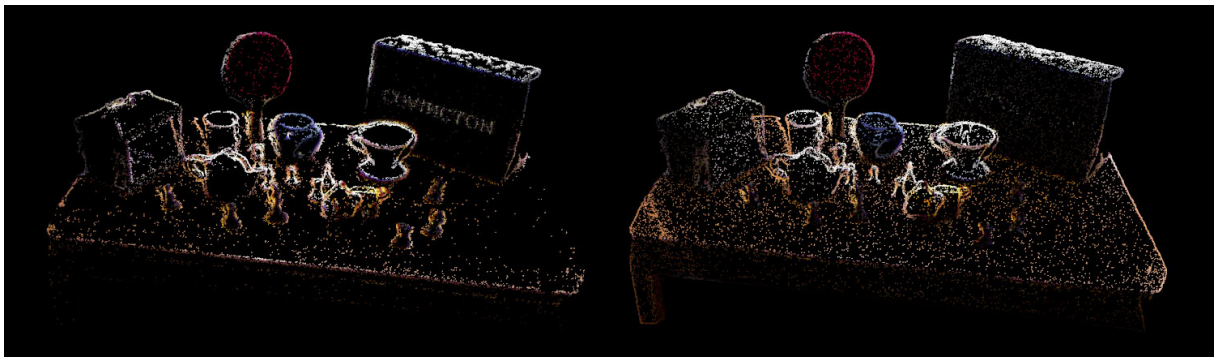


Figure 2.7: Color-aware contour-enhanced downsample (left); Color-blind contour-enhanced downsample (right). Both images contain a relatively equal number of points (approximately 8% of the original point cloud).

The decision to use the color-aware version of the contour-enhanced filter depends on the nature of the collected data itself. Because object color can vary wildly depending on lighting, use of the color-aware filter can lead 3D localization algorithms to develop many of the issues present in 2D localization algorithms mentioned earlier, such as sensitivity to illumination, shadows, and occlusion [4]. However, when the two input point clouds are both evenly illuminated, as they are in the scene registration task, using the color-aware filter can improve transformation estimation. Furthermore, use of the color-aware filter with Kinect v2 data helps mitigate the number of flying pixels present in the downsampled cloud, as the flying pixels tend to take on the same color as pixels at the edge of an object. In Figure 2.8, we can see how the color-aware variant of the contour-enhanced downsampling algorithm captures more useful surface points and fewer flying pixels than the color-blind variant in extremely noisy Kinect v2 data.

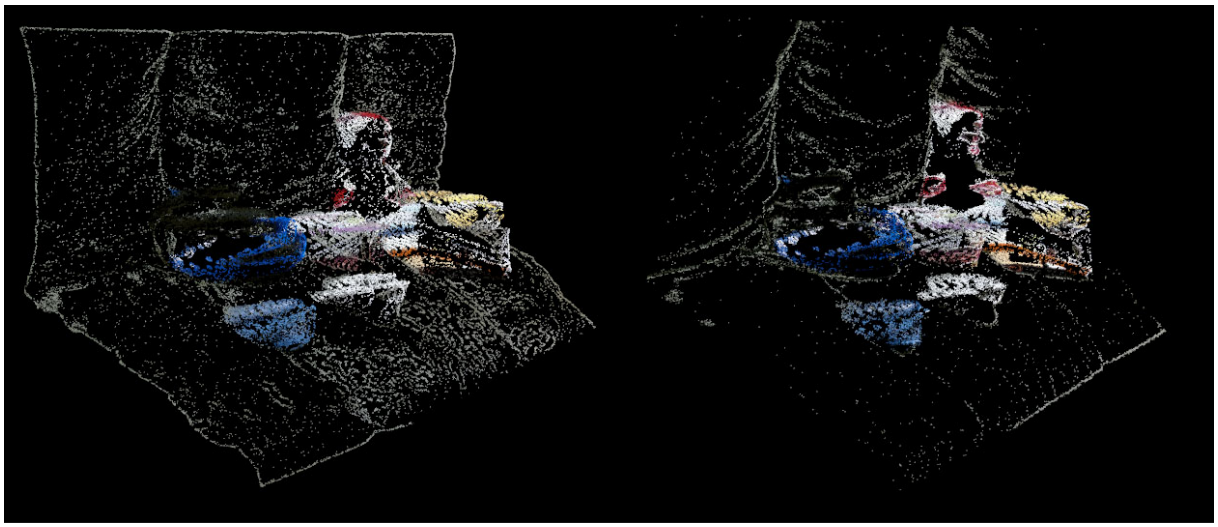


Figure 2.8: Color-aware contour-enhanced downsample (left); Color-blind contour-enhanced downsample (right). Both images contain a relatively equal number of points (approximately 8% of the original point cloud).

The contour-enhanced downsampling algorithm still suffers from the two pitfalls of uniform random downsampling mentioned above. Since weights are assigned to all points in the cloud regardless of location or number of neighbors in a point's proximity, the output cloud will still be less dense as distance from sensor increases. Similarly, contour-enhanced resampling preserves location of points selected for inclusion, so the resultant cloud will still contain sensor noise from the initial scan. As such, it may be useful to preliminarily downsample the cloud using the voxel-grid method prior to contour-enhanced downsampling in order to achieve more useful data.

2.4 Feature Descriptor Calculation

As PCL offers almost thirty different local feature descriptors for point clouds, it was necessary to compare different descriptors and select the ones that would work best for fast point cloud matching using Kinect data. Many descriptors were disqualified from use due to downsides that render them ineffective for the data being used - for example, the Rotation-Invariant Feature Transform (RIFT) relies on an intensity gradient over the point cloud,

making it highly sensitive to shadowing and occlusion.

Because speed is a priority when attempting to handle realtime object localization and scene registration tasks, other descriptors were disqualified because they did not support OpenMP-based parallelization of feature computation. On the target system's CPU (Intel i7-6820HK), OpenMP acceleration was able to reduce descriptor calculation time by approximately 85% for all of the descriptors tested below, as can be expected from computation across 8 separate threads. As a result, three candidate feature descriptors were selected for use in this thesis - Fast Point Feature Histograms (FPFH) [34], Signature of Histograms of Orientations (SHOT) [35], and Color SHOT, a color-aware SHOT variant [36].

