Geometric Image Processing Lab

CT Registration using HoloLens 2

By: Nir Mualem, Oren Shavit

Supervisors: Amit Bracha, Ron Slossberg, Yaron Honen

Contents

The Problem

During a sinus surgery, the surgeon needs to see the patient’s CT scan.
To make it more convenient, the CT scan will be displayed as an STL 3D file in the correct location of the patient’s face using AR with Microsoft HoloLens 2.
The surgeon will be able to see the CT scan aligned to the patient's face while looking at the patient during the operation.

Suggested Solution

Extracting the patient’s face while the surgeon is looking at him in real time stream using the HoloLens sensors. Doing registration between the CT scan and the real time face and visualizing the results on the HoloLens so the surgeon will be able to see it during the operation.

General Architecture Explanation

Our design is constructed from 4 main parts:

A. Sensors to real time point cloud
   i. Sending the input data (as detailed in next section) to the remote server using a TCP communication.
   ii. At the remote server, using the HoloLens' input data, constructing a point cloud.
   iii. Cropping the point cloud to consist only the patient's face.

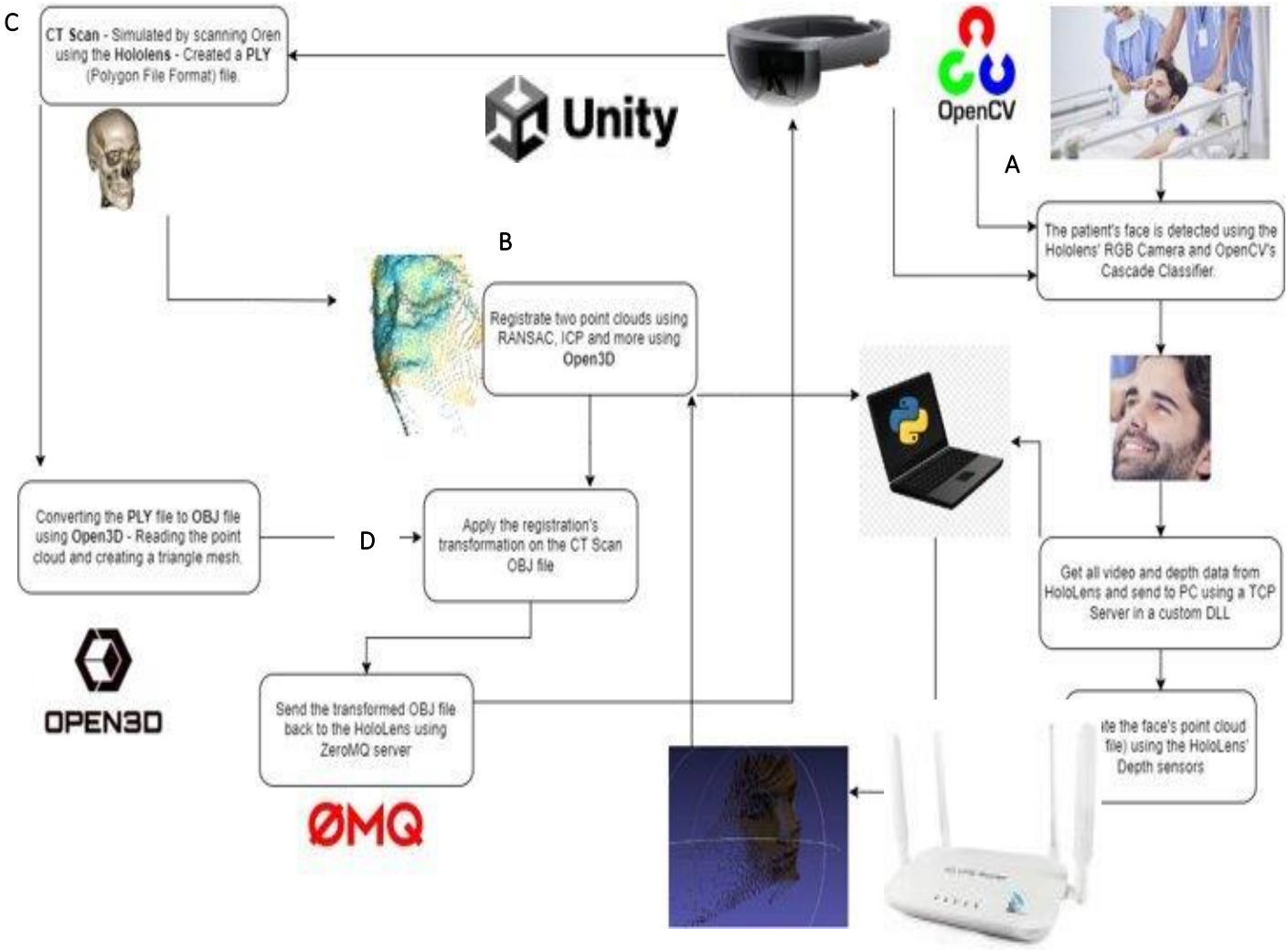
B. Registration block
   i. Getting the CT scan's point cloud and the face real time point cloud as inputs to the block.
   ii. Registration is done using several methods which are explained in detail at the Registration section.
   iii. Getting the transformation matrix as output.

