Implementation of a 3D ICP-based Scan Matcher

Cai, Zhongjie **Chao, Shou-Yu**
Department of Computer Science, University of Freiburg

Abstract

In this paper we shall present our implementation of a 3D ICP Scan Matcher using point clouds data collected by SICK 3D scanner around the FAW campus. We give a brief overview of ICP algorithm as well as its underlying technique for matrix manipulation. Additionally, we shall present the general idea during each stages of matching process and library used for visualizing the matching result. In the last section of the paper we demonstrate the precision of the matcher by comparing the transformation matrix generated from the ICP matcher with the robot's internal movement sensor. Our implementation shall be capable to generate large 3D map based on hundreds of point clouds seamlessly.

1 Introduction

The Iterative Closest Point algorithm was first introduced by [Besl and McKay, 1992]. It is a general purpose, representation independent, shape based algorithm intended to match various geometrical primitives including point sets, line segment sets, triangle sets and parametric curves and surfaces. The algorithm can be used with a variety of geometrical primitives, including point sets, line segment sets [Fitzpatrick and Sonka, 2000]. It is suitable for aligning a very simple 2D shape to a large, complex 3D surface reconstruction in a real-time basis. Currently, the algorithm has become the principle technique for registration of 3D data sets and widely adopted in various industry including but not limited to medical and military scope. Our goal is to implement a point level ICP matcher which allows input from two VRML files containing point clouds generated by the SICK laser range scanner mounted on a robot. The matcher will attempt to match two point clouds and calculate a best possible transformation and rotation matrix using ICP algorithm. After that, the matcher will generate a new point cloud which contains the first cloud and the transformed second cloud. A high performance visualization library from SoWin, a part of Coin3D project with OpenGL support, will render the final result with convenient user interface.

This paper is organized as follows. After a brief overview of our system, we briefly present the internal structure of the VRML point cloud file. In the following section we shall discussing the general concepts of an ICP algorithm. After that, we separate our implementation of the ICP algorithm into several different phases and we shall discuss each phase accordingly. Additionally, we enumerate various visualizing user interface library and present our reason for choosing SoWin. We will also compare the transformation matrix from the ICP algorithm with the actual robot movement to demonstrate the precision of our work. Finally, we evaluate the performance of our implementation and conclude our work.

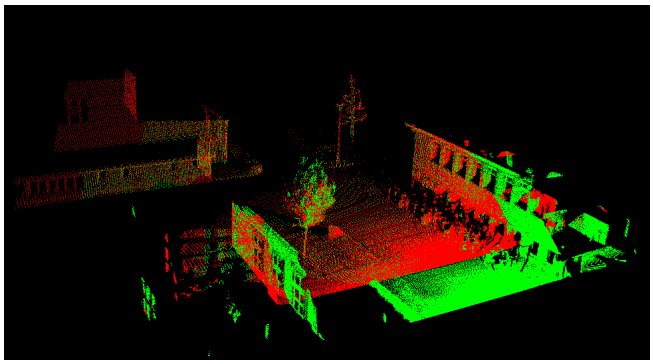

Figure 1: Two point clouds aligned using ICP algorithm

2 Implementation

2.1 System Overview

Our implementation of ICP matcher is written completely in C++. There is several consideration to make this decision. The main concern is performance issue. Another one is the most widely used approximate nearest neighbor library 'ANN' is also written in C++. Although Java could offer platform independent flexibility while accessing native library written in C++ via JNI interface. However, due to complexity involved in using two language at the same time, we decided to stay with C++.

Our program was separated into two parts. The first part loads the VRML file containing point clouds and calculation the rotation and transformation matrix based on ICP algorithm. The output could be either transformation matrix only or a newly generated file containing the matched point cloud

sets. This file could be plotted by GNU Plot as well as visualization interface in the second part of our program. This part of code is platform-independent and has been tested under both Linux and Windows environment.