FPFH descriptors are generated by calculating θ, ϕ, and α rotation values for each of 11 spatial bins containing neighbors within a given radius, creating a final vector of length 33. FPFH feature calculation is $O(n \cdot k^2)$, where n is the number of points in the cloud and k is the number of neighbors within the point's radius. SHOT descriptors are generated by first calculating a 9-dimensional reference frame based on neighboring surface points within a given radius, then creating an isotropic grid with 32 bins (10 angles within each bin) based on the reference frame to create a final vector of length 329 (320 dimensions to describe the bins and 9 dimensions for the reference frame). Color SHOT operates similarly to SHOT, but calculates an additional set of bins to handle color information, leading to a final feature vector of length 1344. SHOT feature calculation is in $O(n \cdot k)$, with n and k defined as above.

Because FPFH has the shortest feature vector of all three descriptors under consideration, correspondence calculation between FPFH features takes very little time, as is demonstrated in Table 2.9. However, because FPFH descriptor calculation is quadratic with respect to the number of neighbors of each point, generating feature vectors takes an exceptionally long time. Furthermore, SHOT and FPFH provide similar accuracy in object localization tasks, with FPFH performing slightly better with flat objects and SHOT performing slightly better with objects that have more varied surfaces. Since this difference appeared to be fairly minimal, and since SHOT feature calculation is much faster than FPFH feature calculation,

I eliminated FPFH as a candidate for the point cloud matching task.

Feature	Vector Size	Descriptor Calculation	Correspondence Matching
FPFH	33	45.631s	0.815s
SHOT	352	4.598s	14.929s
Color SHOT	1344	5.164s	80.044s

Table 2.9: A comparison of feature and correspondence calculation runtime of the three studied feature descriptors. The results in this table are given based on calculations done on 5% of the points (approximately 35,000 points in the scene cloud and 1,500 points in the object cloud) in non-downsampled clouds (approximately 700,000 points in the scene cloud and 30,000 points in the object cloud). Each cell contains the average runtimes of 5 tests.

After deciding to use one of the SHOT descriptors for further calculation, I conducted simple tests to determine how useful color data is in object localization and scene registration. In Figure 2.10, I match the same teapot object cloud to two differently-lit scenes containing the teapot. The scene on the left is lit using a rectangle-shaped area light centered directly above the table, while the scene on the right is lit using a single point light centered above a point approximately 50 centimeters in front of the sofa. As can be expected of a color-based description method, generated features were highly dependent on scene lighting. When scene and object capture occur in similar lighting circumstances, Color SHOT is able to provide highly regular correspondences between the object and the scene. When the lighting is slightly different, as it is in the right subfigure of Figure 2.10, the correspondences returned by matching Color SHOT features are irregular and unusable.

Therefore, while Color SHOT may be useful in providing matches between an object in a scene with similar lighting to the lighting provided during object cloud capture, it is not as useful for the generalized object localization problem, especially considering that correspondence matching for Color SHOT features takes more than four times as long as correspondence matching for normal SHOT features. Thus, I opted to use SHOT feature descriptors for all object localization experiments.

When it comes to scene registration, however, both point clouds will be similarly lit,

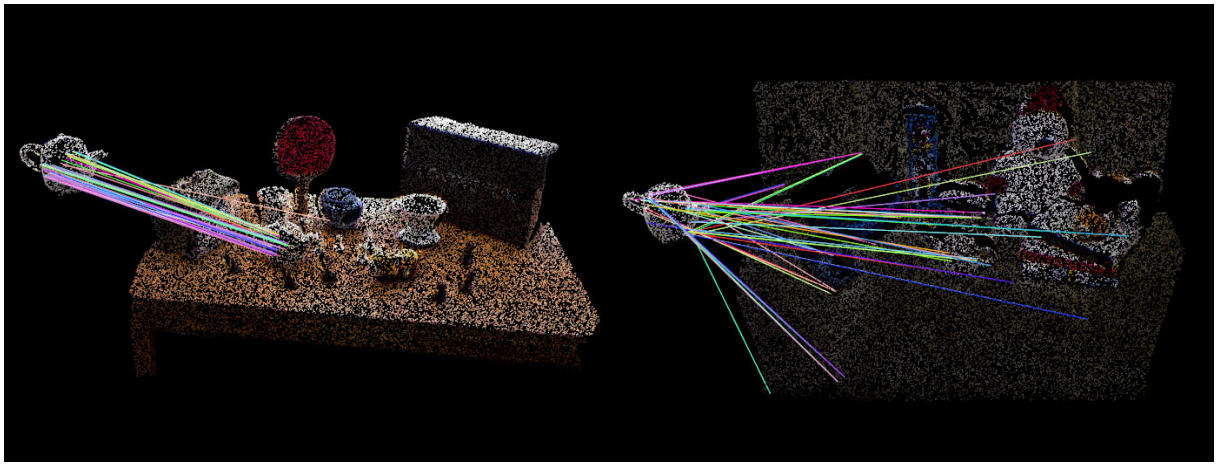


Figure 2.10: Correspondences calculated using Color SHOT with similar scene and object capture lighting (left); Correspondences calculated using Color SHOT with disparate scene and object capture lighting (right).

which makes Color SHOT much more useful. Figure 2.11 demonstrates a sample scene registration task in which both input point clouds are downsampled using a 0.8% voxel-grid resampling filter. We see that, when using Color SHOT, a much higher proportion of correspondences are horizontal (true correspondences), while the correspondences calculated with normal SHOT are much more irregular.

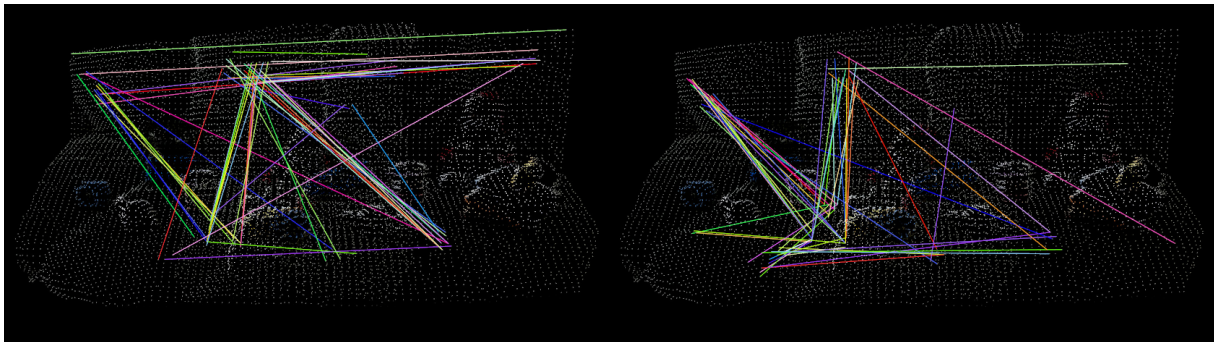


Figure 2.11: Correspondences calculated using Color SHOT (left); Correspondences calculated using normal SHOT (right). The subsampled point cloud contained just 0.8% of the number of points that the original point cloud had.