C. CT scan to triangle mesh (simulated by face scan)
   This part is only for creating facial data that is replacing the role of the CT scan, due to lack of real CT scan data (medical confidentiality reasons).
   i. Creating a point cloud of a face scan (an old scan using the HoloLens, not real time), same way as explained in part A.
   ii. Using the point cloud, we convert it to a triangle mesh for visualization at the device (the Unity app can't visualize only point cloud without meshing).
   The conversion is done by Poisson Surface Reconstruction algorithm (explained in the Surface Reconstruction section).

D. Transforming the CT scan using the transformation matrix from part B and sending it back to the HoloLens for visualization aligned to the real time face.
   Sending back to the device is done by ZeroMQ server (external library based on TCP connection).

- Parts A, B and D are occurring every 10-15 seconds for keeping the registration up to date. We must keep it updated because of the computation and drifting errors that occur due to the communication latencies and the movement of the device in the world.
- Part C is only for simulating the CT scan, this should be replaced by a real 3D CT scan.

General Flow Chart



The HoloLens Input Data – To a Face Point Cloud

Using HoloLens' Research Mode, we were able to use the device's sensors and cameras to scan the patient's face and extract a point cloud of it.

For that we used:

RGB Part

1. The camera's intrinsic matrix allows us to transform the 3D coordinates to 2D coordinates on an image plane using the pinhole camera model. The values f_x and f_y are the pixel focal length and are identical for square pixels. The values o_x and o_y are the offsets of the principal point from the top-left corner of the image frame. The values are expressed in pixels.
$$K = \begin{bmatrix} f_x & 0 & o_x \\ 0 & f_y & o_y \\ 0 & 0 & 1 \end{bmatrix}$$
2. The camera's extrinsic matrix - the extrinsic parameters denote the coordinate system transformations from 3D world coordinates to 3D camera coordinates.
   a. The extrinsic matrix consists of a unitless rotation matrix (R) on the left and a column vector translation (t) on the right.

$$[R \mid t] = \left[\begin{array}{ccc|c} r_{1,1} & r_{1,2} & r_{1,3} & t_1 \\ r_{2,1} & r_{2,2} & r_{2,3} & t_2 \\ r_{3,1} & r_{3,2} & r_{3,3} & t_3 \end{array}\right]$$

3. The PV camera's (Photo Video) frames and timestamps.
4. The transformation matrix from the camera to the world.
   a. Rig -> World
   b. Rig -> Camera – using extrinsic matrix
   c. A and b allow us to compute Camera -> World transformation.

Depth Part

5. LUT – Look Up Table, HoloLens Research Mode output for extraction of xyz from the depth data.
    a. When multiplying the pgm image with the lut we get the 3D points in the camera's space.
6. The camera's location in the world's coordinates system.
7. The AHAT – **A**rticulated **HA**nd **T**racking data, one of the depth sensors (effective range: 0-1 meters).
8. PGM (Portable Gray Map) grayscale frames that show depth for each pixel in the frame.
9. Projecting the RGB image on the 3D points.
10. Saving the point cloud using Open3D.