The second part of code virtualizes the point cloud by using SoWin library from Coin3D project. It provide a convenient user interface for user to see the matched result from two point clouds. The library is written by MFC and is currently Windows only.

2.2 Interpreting point clouds stored in VRML format

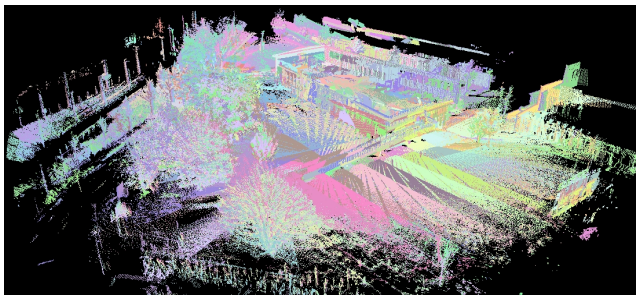

Figure 2: ICP registration of all VRML files

We received 130 specially formatted VRML files from our instructor. These files contain point clouds from a SICK 3D laser range scanner in figure 3 mounted on a robot. By rendering all VRML files together in figure 2, the robot appear to be scanning certain buildings and offices around the Faculty of Engineering campus of the University of Freiburg in a predefined path. Each file contains around 150,000 to 200,000 individual points. The files' header also contains a estimated rotation and translation by the robot's internal movement sensor, which is essential before the ICP algorithm is applied.

Figure 3: A SICK LMS-200 laser range scanner

Our program will first attempt to read the estimated rotation between the current points cloud and next points cloud from the VRML header. The rotation is stored in coordinate axis format. This value shall be useful for the ICP algorithm to limit maximum rotation between two points. Therefore, it is necessary for us to use the following decomposition equation for converting into matrix representation.

$$Q_u(\theta) =$$

$$\begin{bmatrix} x^2(1-c_\theta)+c_\theta & xy(1-c_\theta)-zs_\theta & xz(1-c_\theta)+ys_\theta \\ xy(1-c_\theta)+zs_\theta & y^2(1-c_\theta)+c_\theta & yz(1-c_\theta)-xs_\theta \\ xz(1-c_\theta)-ys_\theta & yz(1-c_\theta)+xs_\theta & z^2(1-c_\theta)+c_\theta \end{bmatrix}$$

The translation field in the header also indicated the estimated translation between two clouds. There is no need to convert this value for future usage.

Following the header, the VRML contains large amount of points stored in *(x,y,z)* format. Each value of the axis is saved in double format which require 8 bytes to store. In order to store a point in 3D format, 24 bytes will be needed. The point clouds we received usually contain approximately a hundred thousand points. Due to large amounts of data, an data structure with exceptional memory management feature is required for faster ICP calculation. Therefore, we use the map data structure in C++ to store the points, which allows us to write the code in a more readable manner.

By collecting estimated rotation, translate and the points from two different VRML file, we are now having enough data to perform the ICP scan matching.

2.3 Matching point clouds using iterative closest point algorithm

In the previous section we collected all required information from two point clouds. We are now preparing to match them according to the ICP algorithm. We separated our implementation of ICP algorithm into several phases. We shall now discuss each phase individually.

Phase 1: Find closest points for each point in the base cloud with the target cloud

This phase is the most important step in the ICP algorithm. For each point in the base cloud, we attempt to choose a point in the target cloud which has the minimum Euclidean distance, as in figure 4. Mathematically speaking, we want to minimize:

$$\frac{1}{|M|}\sum_{v\in M}\|v - match_v(p)\|_2^2$$

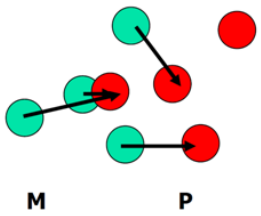

Figure 4: Matching closest points from base cloud to target cloud

Originally, we were using brute-force approach with two levels for loop to match two point clouds. Although we could filter most points which are obviously not matchable, this approach still has a running time of $O(n^2)$. Assume that each

VRML point cloud contains 200,000 points, a matching process will need to do 40 billion time of Euclidean distance calculation, which is clearly unacceptable in a production environment.