The more regular correspondences obtained when using Color SHOT lead to improved results in alignment. As shown in Table 2.12, using Color SHOT resulted in a rotational improvement of 17.5% and a translational improvement of 32.9% over regular SHOT for the experiment conducted in Figure 2.11. However, this improvement in accuracy comes at a cost

- the use of Color SHOT leads to a 432% increase in runtime over the use of standard SHOT. Furthermore, this accuracy improvement drops off as the downsampling rate is decreased. At 6% uniform random downsampling, both techniques have similar registration accuracy while Color SHOT takes more than four times as long to compute features and correspondences for, primarily due to the increased number of vector dimensions of the Color SHOT descriptors. Color SHOT is thus only more useful than standard SHOT for scene registration when input point clouds are sparse or if an extreme amount of downsampling is done prior to feature calculation.

Feature	Runtime (per point)	Rotation Error	Translation Error
SHOT	8.30ms	1.26°	3.01cm
Color SHOT	44.15ms	1.04°	2.02cm

Table 2.12: Demonstration of improved scene registration when using Color SHOT over normal SHOT in a very sparse point clouds at the cost of significant extra runtime. Reported metrics are results that have been averaged over five runs of the registration algorithms.

2.5 Correspondence Matching and Alignment

Once features are generated for each point cloud, a correspondence is created between each point in the source scene or object cloud and its closest point in the target scene (determined by L2-distance between feature vectors). In the correspondence figures presented herein, as in Figure 2.10, only the top 50 correspondences between the clouds are graphed to reduce clutter.

Since many of these correspondences are inaccurate, as can clearly be seen in Figure 2.11, the initial correspondences are fed into a RANSAC algorithm that iteratively rejects false correspondences until a satisfactory model is fit between the two clouds. Finally, ICP is used to find a transformation between clouds that results in a locally minimal mean squared error (MSE) between the points in the source cloud and their closest neighbors in the target cloud.

2.6 Error Metrics and Transformation Analysis

The accuracy and error metrics presented in this paper are used to compare location and orientation produced by the cloud alignment algorithms to a ground truth for both object localization and scene registration. The ground truth transformation is obtained by manually aligning point clouds using CloudCompare, then running ICP to minimize MSE between clouds. This is a rigid transformation, preserving distances between every pair of points in the cloud, and is represented by a standard 4-by-4 rigidbody transformation matrix [37] in the form $\left[\begin{smallmatrix} R & t \\ 0 & 1 \end{smallmatrix}\right]$, where R is a 3-by-3 rotation matrix and t is a 3-by-1 translation matrix. The RANSAC and ICP algorithms performed in localization and registration produce similarly formatted 4-by-4 rigid transformation matrices.

The translation error between a ground truth transformation A and a calculated transformation B is simply the L-2 distance between the translation components of A and B, or $\|t_A - t_B\|_2$. The rotation error is slightly more difficult to compute, and first requires knowing the rotation between the rotation components of A and B, which can be calculated as $R_A^T \cdot R_B$. This resultant rotation matrix can then be converted to axis-angle form. For 3D rotations, the following formula [38] can be used to determine rotation error: $Tr(R_A^T \cdot R_B) = 1 + 2 \cdot cos(\theta)$. Translation and rotation error are the two primary error metrics used in Chapter 3 of this thesis.

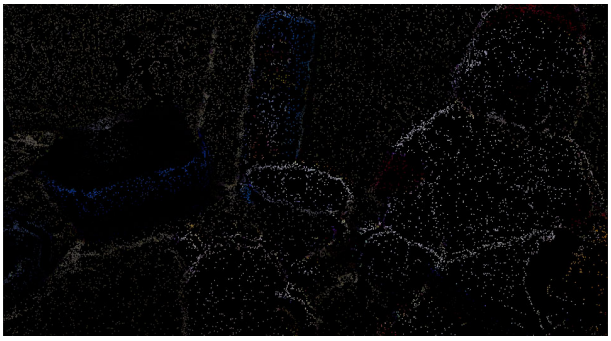
Chapter 3

Results

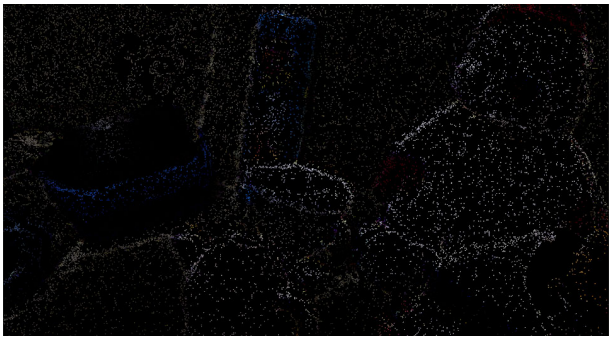
3.1 Contour-Enhanced Resampling Parameter Tuning

Given that contour-enhanced resampling requires running a k-d tree search at every point in the input point cloud, the algorithm runs in $O(nk \cdot log(n))$ time, making it much more computationally complex than uniform random resampling (which runs in $O(n)$) and voxel-grid resampling (runs in $O(n + m \cdot log(m))$, where m is the number of non-empty voxels in the grid). Though the $n \cdot log(n)$ factor of contour-enhanced resampling cannot be reduced without implementation of more advanced tree search techniques, such as an Approximate Nearest Neighbor search [39] or Local Sensitivity Hashing [40], the k factor can be easily reduced simply by selecting a lower value for k.

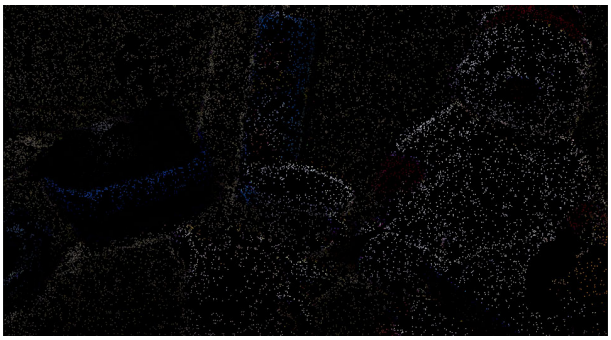
However, lowering the value of k also lowers the amount of information that each point has about its neighborhood, reducing the strength of contour estimation. Figure 3.1 provides a demonstration of contour strength loss with decreasing k. While the $k = 15$ case (a) has rather strongly preserved contours, the $k = 2$ case (d) looks almost comparable to random resampling due to the loss of neighbor information. This effect is particularly noticable when looking at the circle-shaped object (rim of a coffee filter) in the middle of each subfigure - it's evident that the edges are much more clear in the $k = 15$ case than they are when $k = 2$.