We extracted the point cloud from the depth data (pgm format) using the LUT.

We matched a point from the point cloud to correspondent pixel in the RGB image (PV camera) using the camera parameters (intrinsic, extrinsic, Rig to World and Rig to Camera matrices and sensor's locations). The dependencies between pixel (2D) and the world (3D) are as follow (we used thresholding for finding the nearest closest point between those two sensors).

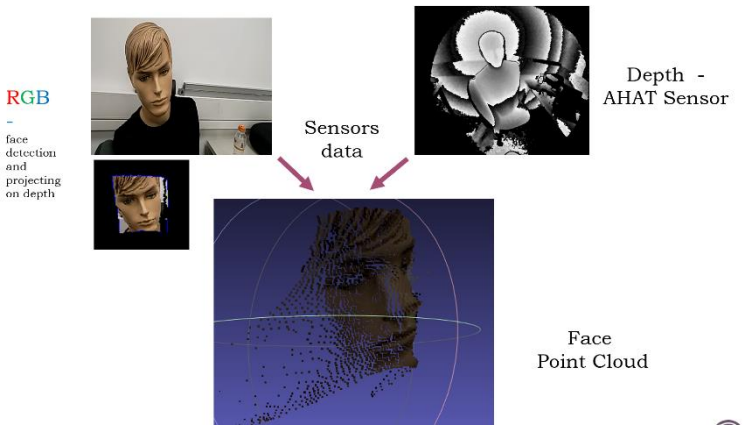
$$w\,[x\ y\ 1] = [X\ Y\ Z\ 1]\,P$$

Scale factor — Image points — World points

$$P = \begin{bmatrix} R \\ t \end{bmatrix} K$$

Camera matrix — Extrinsics: Rotation and translation — Intrinsic matrix

We cropped the point cloud to consist only the patient face. We are doing it by masking all the points which their projections on the PV frame are outside the face. We can find the borders of the face at the PV frame by face detection (explained in the Face Detection section).



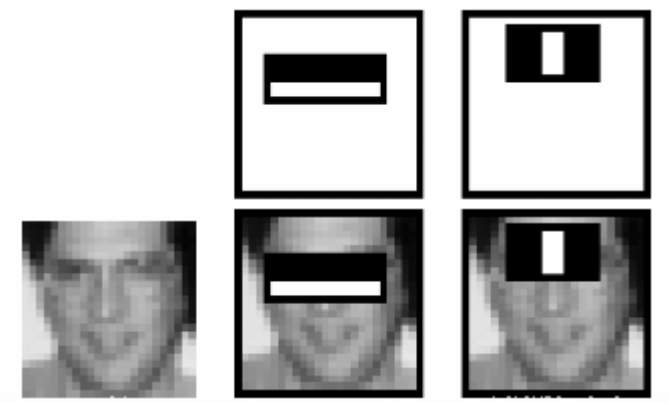
Face Detection Section

We wanted to create a point cloud that contains only the patient's face.

So, we used OpenCV's Cascade Classifier to detect the face in the RGB frame from the HoloLens' camera and mask the RGB image and the depth image.

This algorithm was first introduced by Paul Viola and Michael Jones in their paper, "Rapid Object Detection using a Boosted Cascade of Simple Features" in 2001.

It is a machine learning based algorithm that uses HAAR feature as the convolutional kernel of the network.



The algorithm used AdaBoost using 6000 features (like in the examples above) as the "weak" classifiers for the AdaBoost.

Instead of applying all 6000 features on a window, the features are grouped into different stages of classifiers and applied one-by-one – a Cascade of Classifiers.

This method for combining increasingly more complex classifiers in a "cascade" allows background regions of the image to be quickly discarded while spending more computation on promising object-like regions (the eyes for example).

Registration Section

The registration block in our project is based on a few known algorithms, at this section we will explain each part in the registration flow chart.