Our instructor suggested us that a well-known open source library, namely ANN, are specially designed to solve the Approximate Nearest Neighbor problem. The library use a variant of KD-tree which could archive significantly faster running time on the order of 10's to 100's with relatively low error rate. It also allows user to specify a maximum approximation error bond, thus allowing the user to control the tradeoff between accuracy and running time [Mount, 2010].

After adopting the library, we see a significant performance incensement with lesser memory consumption. We were also able to simplify our code for this phase into 50 lines. The library also allows user to assign a maximum allowed distance between the base cloud and the target cloud. This significantly help us to reduce the noise in the result with match that obviously not possible.

Phase 2: Finding the geometric centroid of accepted matches

In this phase we shall need to find out the geometric centroid of the matched base cloud and target cloud from previous phase, respectively. This step is relatively trivial. We will need to sum up all the matched point in each clouds and divide it by the total count of point as the following equation:

$$\mu_x = \frac{1}{N_x}\sum_{i=1}^{N_x} x_i \text{ and } \mu_p = \frac{1}{N_p}\sum_{i=1}^{N_p} p_i$$

Which *x* is the base cloud and *p* is the target cloud. After that we subtract the centroid from the matched point cloud.

$$X' = \{x_i - \mu_x\} = \{x'_i\}$$

$$P' = \{p_i - \mu_p\} = \{p'_i\}$$

We now shifted the center of mass of the point clouds to the origin of the coordinate system.

Notice that all operation above are matrix operations.

Phase 3: Applying Singular Value Decomposition

In this phase we will calculate the rotation matrix based on the point pair match results from ANN. There are exist several possible closed form solutions for rigid body transformation:

- SVD [Besl and McKay, 1992]
- Quaternions [Horn, 1987]
- Orthonoraml matrices [Horn *et al.*, 1988]
- Dual quaternions [Walker *et al.*, 1991]

A paper first published by [Arun *et al.*, 1987] suggested that it is more robust and easier to implement based on singular value decomposition (SVD) approach. SVD denote that given A, a m by n matrix, we could decompose it and find out a matrix *U*, a m by m matrix, Σ, a m by n diagonal matrix with nonnegative real numbers on the diagonal, and V^T, an n by n unitary matrix. The diagonal entries of Σ are known as the singular values of *A* and the result are not unique [McMahon, 2006]. Figure 5 illustrated the concept of the SVD. Based on

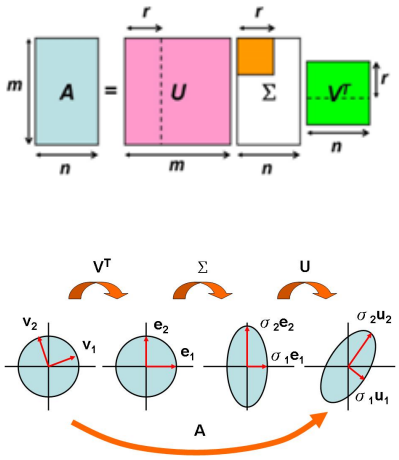

Figure 5: Concept of SVD

the SVD algorithm, we will be able to find out the rotation matrix R, where $R = UV^T$.

Due to the complexity of SVD algorithm, we use a public domain library which have an $O(mn^2)$ running time. We will first need to build a 3-dimentianal matrix for SVD. The matrix consist of the sum of all multiplication of matched base point and target point pair, where all point was adjusted by their geometric center. The following pseudo code shows how the matrix was built:

```
  FOR ALL matched point pair AS i:
A += (#i point in base cloud - geometric
center of base cloud)*(#i point in
target cloud - geometric center of
target cloud)
```

By sending the A matrix for decomposition, we will able to find out U and V^T. We will need these matrixes to compute the rotation matrix.