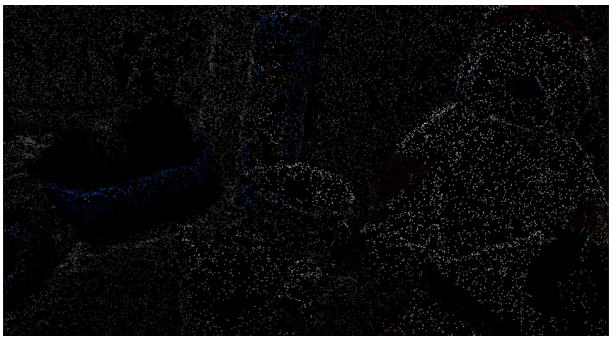
(a) $k = 15$



(b) $k = 10$



(c) $k = 5$



(d) $k = 2$

Figure 3.1: k-dependence of Contour Strength. All subfigures contain a 5% downsample

Figure 3.2 shows the runtime of contour-enhanced resampling on a cloud containing approximately 743,000 points at various values of k. While a resampling that uses a k of 15 provides much higher contour strength than one that uses a k of 2 does, it also takes approximately twice as long to run, with an increase of approximately 69.3 milliseconds in runtime per extra neighbor used in weight calculation.

However, increased contour strength does not directly translate to corresponding improvements in transformation estimation. When contours are too strong, SHOT contains many empty spatial bins in the calculated isotropic spherical grid for each point, resulting in decreased feature descriptor quality. As such, it was necessary to select a value of k that provides a good balance between runtime, contour strength, and transformation estimation quality. Figure 3.3 shows the effect of varying k on transformation error for a scene registration task. To regularlize across runs, downsampling was done such that downsampled clouds contained approximately 1.8% of the points that input clouds did. Both very low

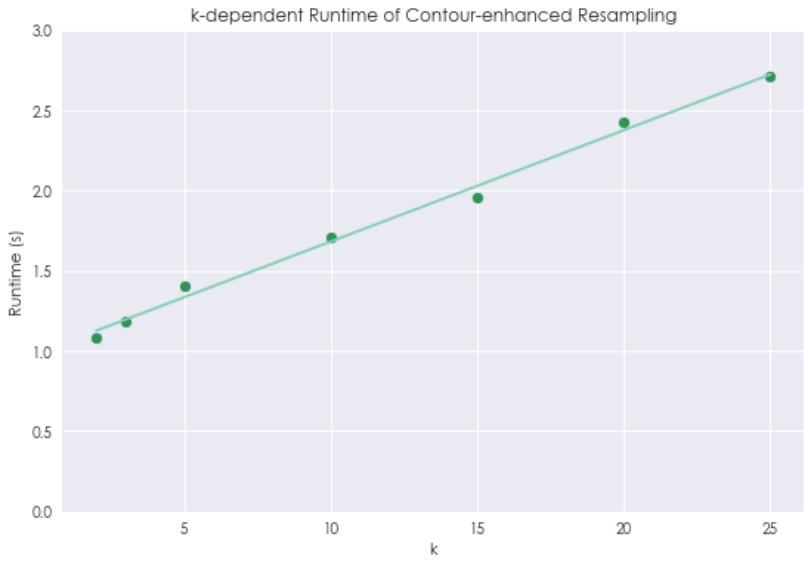


Figure 3.2: Runtime of contour-enhanced resampling at various values of k. Each point on the graph is the result of averaging together runtimes from 5 runs of resampling.

and very high values of k lead to increased error in translation estimation, with a value of approximately $k = 5$ having the best performance. Provided that resampling using $k = 5$ provided the best transformation accuracy while only being 30.2% slower than resampling using $k = 2$, k was set to be equal to 5 for the rest of this analysis.

Figure 3.3: Translation error of estimated transformation at varying values of k. Each point on the graph is the result of averaging together errors from 5 runs of scene registration. Tests at different downsampling proportions had similar results (data not shown).

3.2 Resampling Algorithm Speed

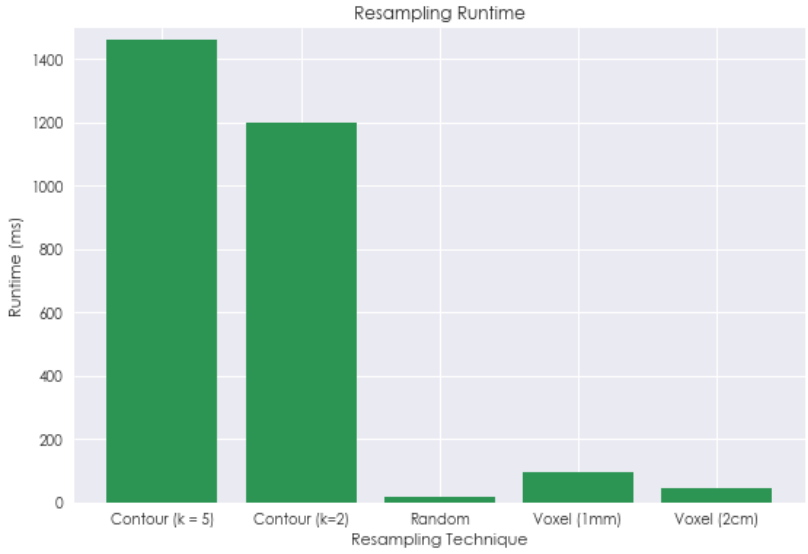


Figure 3.4: Comparison of runtimes of each resampling algorithm on a cloud containing approximately 743,000 points. Each bar presented here is the result of averaging together 50 runs of each technique.

Next, I present a brief comparison of each resampling algorithm's runtime. As seen in Figure 3.4, these results are as expected based on the asymptotic dynamics of each method, described in the previous section - contour-enhanced resampling is much slower than both uniform random and voxel-grid resampling, largely due to k-d tree generation and search. Basic performance profiling revealed that 12.30% of the runtime for contour-enhanced resampling was spent building a k-d tree over the input point cloud while another 50.68% was spent searching for the k-nearest neighbors of each point.

Due to the large k-d tree creation and search overhead, contour-enhanced resampling is best used (1) when it can significantly decrease the number of post-downsampling points (and thus overall localization or registration runtime) while maintaining similar accuracy, or (2) when it can offer improved accuracy over random or voxel-grid resampling at the same level of downsampling. Both of these cases are tested for object localization and scene registration in the sections to follow.