Next section, at the Registration Algorithms Explained we will explain in detail the main algorithms we used at the registration block.

The inputs for this block are 2 objects :

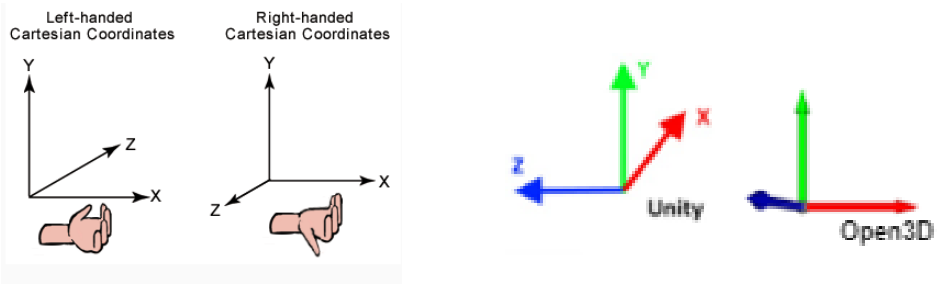
1. The patient's face in real time as a point cloud - destination.
2. The patient's CT scan as a point cloud - source.

There are 3 main steps in the registration block.

A.

First step, those 2 inputs objects must be prepared for the main algorithms later in this pipeline, and the preparation is done using 4 main actions.

1. Transform those point clouds from Open3D coordinate system (runs using python on the remote server) to Unity coordinate system (runs on the device).
   Open3D coordinate system is right-handed coordinate system, in contrast to Unity left-handed coordinate system. Transformation between those two systems will inverse the z-axis direction.



2. Down-sample the point clouds using Voxel down-sampling, this down-sampling operates in two steps:
   a. Points are bucketed into voxels.
   b. Each occupied voxel generates exacty one point by averaging all points inside.
3. Estimating the normal at each point in the point clouds, the estimation at each point is by using the neighbors' points in some defined neighborhood.
4. Fast Point Feature Histogram – a feature extractor we are using on each point in the point clouds. This will be used as a descriptor for finding the matching point pairs between the two-point clouds. The algorithm and the thoughts behind choosing it will be explained in detail in the Registration Algorithms Explained section.

B.

The next step after the preparation is to calculate a rough registration, this information will lie in an affine transformation matrix (4X4 matrix that act as a transformation operator for each of the source's points).

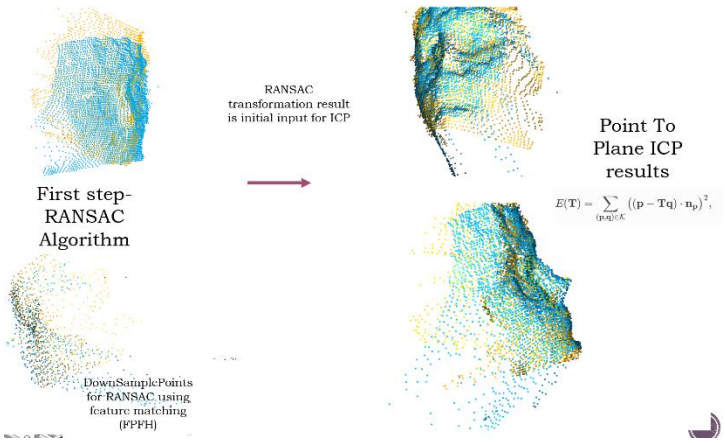
$$\begin{bmatrix} x_o \\ y_o \\ z_o \\ 1 \end{bmatrix} = \begin{bmatrix} a_{xx} & a_{xy} & a_{xz} & a_{xt} \\ a_{yx} & a_{yy} & a_{yz} & a_{yt} \\ a_{zx} & a_{zy} & a_{zz} & a_{zt} \\ 0 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} x_i \\ y_i \\ z_i \\ 1 \end{bmatrix}$$

We are using Global RANSAC based on feature matching algorithm for this part. This is an iterative algorithm using random samples selection for improving the solution at each iteration. The algorithm and the thoughts behind choosing it will be explained in detail in the Registration Algorithms Explained section.