Phase 4: Finding out rotation and translation based on SVD result

Based on the proposal from [Besl and McKay, 1992], the rotation matrix could be found by:

$$R = UV^T$$

Where R will be a 3 by 3 matrix in our case.

To find out the translation matrix, we use the following equation:

$$T = B_c - T_c * R$$

Where B_c and T_c is the matrix representation of the geometric center of base cloud and target cloud, respectively. T will be a 1 by 3 matrix in our case.

Phase 5: Calculate new average distances using the rotation and translation matrix

In this phase we are going to calculate a new average distance between base cloud and target cloud by applying the rotation

and translation matrix to the target cloud. The new target cloud could be calculated by following formula:

$$TPC_n = R * TPC_o + T$$

Where TPC_n is the matrix representation of new target point cloud and TPC_o is the old target point cloud.

Now we shall sum up all Euclidean distance of the matched point pairs of base cloud and transformed target cloud and find out their average.

$$\frac{1}{N}\sum_{i=1}^{N}|BPC_i - TPCn_i|$$

Our experiment shows that if average distance is smaller than 0.0012, two point cloud will shows a reasonable match and we could stop the iterative process. Otherwise we will go back to Phase 1 if the loop counter is less than 20. If the loop counter equals to 20, the iterative process will also stop and the corresponding rotation matrix and translate matrix will be returned to the main program. The maximum number of iterative is adjustable. Therefore, if the user desire a higher precision, he could elect to run the iterative process as much as he likes. However, Figure 6 show that after 20 time of iterative process, the improvement were almost unobservable.

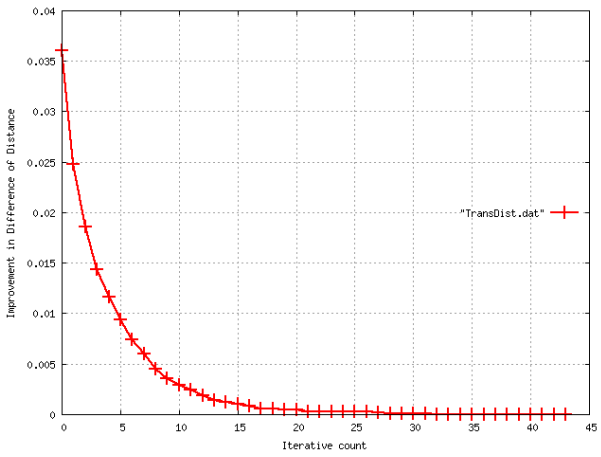

Figure 6: Improvement in each iterative step

3 Point clouds virtualization

In the previous section we calculated the transformation and rotation matrix by ICP algorithm. This information should be enough for any further usage by other programs. However, we believe that creating a user interface to visualize the result is vital for demonstrating the correctness of our project as well as helping us to avoid potential errors and bugs during development stage.

3.1 GNUPlot

The first visualizing interface which comes up to our mind is GNUPlot. It is relatively simple for us to plot the result in GNUPlot. All we need to do is to generate a script containing the command and all points in the base cloud and the transformed target cloud. However, the newly generated files could be as large as 10MB which contains more than 250,000 points. It is extremely resource intensive for a 2D X11 software without graphical driver support. A render, rotating, zooming could take as long as 10 second, which is obviously not very user friendly. Figure 7 shows a render sample from GNUPlot.