3.3 Object Localization

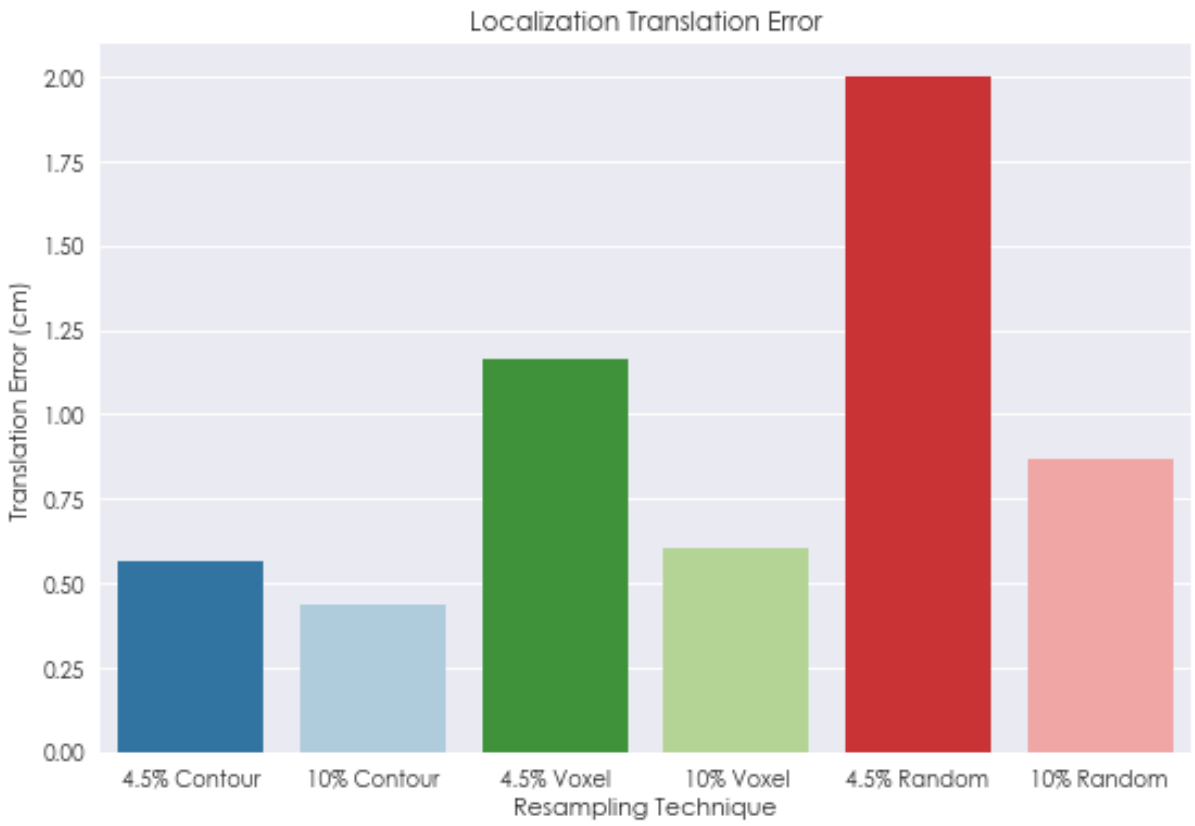


Figure 3.5: Comparison of translation error relative to ground truth for various resampling strategies. Each bar presented here is the result of averaging together 5 runs of each technique.

The localization task tested in this section is the same as the task presented in the left panel of Figure 2.10, in which a teapot model is localized within a busy and slightly occluded scene containing that teapot. Data was obtained from the Kinect v1, and SHOT features were calculated using a radius of 2 centimeters for all presented experiments. The ground truth transformation matrix was obtained via manual alignment followed by ICP in CloudCompare. As discussed in Section 2.3.3, the color-blind variant of contour-enhanced resampling is used for object localization.

As can be seen in Figures 3.5, use of contour-enhanced resampling can dramatically improve localization accuracy when the number of points post-resampling is held constant.

Furthermore, this improvement in localization estimation is not accompanied by a large increase in runtime. As shown in Table 3.7, contour-enhanced resampling introduces only a 3.6% performance overhead at a downsampling rate of 4.5%, and a 0.7% performance overhead at a downsampling rate of 10%, as most of the runtime is occupied by feature description and correspondence matching, not by resampling.