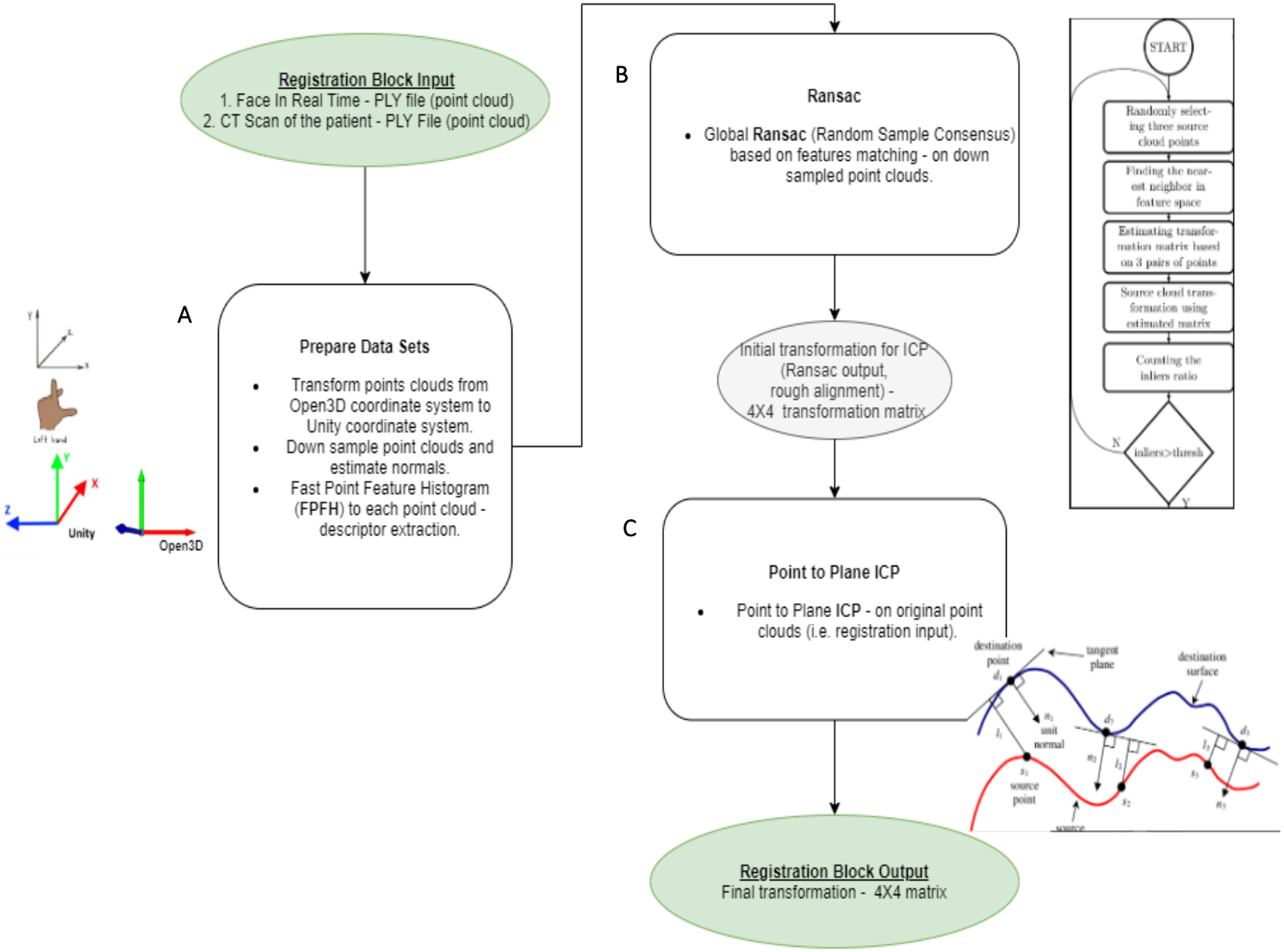
C.

The transformation matrix that we got by the last step will act as an initial solution for the last step.