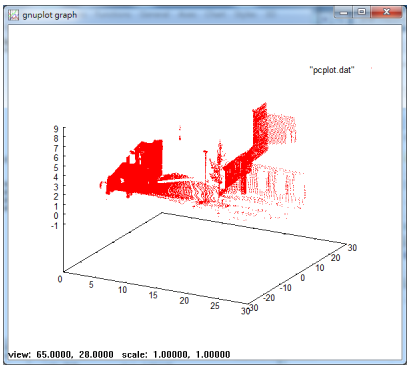

Figure 7: pcloud000.wrl plotted by GNUPlot

3.2 SoWin from Coin3D graphics developer kit

At the beginning we were planning to let our program platform independent. However, being platform independent means a big trade off between performance due to the fundamental different among different operation system and their underlying API. However, we do found a company which offers a OpenGL based virtualization library namely SoWin, which is licensed under GPL. The SoWin library is a C++ GUI toolkit that binds together the Coin rendering library with the user interface parts of the Win32 API. It also offer variants such as SoQt and SoXt which runs under QT and X Windows respectively. Except the OS dependent part, all of them share similar interface after they were initialed. Currently, due to the driver limit of our hardware, we use SoWin to render the point clouds. Figure 8 shows a screenshot of SoWin.

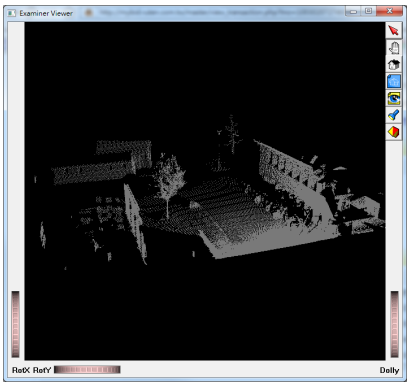

Figure 8: pcloud000.wrl plotted by SoWin

4 Issues during implementation

During the implementation stage we experience several issues. We shall now enumerate these issues and discuss their solution accordingly.

4.1 ICP algorithm stuck in local optimize dilemma

In our earlier version of code, we did not took the initial transition stated in the header of VRML files into consideration. However, ANN library were still able to find large amount of point match and the ICP algorithm could still find a rotation and translation matrix. After rendering two point clouds on the SoWin, we found out that they did not overlap correctly. Two clouds were matched at the standpoint of the robot in two clouds instead of the correct match, as in figure 9. Initially,

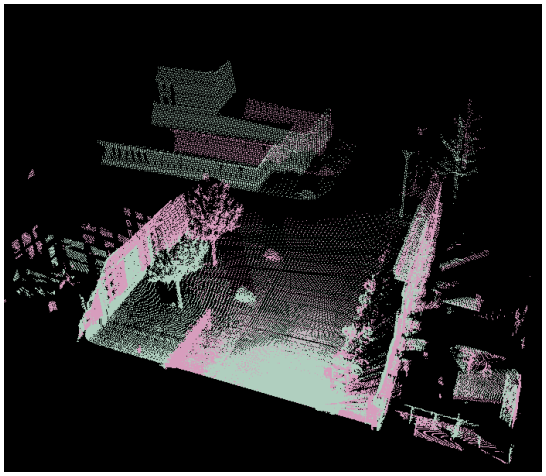

Figure 9: Using ICP without applying initial estimation

we thought that ICP is an iterative algorithm. Therefore, we attempt to run the ICP algorithm for few hundred times on the same cloud sets to see if there is any improvement. Unfortunately, this attempt was not success and the clouds did not change significantly.

We moved our focus to the point clouds gathered from the SICK sensor. We observed that the terrain around 5 meters of reach of the robot has much higher point destiny and it decays exponentially. For example, a tree 20m away, which should very helpful for ICP algorithm, got merely 200 hundreds point but the floor, which looks the same everywhere and not helpful for alignment process, has more than 10,000 points. It sounds reasonable that all floor point surrounding the robot got the 'vote' because it is simply too much of them. Therefore, we try to implement an algorithm to normalize the density of point clouds.