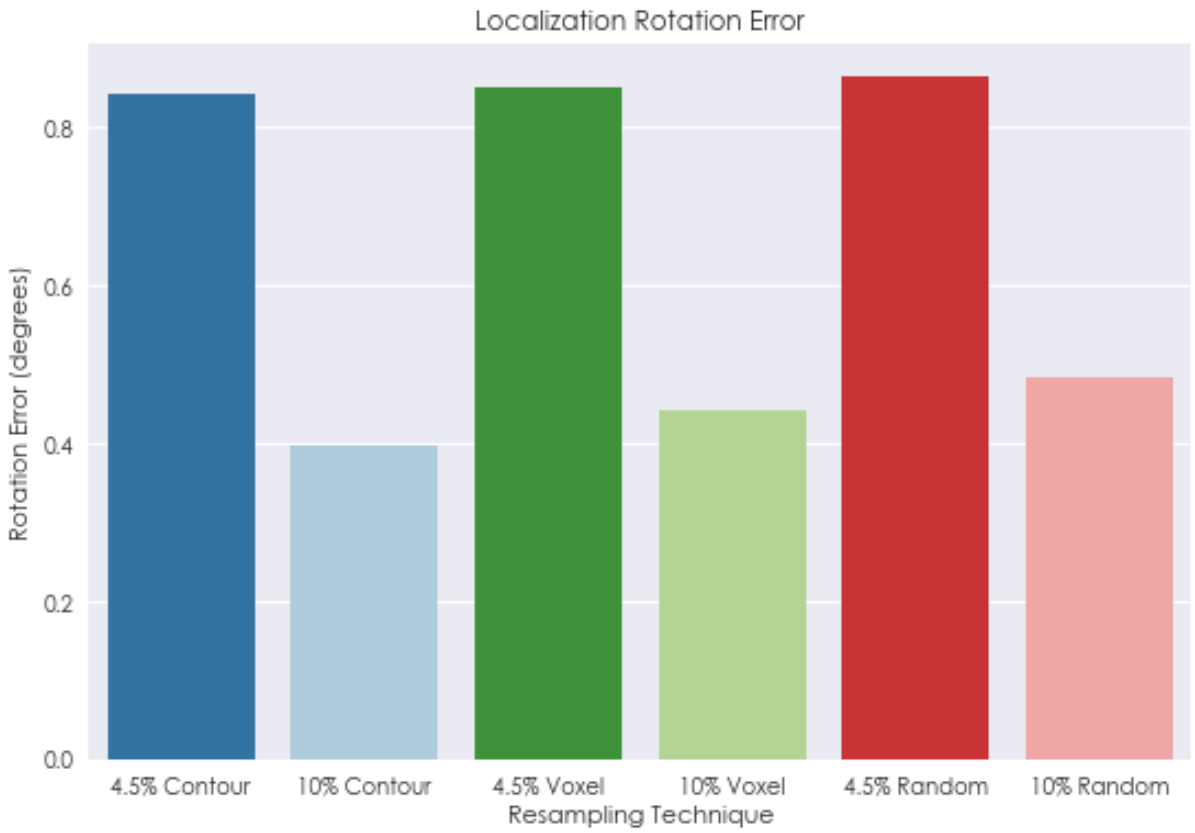


Figure 3.6: Comparison of rotational error relative to ground truth for various resampling strategies. Each bar presented here is the result of averaging together 5 runs of each technique.

While using contour-enhanced resampling leads to a decrease in translational error when compared to the other two resampling strategies, it does not provide much of a corresponding decrease in rotational error. As demonstrated in Figure 3.6, rotational error is significantly more correlated to the proportion of original points in the downsampled cloud than it is to the method of downsampling being applied. Nonetheless, when it comes to translational error, 4.5% contour-enhanced resampling is able to perform just as well as 10% voxel-grid

resampling despite taking less than a third of the runtime.

	Runtime (seconds)	
Resampling Technique	4.5% Downsample	10% Downsample
Uniform Random	20.863	69.601
Voxel-grid	20.580	68.623
Contour-enhanced	21.614	70.120

Table 3.7: Transformation estimation runtime of the various resampling strategies for a localization task. Reported metrics are results that have been averaged over five runs of the localization algorithms.

3.4 Scene Registration

Figure 3.8: Comparison of translation error relative to ground truth for various resampling strategies. Each bar presented here is the result of averaging together 5 runs of each technique.

The registration task tested in this section is the same as the task presented in the right panel of Figure 1.1, in which a scene captured from a central angle is registered to the same scene captured from above and to the left. Data was obtained from the Kinect v1, and SHOT features were calculated using a radius of 5 centimeters for all presented experiments. The ground truth transformation matrix was obtained via manual alignment followed by ICP in CloudCompare. As discussed in Section 2.3.3, the color-aware variant of contour-enhanced resampling is used for registration.

Figure 3.9: Comparison of rotational error relative to ground truth for various resampling strategies. Each bar presented here is the result of averaging together 5 runs of each technique.

The results seen in comparative analyses of resampling methods for scene registration differ markedly from the results seen for object localization, though contour-enhanced resampling still shows considerable improvement over the other two resampling methods. As seen in Figure 3.8, when it comes to translation error, all three resampling methods perform

relatively equally at the low downsampling rate of 1.8%. However, when the downsampling rate is increased to 10%, contour-enhanced resampling outperforms voxel-grid resampling by 34% and uniform random resampling by 67.6%.

Rotational error, on the other hand, is not impacted much by resampling rate, and depends more on the resampling technique being used. As shown in Figure 3.9, contour-enhanced resampling also is better when it comes to rotational error, outperforming voxel-grid resampling by 43.1% and uniform random resampling by 25.1% at a resampling rate of 1.8%.

	Runtime (seconds)	
Resampling Technique	1.8% Downsample	10% Downsample
Uniform Random	14.83	137.60
Voxel-grid	15.06	155.25
Contour-enhanced	39.74	151.72

Table 3.10: Transformation estimation runtime of the various resampling strategies for a registration task. Reported metrics are results that have been averaged over five runs of the registration algorithms.

As for runtime, contour-enhanced resampling performs relatively poorly compared to the other two methods at a 1.8% resampling rate, as a greater proportion of calculation time is spent doing the actual resampling. At higher rates, however, 10% contour-enhanced resampling does much better, and actually slightly outperforms 10% voxel-grid resampling due to the increased granularity of the voxel grid. Comparing translational accuracy across different methods, 1.8% contour-enhanced resampling is able to outperform 10% uniform random resampling while being 3.46 times as fast. Therefore, as in object localization, scene recognition efficiency and accuracy can be improved tremendously by using contour-enhanced graph-based resampling.

Chapter 4

Conclusions

In this thesis, I presented an efficient implementation of contour-enhanced graph-based resampling of point clouds that runs in $O(nk \cdot log(n))$ time. I develop both color-blind and color-aware versions of this technique, which I then apply to the problems of object localization and scene registration.

In object localization, use of contour-enhanced resampling prior to correspondence-based transformation estimation can reduce calculation runtime by more than 67% while maintaining a similar accuracy to a voxel-grid resampling method. When given a similar amount of time to run, contour-enhanced resampling can reduce translation error of localization by more than 50% over voxel-grid resampling. The contour-enhanced technique also has similar effects in scene recognition, and is able to alternatively either lower runtime while maintaining accuracy or improve accuracy while maintaining runtime, though these benefits tend to diminish as the amount of downsampling is increased.

Ultimately, by being able to effectively downsample point clouds to their important contour points, the contour-enhanced graph-based resampling technique presented in this thesis is able to effectively reduce runtime and error in downstream processing, making it a powerful tool within the field of computer vision.

Chapter 5

Future Works and Applications

5.1 Improvements to Contour-Enhanced Resampling

Though contour-enhanced resampling can help dramatically speed up downstream calculations in object localization and scene registration, the resampling algorithm itself is still somewhat slow - more than 12 times slower than voxel-grid resampling, primarily due to the k-d tree search that occurs for every point within the source point cloud. One important task to complete before contour-enhanced resampling can be used in a realtime framework is to reduce its runtime considerably. Using a heuristic-based search algorithm similar to the one presented by Jost and Hügli [23] could help dramatically accelerate contour-enhanced resampling runtime. Alternatively, using neural networks for 3D contour detection [41] could eliminate the need to search for neighbors to enable a much faster resampling algorithm.

Since each point in the original cloud is selected for inclusion in the final sample separately, parallelization of the contour-enhanced resampling algorithm could also be used to bring runtime down. Though initial attempts [42] to parallelize k-d tree search on GPUs failed due to kernel switching and the overlapping nature of voxelized point clouds, recent advances in GPGPU-based computation have made it possible to search for thousands of points within a k-d tree simultaneously [43]. Implementation of this technique could effec-

tively diminish the tree search overhead for contour-enhanced resampling to make it as fast as the voxel-grid resampling algorithm.

