Implementation Details of Surface Normal Sensing with ToF-Array and PCA

Ethan Chun[1]

I. Introduction

When performing non-planar grasping, the current reflexive controller struggled to differentiate target objects from background materials and struggled to achieve antipodal grasping. To remedy this, we begin development on a new type of sensor, capable of detecting the surface normal of any target it is pointed at a high update rate. We base our platform on the VL53L8CX 8x8 Time of Flight sensor array and the Nucleo F411RE MCU. We use principal component analysis to extract surface normals from the distance data. In preliminary testing, we show accurate perception of surface normals at an update rate of 30 Hz.

II. Algorithm

A. High Level Overview

Our goal is to find the normal of the surface directly in front of the sensor. We approach this by sampling a collection of points on the surface, then fitting a plane to them. Refering to the chart in Fig 1, we first convert the ToF sensor's raw distance values into points in the coordinate frame of the sensor. Then, we use principal component analysis to fit a plane.

B. Data Inputs

The time of flight sensor array outputs a list of either $4 \times 4 = 16$ or $8 \times 8 = 64$ distance values in mm, corresponding to each of the zones on the ToF array. We set the ToF sensor to output the 4×4 output as this is significantly faster. These can be called using `VL53L8A1_RANGING_SENSOR_GetDistance(...);`.

C. Distance to Coordinates

Idealized Algorithm. Based on the spec sheet, the sensor has a view frustum with an angle of 45 degrees between opposing sides. We assume that the sensing rays corresponding to each of the zones are evenly spaced between these ranges such that their effective angles from the middle are $\{-22.5, -7.5, 7.5, 22.5\}$ degrees. *Note, this is a pretty large assumption and empirical testing shows that it might not be entirely correct. Future work may calibrate this in to determine the exact angles and directions of the zones.* Given these zones, we pre-compute a unit vector pointing from the origin along each of the rays. This can be done at startup or stored in a lookup table. Finally, each time we pull from the sensor, we multiply the corresponding ray with the sensing distance to extract the point on the surface that the ray intersected. Following this algorithm, we end up with an array of 16 coordinates, each with three components.

Implementation Notes. Sometimes when looking at a target, rays will miss the target altogether and intersect something much further away from the sensor. We filter these by adding a threshold to the distance value. If the value exceed the value, we do not add the coordinate to the coordinate array. Since this leads to a variable number of coordinates, we keep track of the number of coordinates, n, and use that in the subsequent steps.

D. Principal Component Analysis

Given a collection of points in $\mathbb{R}^3$, we want to find the normal of the surface corresponding to these points. Due to the presence of noise, it is not sufficient to just compute the cross product between some number of points. A more robust solution is to fit a plane to the points such that the plane most closely aligns with the measured surface. There are many ways to do so (read `https://math.stackexchange.com/questions/99299/best-fitting-plane-given-a-set-of-points`), however, an easy to implement and robust solution is to mean center the points, compute PCA, and use the normal of the plane defined by the two largest principal components. This is the smallest principal component.

Intuition. For a mean-centered collection of points in 3D space that somewhat resemble a plane, one would expect the location of the points to change the most along some direction in the plane. Therefore, we expect a unit vector representing the direction of largest variance to also lie on the plane. Since a plane can be defined by two orthogonal vectors, we expect the unit vector along the next highest direction of variance that is orthogonal to the first vector to also lie on the plane. These are exactly the first two principal components. Finally, we know that the third principal component must be orthogonal to the first two. Therefore, the third and smallest principal component is the normal (or inverse normal) of the surface.

Definition. Given a set of data X defined on vector space V, PCA is a transformation $f : V \to V'$ where the first coordinate of V' is along the direction of highest variance in data X. The second coordinate of V' is along the direction of next highest variance, and so on. This can either involve dimensionality reduction where $\dim(V') < \dim(V)$ or not, where $\dim(V') = \dim(V)$. The principal components are the unit vectors in V along the directions of highest variance.

[1]Author is with the Biomimetic Robotics Laboratory at the Department of Mechanical Engineering, Massachusetts Institute of Technology (MIT), Cambridge, MA, 02139, USA.

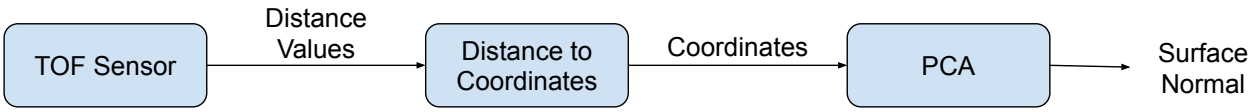

Fig. 1. **Normal Sensor Data Flow** – We process the raw ToF distance values to points in 3D space, then compute pca to find the surface normal.

Given a set of data X of shape $n \times 3$, it is known that the principal components of X are the eigenvectors of the covariance matrix of the data, $X^T X$. The first principal component is the eigenvector with the highest associated eigenvalue and so on.

For more details, I highly recommend this chapter by Dr. Richard Wilkinson `https://rich-d-wilkinson.github.io/MATH3030/4.1-pca-an-informal-introduction.html`.

Implementation Details. We must compute eigenvectors numerically. This is non-trivial. Two helpful resources are the Handbook for Automatic Computation by J. H. Wilkinson, C. Reinsch [1] and Matrix Computations by G Golub and C Van Loan [2].