Our first attempt to normalizing the point clouds is to create a lot of virtual three dimensional square cube. These square cube could have a size of $7cm^3$. In each cube, we calculate the count of points. And for each cube we assign a score from 0 to 1 and we calculate the score by following formula:

$$S = \sum_{i=1}^{n} \frac{1}{2^n}$$

This imply if a cube contains 2 points, the first point will get 0.5 point, the second point will get 0.25 and the total would be 0.75 point. By using this filter, no matter how much point does the cube contains, each cube could only get as much as $0.\overline{9}$ point, which is only twice of the cube with one point. By using the filter above, our graph is reasonable normalized. However, this attempt was not successful. The result still looks the same as before.

We now suspect that we did not understand the ICP algorithm correctly. After reading papers from [Rusinkiewicz and Levoy, 2001] and [Madhavan and Messina, 2003], we found out that ICP is an algorithm intended to calculate local optimum instead of global optimum. Hence, it is essential to know the estimated transformation before the ICP took place.

Estimated transformation is a rotation and translation matrix, which can be either calculated from the GPS, internal movement sensor or assigned by human [Madhavan and Messina, 2003]. The estimated transformation must provide a reasonable match between two point clouds and ICP algorithm is intended to fix the minor error from the sensors.

After applying the estimated transformation in phase 1 of previous section, we are now able to calculate ICP correctly.

4.2 Accuracy

In figure 10 we demonstrate the accuracy of our implementation. We apply ICP algorithm to all 130 VRML file and we compare their initial estimation with their corresponding translation matrix from the ICP result. The figure indicated

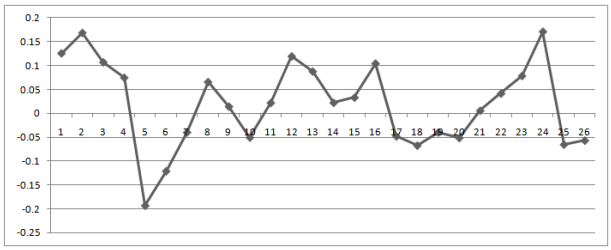

Figure 10: Euclidean distance difference between estimated translation and ICP translation matrix

that the ICP algorithm only apply minor transformation to the initial transformation. Which also implies that the initial transformation is relatively accurate.

4.3 Performance issue under different platform

[Chetverikov *et al.*, 2002] proposed that ICP is intended for real-time alignment purpose. Hence, it is crucial that our algorithm runs fast enough for application such as real-time terrain matching. In earlier section we mentioned that we separate our program into two parts, ICP calculation and virtualization. The ICP calculation code is platform-independent. We observe an interesting fact that the code runs much faster under Linux than in Windows using similar hardware configuration. Table 1 shows a comparison under both Linux and Windows.

We separate the ANN run time from total run time in the comparison table above. Most phases, expect the approximate neighbor finding phases, have $O(1)$ or $O(n)$ runtime.

OS	Total run time	ANN	Other code
Windows	$1265.132ms$	$1100.756ms$	$164.376ms$
Linux	$603.925ms$	$548.639ms$	$55.286ms$

Table 1: Matching pc_vrml000.rml and pc_vrml001.rml

We observed that the running time were mostly spent on ANN. But one unusual fact is that it runs almost two times faster under Linux than under Windows, while the compiler optimization level remains the same. Our assumption is that ANN might be specially designed for Linux and ported later to Windows. Two operation system has different ways to handle memory management. Using nonnative Linux style memory management might be slower under Windows.

Because our code heavily relies on 8 bytes floating point operation of the CPU. Compiling using SSE2, SSE3 optimizing or compiling under x64 environment might also benefit the execution performance.

4.4 Code complexity

C++ does not define a standardized representation of matrix data structure. Therefore, different library define their own ways to store the matrix and they are usually not interchangeable. A significant portion of our code were working on matrix related operation such as converting from one representation to another. We also wrote the code for addition, multiply, subtraction, dividing, rotation and translate of matrix by ourselves. The code could be highly simplified and more readable if it was written by mathematical scripting languages such as *Matlab*, which has all operation above built-in. However, due to the nature of scripting language, the performance would be a big issue.