5.2 Generalized 3D Object Recognition

One problem with all of the 3D localization approaches mentioned in section 1.3 is that correspondence is tied to the exact model that is being searched for within the scene. That is, if a model is deformed in some non-rigid way, none of the above algorithms would be able to recognize it within the scene, even if they were able to recognize the original model. An example of a non-rigid deformation is joint movement - an algorithm that is able to find a model dog in a ”standing” pose within a scene would be unable to locate a ”sitting” version of the same dog within that scene, or even a different dog in the same standing position. Humans perform much better at this task than computer systems do, and are able to recognize objects even after significant deformations have been applied. Though some field-specific work has been done in recognition of deformed 3D objects, particularly in face [44] and isometric pose-deformation recognition [45], these techniques are too specific for general arbitrary 3D object recognition.

In the field of 2D object detection, on the other hand, deformed object recognition has been studied a lot more deeply. For example, Le et al. at Google were able to use unsupervised deep learning methods to create a detector that could identify if a face belonged to a cat, regardless of cat species and picture viewpoint [46]. Though their detector required around 10 million training samples, more recent advances in the field have been able to perform identification from only a single training example - a technique known as one-shot learning. One-shot learning techniques have potential to be used in object recognition tasks, as a single model (the training example) is to be located within a scene that contains some deformed version of that model.

One of the first uses of one-shot style learning was for signature verification [47]. In their

work, Bromley et al. created a new class of neural network that they dubbed a "Siamese Network." The Siamese Network was composed of two identical time-delay neural networks that attempted to learn vector representations of the signature. The vector representations were learned such that the distance between a signature and a forgery of the signature was maximized, while the distance between a signature and its reproduction by the same individual was minimized. The group was then able to feed the networks signatures outside of the training set and were able to detect 80 percent of forgeries with only 219 training examples. Siamese Networks have since been used for written language detection [48] and even more advanced image identification tasks like face verification [49] and arbitrary image patch comparison [50].

Though one-shot learning-based approaches have not yet been used to learn useful representations of three-dimensional data for the purposes of object recognition, their ability to learn vector representations of objects could be applied directly to the generalized 3D object recognition task if learning is done over voxelized representations of point cloud data. These vector representations could then be used to directly search for a voxelized scene subset that matches a voxelized object for generalized 3D object localization.

Bibliography

[1] Simon Thorpe, Denis Fize, and Catherine Marlot. Speed of processing in the human visual system. *nature*, 381(6582):520, 1996.

[2] Shimon Ullman et al. *High-level vision: Object recognition and visual cognition*, volume 2. MIT press Cambridge, MA, 1996.

[3] David G Lowe. Object recognition from local scale-invariant features. In *Computer vision, 1999. The proceedings of the seventh IEEE international conference on*, volume 2, pages 1150–1157. Ieee, 1999.

[4] Thomas Serre, Lior Wolf, Stanley Bileschi, Maximilian Riesenhuber, and Tomaso Poggio. Robust object recognition with cortex-like mechanisms. *IEEE transactions on pattern analysis and machine intelligence*, 29(3):411–426, 2007.

[5] Ajmal S Mian, Mohammed Bennamoun, and Robyn Owens. Three-dimensional model-based object recognition and segmentation in cluttered scenes. *IEEE transactions on pattern analysis and machine intelligence*, 28(10):1584–1601, 2006.

[6] Zhengyou Zhang. Microsoft kinect sensor and its effect. *IEEE multimedia*, 19(2):4–10, 2012.

[7] Mark Draelos, Qiang Qiu, Alex Bronstein, and Guillermo Sapiro. Intel realsense= real low cost gaze. In *Image Processing (ICIP), 2015 IEEE International Conference on*, pages 2520–2524. IEEE, 2015.

[8] Y. Lamdan and H. J. Wolfson. Geometric hashing: A general and efficient model-based recognition scheme. In *[1988 Proceedings] Second International Conference on Computer Vision*, pages 238–249, Dec 1988.

[9] Kourosh Khoshelham and Sander Oude Elberink. Accuracy and resolution of kinect depth data for indoor mapping applications. *Sensors*, 12(2):1437–1454, 2012.

[10] Dirk Holz and Sven Behnke. Sancta simplicitas-on the efficiency and achievable results of slam using icp-based incremental registration. In *Robotics and Automation (ICRA), 2010 IEEE International Conference on*, pages 1380–1387. IEEE, 2010.

[11] Federico Tombari and Luigi Di Stefano. Object recognition in 3d scenes with occlusions and clutter by hough voting. In *Image and Video Technology (PSIVT), 2010 Fourth Pacific-Rim Symposium on*, pages 349–355. IEEE, 2010.

[12] W Eric L Grimson and Daniel P Huttenlocher. On the sensitivity of geometric hashing. In *Computer Vision, 1990. Proceedings, Third International Conference on*, pages 334–338. IEEE, 1990.

[13] Andrei Zaharescu, Edmond Boyer, Kiran Varanasi, and Radu Horaud. Surface feature detection and description with applications to mesh matching. In *Computer Vision and Pattern Recognition, 2009. CVPR 2009. IEEE Conference on*, pages 373–380. IEEE, 2009.

[14] Anuj Sehgal, Daniel Cernea, and Milena Makaveeva. Real-time scale invariant 3d range point cloud registration. *Image analysis and recognition*, pages 220–229, 2010.

[15] Andrea Frome, Daniel Huber, Ravi Kolluri, Thomas Bülow, and Jitendra Malik. Recognizing objects in range data using regional point descriptors. *Computer vision-ECCV 2004*, pages 224–237, 2004.

[16] Günter Hetzel, Bastian Leibe, Paul Levi, and Bernt Schiele. 3d object recognition from range images using local feature histograms. In *Computer Vision and Pattern Recognition, 2001. CVPR 2001. Proceedings of the 2001 IEEE Computer Society Conference on*, volume 2, pages II–II. IEEE, 2001.

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

[18] Chavdar Papazov and Darius Burschka. An efficient ransac for 3d object recognition in noisy and occluded scenes. In *Asian Conference on Computer Vision*, pages 135–148. Springer, 2010.

[19] Gregory C Sharp, Sang W Lee, and David K Wehe. Icp registration using invariant features. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 24(1):90–102, 2002.

[20] Phaneendra Vinukonda. A study of the scale-invariant feature transform on a parallel pipeline. *LSU Master's Theses*, 2721, 2011.

[21] Paul J Besl, Neil D McKay, et al. A method for registration of 3-d shapes. *IEEE Transactions on pattern analysis and machine intelligence*, 14(2):239–256, 1992.