In the last step we are using ICP – Point to Plane algorithm, the ICP must have an initial solution and it is an iterative algorithm. Every iteration the solution will be improved till some threshold. The algorithm and the thoughts behind choosing it will be explained in detail in the Registration Algorithms Explained section.



Registration Block Flow Chart



Registration Algorithms Explained

Fast Point Feature Histogram – FPFH

The FPFH is an algorithm for getting features which describes any point in a point cloud. The result is descriptor to a point, that helps us find differences between different points in the point cloud. Using that, we can match pairs of points from the source and the target point clouds.

FPFH works as follows:

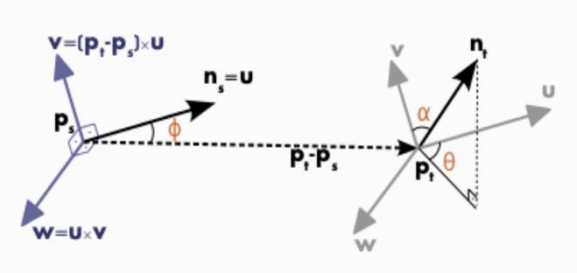
i. For each query point p_q, a set of tuples α, ϕ, θ between the point and its neighbors are computed as described below (Simplified point feature histogram SPFH that is a part of the point feature histogram PFH):

A PFH representation is based on the relationships between all the points in the k-neighborhood and their estimated normals.
The SPFH is the part that computes relative difference between two points p_s and p_t and their associated normals n_s and n_t. We will define a fixed coordinate system at one of the points.

$u = n_s$

$v = u \times \frac{p_t - p_s}{||p_t - p_s||_2}$

$w = u \times v$



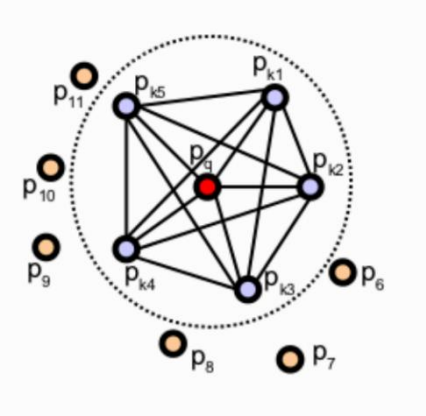
Therefore,

$\alpha = v \cdot n_t$, $\phi = u \cdot \frac{p_t - p_s}{d}$, $\theta = \arctan(w \cdot n_t, u \cdot n_t)$

Where d is the Euclidean distance $d = ||p_t - p_s||_2$.

The computational complexity of SPFH to a specific point is $O(k)$

(The final PFH descriptor is computed as a histogram of relationships between all pairs of points in the k-neighborhood, thus the computational complexity of PFH of all the n points of the point cloud is $O(nk^2)$).