5 Conclusion and Outlook

Our implementation has demonstrated that the ICP algorithm can correctly estimated the rotation and translation under the condition that initial estimation is previously known and reasonable aligned. However, in many real life situation, it is not always possible. Consider a laser range scanner installed above a Google Street View car. The car has to drive into small streets and narrow lanes to scan entire city. Driving in constant speed or relative stable speed is not possible. Moreover, the internal speed meter and GPS on board might not always be accurate. How do we align the graph under these cases? The paper proposed by [Rusinkiewicz *et al.*, 2002] and [Turk and Levoy, 1994] provided a feasible alternative to the ICP algorithm.

Currently, all SICK laser range scanner are point based. Our implementation also focused mainly on point level matching. Nevertheless, [Besl and McKay, 1992] indicated that ICP could be also applied to line segment, triangle, curves and surface as well. We believe that this would be worth for further discussion.

References

[Arun *et al.*, 1987] K. S. Arun, T. S. Huang, and S. D. Blostein. Least-squares fitting of two 3-d point sets. *IEEE Trans. Pattern Anal. Mach. Intell.*, 9(5):698–700, 1987.

[Besl and McKay, 1992] Paul J. Besl and Neil D. McKay. A method for registration of 3-d shapes. *IEEE Trans. Pattern Anal. Mach. Intell.*, 14(2):239–256, 1992.

[Chetverikov *et al.*, 2002] D. Chetverikov, D. Svirko, D. Stepanov, and Pavel Krsek. The trimmed iterative closest point algorithm. In *In International Conference on Pattern Recognition*, pages 545–548, 2002.

[Fitzpatrick and Sonka, 2000] J. M. Fitzpatrick and M. Sonka. *Handbook of Medical Imaging, Volume 2. Medical Image Processing and Analysis (SPIE Press Monograph Vol. PM80)*. SPIE–The International Society for Optical Engineering, 1 edition, 2000.

[Horn *et al.*, 1988] Berthold K. P. Horn, H.M. Hilden, and Shariar Negahdaripour. Closed-form solution of absolute orientation using orthonormal matrices. *JOURNAL OF THE OPTICAL SOCIETY AMERICA*, 5(7):1127–1135, 1988.

[Horn, 1987] Berthold K. P. Horn. Closed-form solution of absolute orientation using unit quaternions. *Journal of the Optical Society of America A*, 4(4):629–642, 1987.

[Madhavan and Messina, 2003] R. Madhavan and E. Messina. Iterative registration of 3d ladar data for autonomous navigation. In *In Proceedings of the IEEE Intelligent Vehicles Symp*, pages 186–191, 2003.

[McMahon, 2006] David McMahon. *Linear Algebra Demystified. A Self-Teaching Guide*. Demystified Series. McGraw-Hill, New York, NY, 2006.

[Mount, 2010] David M. Mount. *ANN Programming Manual*. Department of Computer Science and Institute for Advanced Computer Studies, University of Maryland, College Park, Maryland, 2010.

[Rusinkiewicz and Levoy, 2001] Szymon Rusinkiewicz and Marc Levoy. Efficient variants of the icp algorithm. In *INTERNATIONAL CONFERENCE ON 3-D DIGITAL IMAGING AND MODELING*, 2001.

[Rusinkiewicz *et al.*, 2002] Szymon Rusinkiewicz, Olaf Hall-Holt, and Marc Levoy. Real-time 3d model acquisition, 2002.

[Turk and Levoy, 1994] Greg Turk and Marc Levoy. Zippered polygon meshes from range images. In *SIGGRAPH '94: Proceedings of the 21st annual conference on Computer graphics and interactive techniques*, pages 311–318, New York, NY, USA, 1994. ACM.

[Walker *et al.*, 1991] Michael W. Walker, Lejun Shao, and Richard A. Volz. Estimating 3-d location parameters using dual number quaternions. *CVGIP: Image Underst.*, 54(3):358–367, 1991.