We opt for the Jacobi method to compute eigenvalues. The following is summarized from Golub and Van Loan's Matrix Computations [2].

The algorithm is defined as:

Given a symmetric $A \in \mathbb{R}^{n \times n}$ and a tolerance tol, overwrite A with $V^T AV$.

$V = I_n$, $\delta = tol \cdot ||A||_F$

while off $(A) > \delta$ **do**

Choose (p,q) so $|a_{pq}| = max_{i \neq j} |a_{ij}|$

[c, s] = symSchur2(A, p, q)

$A = J(p,q,\theta)^T A J(p,q,\theta)$

$V = VJ(p,q,\theta)$

end while

Here, symSchur2 is a 2 x 2 Symmetric Schur Decomposition. The idea is to find a pair of indices with the largest off-diagonal terms, then apply the Symmetric Schur Decomposition to set the off diagonal terms to zero. Continually applying this eventually diagonalizes the original matrix

We use an implementation by John Burkardt at Florida State University `https://people.sc.fsu.edu/~jburkardt/c_src/jacobi_eigenvalue/jacobi_eigenvalue.html`.

III. Hardware

A. Time of Flight Configuration

We use the X-Nucleo-53L8A1 shield, a Dynamixel Shield, and a Nucleo-F411RE to test the array. To interface with the Dynamixels, a number of configuration changes were made. These are:

- Set i2c to "fast mode" and set baudrate to 400000.
- Change Dynamixel usart port to use one other than the default on the motor shield. This prevents interference with the printf commands.

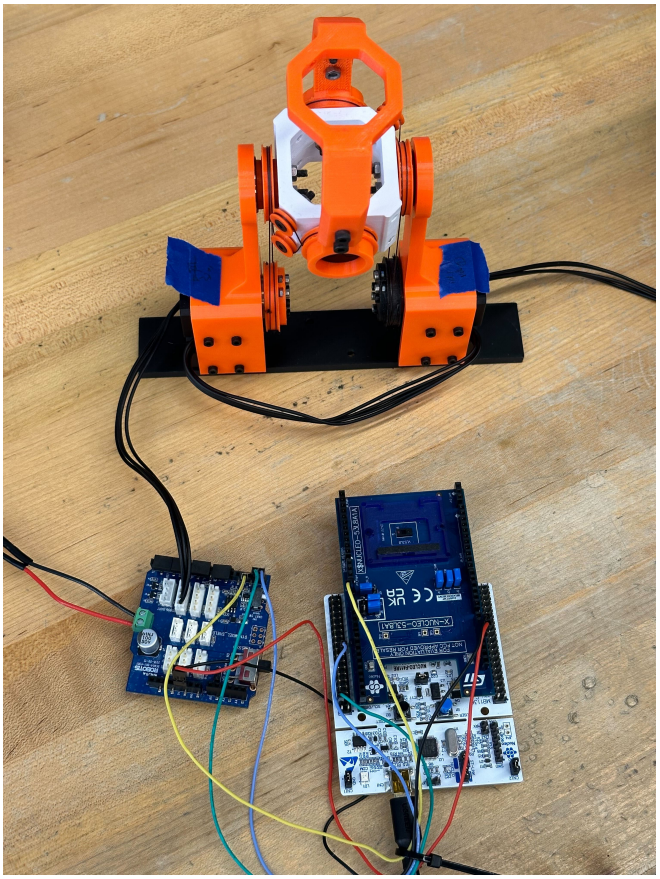

Fig. 2. **Hardware Setup** – We have four components: the wrist (top), the motor dynamixel power board (bottom left), the ToF shield (bottom right blue), and the STM32 (white).

- Connect power to the dynamixel shield (it needs both a 12v and 5v supply).
- Update clock settings in stmcube to the max settings. This allows for the full 4Mb connection speed to the dynamixels.
- Disable interrupts. i2c does not function well with timer interrupts enabled.

Connecting the smaller Satel-VL53L8A breakout board was also tested, however communication between the stm32 and the sensor was not robust. We think that the long i2c lines created room for noise.

IV. Demo

We produce a demonstration where a user places an object in front of the ToF sensor and the wrist actuators move to point in the direction of the surface normal of the object.

We also attempted a demonstration where the sensor was mounted to the wrist. But, control with the sensor moving proved difficult. With substantially more effort, this would likely be feasible.

V. Conclusion

We use a VL53L8CX 8x8 Time of Flight sensor array to compute the surface normal of objects placed within view of the sensor. To visualize this, we control a 2 DoF wrist to point in the direction of the normal. We propose the use of PCA as an elegant and robust means to predict the normal. This demonstration demonstrates the feasibility of using sensor arrays for surface normal detection. Future work may incorporate these sensors onto the robotic hand for normal sensing of target objects to grasp.

References

[1] A.S., J. Wilkinson, and C. Reinsch, *Handbook for Automatic Computation: Volume II: Linear Algebra* (Grundlehren der mathematischen Wissenschaften). Springer Berlin Heidelberg, 1971, ISBN: 9783642869402. [Online]. Available: https://link.springer.com/book/10.1007/978-3-642-86940-2#affiliations.

[2] G. Golub and C. Van Loan, *Matrix Computations* (Johns Hopkins Studies in Atlantic History & Culture). Johns Hopkins University Press, 1983, ISBN: 9780801830105. [Online]. Available: https://math.ecnu.edu.cn/~jypan/Teaching/books/2013%20Matrix%20Computations%204th.pdf.