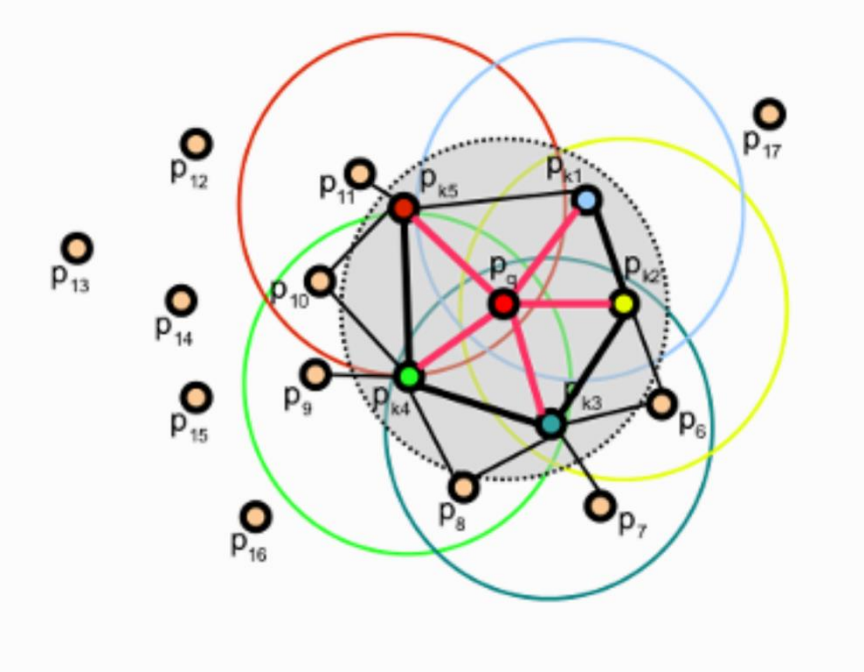


ii. Now, after calculating SPFH for each point, we will use the neighboring SPFH values to weight the final histogram of p_q as follows:

$$FPFH(p_q) = SPFH(p_q) + \frac{1}{k}\sum_{i=1}^{k}\frac{1}{w_i} \cdot SPFH(p_i)$$

Where w_i represents the distance between p_q and p_i in some metric space (Euclidean space in our case).

The computational complexity for all the n points is $O(nk)$.



<u>The reasons for choosing FPFH:</u>

- We had to find some feature space that is fast enough to calculate in a real time application and still can catch most of the discriminative power of the PFH (which known as a powerful feature descriptor for 3D point clouds). The computation is on the down-sampled point clouds.
- FPFH is a good and simple algorithm for description of a local geometric property of a 3D point.
- FPFH has an easy and comfortable implementation within the Open3d library.
- A nearest neighbor query in this feature space can return points with similar local geometric structures, therefore we can use it for matching 2 points from the source and the target point clouds.

<u>Cons</u>

- FPFH does not fully interconnect all neighbors of p_q , and thus missing some value pairs which might contribute to capture the geometry around p_q.

Global RANSAC based on feature matching algorithm

RANSAC (Random sample consensus) is an iterative method to estimate parameters of a mathematical model from observed data. Therefore, we can use this method as a registration method where our observed data is the source, and the mathematical model will be the target.

RANSAC works as follows:

i. Randomly selecting n source cloud points.
ii. Finding the nearest neighbor within the target using the feature space (in our case the feature space based on the FPFH algorithm).
iii. Estimating the transformation matrix based on 3 pairs of points.
iv. Source cloud transformation using the estimated matrix.
v. Counting the inliers ratio:
    - if inliers > threshold or other criteria than end.
    - Else, go back to i.

RANSAC is known as a great non-deterministic algorithm for rough registration, the randomly choices of the points are useful for getting the source closer to the target by terms of translation, rotation, and scaling, still roughly though.

The better the feature space (using the FPFH) is, the better RANSAC will be as it must match between the points.

We also use pruning step that takes fast pruning algorithm to quickly reject false matches early.

<u>The reasons for choosing RANSAC:</u>

- We had to find initial alignment for the ICP algorithm (next algorithm in the pipeline), RANSAC provides some solution. This is not a sufficient quality solution though but a sufficient rough solution.
- This method doesn't deserve an initial solution of its own, due to its randomly choices nature, therefore it is a good idea for using it as an initial solution provider.
- It is a fast-enough method (according to a chosen criterion for convergence) for a real-time application.
- RANSAC based on feature matching has an easy and comfortable implementation within the Open3d library.

<u>Cons</u>

- Non-deterministic algorithms are harder to debug, as every run the result might be different.
- It might be difficult to decide on a sufficient convergence criterion.

ICP Point-to-plane algorithm

ICP (Iterative Closest Point) is an iterative algorithm to minimize the difference between two-point clouds. Therefore, we can use this method as a registration method where we must minimize the difference between the source and the target.

It has been a mainstay of geometric registration in both research and industry for many years. The inputs are two-point clouds and an initial transformation that roughly aligns the source to the target, while the output is a refined transformation that tightly aligns the two-point clouds

ICP variations are using different objective function as explain below.