[22] David A Simon, Martial Hebert, and Takeo Kanade. Techniques for fast and accurate intrasurgical registration. *Journal of image guided surgery*, 1(1):17–29, 1995.

[23] Timothée Jost and Heinz Hügli. Fast icp algorithms for shape registration. In *Joint Pattern Recognition Symposium*, pages 91–99. Springer, 2002.

[24] Rafael Beserra Gomes, Bruno Marques Ferreira da Silva, Lourena Karin de Medeiros Rocha, Rafael Vidal Aroca, Luiz Carlos Pacheco Rodrigues Velho, and

Luiz Marcos Garcia Gonçalves. Efficient 3d object recognition using foveated point clouds. *Computers & Graphics*, 37(5):496–508, 2013.

[25] Siheng Chen, Dong Tian, Chen Feng, Anthony Vetro, and Jelena Kovačević. Fast resampling of 3d point clouds via graphs. *arXiv preprint arXiv:1702.06397*, 2017.

[26] Jens-Malte Gottfried, Rahul Nair, Stephan Meister, Christoph S Garbe, and Daniel Kondermann. Time of flight motion compensation revisited. In *Image Processing (ICIP), 2014 IEEE International Conference on*, pages 5861–5865. IEEE, 2014.

[27] Oliver Wasenmüller and Didier Stricker. Comparison of kinect v1 and v2 depth images in terms of accuracy and precision. In *Asian Conference on Computer Vision*, pages 34–45. Springer, 2016.

[28] Shahram Izadi, David Kim, Otmar Hilliges, David Molyneaux, Richard Newcombe, Pushmeet Kohli, Jamie Shotton, Steve Hodges, Dustin Freeman, Andrew Davison, et al. Kinectfusion: real-time 3d reconstruction and interaction using a moving depth camera. In *Proceedings of the 24th annual ACM symposium on User interface software and technology*, pages 559–568. ACM, 2011.

[29] Radu Bogdan Rusu and Steve Cousins. 3d is here: Point cloud library (pcl). In *Robotics and automation (ICRA), 2011 IEEE International Conference on*, pages 1–4. IEEE, 2011.

[30] Daniel Girardeau-Montaut. Cloudcompare-open source project. *OpenSource Project*, 2011.

[31] Dirk Holz, Alexandru E Ichim, Federico Tombari, Radu B Rusu, and Sven Behnke. Registration with the point cloud library: A modular framework for aligning in 3-d. *IEEE Robotics & Automation Magazine*, 22(4):110–124, 2015.

[32] Leif Kobbelt and Mario Botsch. A survey of point-based techniques in computer graphics. *Computers & Graphics*, 28(6):801–814, 2004.

[33] Jelena Kovacevic, Vivek K Goyal, and Martin Vetterli. Fourier and wavelet signal processing. *Fourier and Wavelets. org*, pages 1–294, 2013.

[34] Radu Bogdan Rusu, Nico Blodow, and Michael Beetz. Fast point feature histograms (fpfh) for 3d registration. In *Robotics and Automation, 2009. ICRA'09. IEEE International Conference on*, pages 3212–3217. IEEE, 2009.

[35] Samuele Salti, Federico Tombari, and Luigi Di Stefano. Shot: Unique signatures of histograms for surface and texture description. *Computer Vision and Image Understanding*, 125:251–264, 2014.

[36] Federico Tombari, Samuele Salti, and Luigi Di Stefano. A combined texture-shape descriptor for enhanced 3d feature matching. In *Image Processing (ICIP), 2011 18th IEEE International Conference on*, pages 809–812. IEEE, 2011.

[37] John Ashburner and Karl J Friston. Rigid body registration. *Statistical parametric mapping: The analysis of functional brain images*, pages 49–62, 2007.

[38] Tadej Bajd, Matjaž Mihelj, and Marko Munih. *Introduction to robotics*. Springer Science & Business Media, 2013.

[39] Sunil Arya, David M Mount, Nathan S Netanyahu, Ruth Silverman, and Angela Y Wu. An optimal algorithm for approximate nearest neighbor searching fixed dimensions. *Journal of the ACM (JACM)*, 45(6):891–923, 1998.

[40] Alexandr Andoni and Piotr Indyk. Near-optimal hashing algorithms for approximate nearest neighbor in high dimensions. In *Foundations of Computer Science, 2006. FOCS'06. 47th Annual IEEE Symposium on*, pages 459–468. IEEE, 2006.

[41] Saurabh Gupta, Ross Girshick, Pablo Arbeláez, and Jitendra Malik. Learning rich features from rgb-d images for object detection and segmentation. In *European Conference on Computer Vision*, pages 345–360. Springer, 2014.

[42] Tim Foley and Jeremy Sugerman. Kd-tree acceleration structures for a gpu raytracer. In *Proceedings of the ACM SIGGRAPH/EUROGRAPHICS conference on Graphics hardware*, pages 15–22. ACM, 2005.

[43] Linjia Hu and Saeid Nooshabadi. Massive parallelization of approximate nearest neighbor search on kd-tree for high-dimensional image descriptor matching. *Journal of Visual Communication and Image Representation*, 44:106–115, 2017.

[44] F Al-Osaimi, Mohammed Bennamoun, and Ajmal Mian. An expression deformation approach to non-rigid 3d face recognition. *International Journal of Computer Vision*, 81(3):302–316, 2009.

[45] Dirk Smeets, Jeroen Hermans, Dirk Vandermeulen, and Paul Suetens. Isometric deformation invariant 3d shape recognition. *Pattern Recognition*, 45(7):2817–2831, 2012.

[46] Quoc V Le. Building high-level features using large scale unsupervised learning. In *Acoustics, Speech and Signal Processing (ICASSP), 2013 IEEE International Conference on*, pages 8595–8598. IEEE, 2013.

[47] Jane Bromley, Isabelle Guyon, Yann LeCun, Eduard Säckinger, and Roopak Shah. Signature verification using a" siamese" time delay neural network. In *Advances in Neural Information Processing Systems*, pages 737–744, 1994.

[48] Gregory Koch, Richard Zemel, and Ruslan Salakhutdinov. Siamese neural networks for one-shot image recognition. In *ICML Deep Learning Workshop*, volume 2, 2015.

[49] Yaniv Taigman, Ming Yang, Marc'Aurelio Ranzato, and Lior Wolf. Deepface: Closing

the gap to human-level performance in face verification. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1701–1708, 2014.

[50] Sergey Zagoruyko and Nikos Komodakis. Learning to compare image patches via convolutional neural networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 4353–4361, 2015.