ICP works as follows:

i. Find a correspondent set $\mathcal{K} = \{(p, q)\}$ from target point cloud P and source point cloud Q transformed with current transformation T (q is the closest point to p).
ii. Update the transformation T by minimizing an objective function $E(T)$ defined over the correspondent set $\mathcal{K}$.
iii. If converges criterion has been reached than end, else go back to i.

The point-to-plane objective function $E(T)$ is:

$$E(T) = \sum_{(p,q)\in\mathcal{K}} \Big((p - Tq) \cdot n_p\Big)^2$$

Where n_p is the normal of point p.

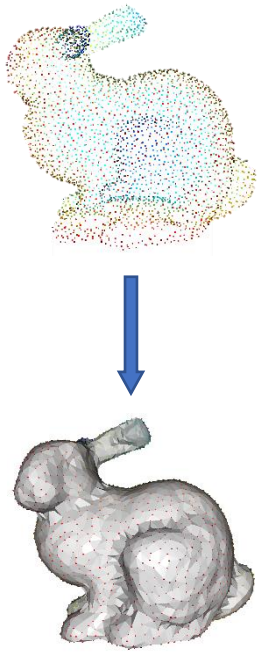
<u>The reasons for choosing ICP:</u>

- ICP is famous and well-known algorithm for geometric registration.
- Point-to-plane ICP assumes that the point clouds have some geometric structure with normal, which works well in our case. The 2-point clouds are face structures.
- It is a fast-enough method (according to a chosen criterion for convergence) for a real-time application.
- ICP has an easy and comfortable implementation within the Open3d library.

<u>Cons</u>

- It might be hard to decide on sufficient convergence criterion.

Surface Reconstruction



To present the point clouds in the scene using Unity, we wanted to convert the point cloud into a triangle mesh.

We used Michael Kazhdan's "Poisson Surface Reconstruction" algorithm.

For that we used the following steps

1. Read the PLY file.
2. The algorithm expects oriented normal for every point, so we used Open3D's function that computes normal by finding adjacent points and calculating the principal axis of the adjacent points using covariance analysis. It implements it using a KD-Tree.
3. The Poisson Surface Reconstruction solves a regularized optimization problem to obtain a smooth surface.
   It defines a 3D indicator function χ (defined as 1 at points inside the object, and 0 at points outside), and then obtains the reconstructed surface by extracting an appropriate iso-surface.
   The gradient of the indicator function is a vector field that is zero almost everywhere (since the indicator function is constant almost everywhere), except at points near the surface, where it is equal to the inward surface normal. Thus, the oriented point samples can be viewed as samples of the gradient of the model's indicator function.
   We will denote this vector field as $\vec{V}$.
   So, we want the vector field to be as close as possible to the indicator's function's gradient, so our problem is: $\min \chi \left|\left|\nabla\chi - \vec{X}\right|\right|$. We can now apply the divergence operator and get the Poisson problem: $\Delta\chi \equiv \nabla \cdot \nabla\chi = \nabla \cdot \vec{V}$.
   Now all is needed is to solve the Poisson problem and get χ and from it to get the reconstructed object.

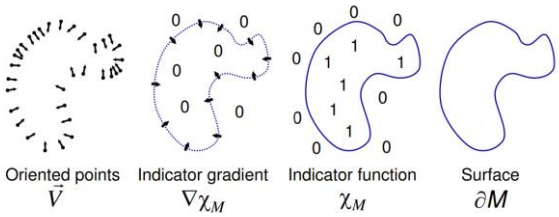


Figure 1: *Intuitive illustration of Poisson reconstruction in 2D.*

Challenges & Solutions

Challenge	Solution
The transformation was incorrect in the Unity's coordinates system.	Applied the transformation on the object and sent the object using ZeroMQ.
Slow internet for sending data.	Used our own designated router.
Complex HoloLens sensors.	Chose AHAT depth sensor for short distance depth.
No good example CT scan model file.	Used the HoloLens to scan a face and create a model to be used as a CT scan.
HoloLens 2 is still developing and not very popular among many developers.	Ongoing Google searching, GitHub.

Conclusion and future work

We suggested a solution to the problem that is algorithmic and engineering solution as well.

Our solution is autonomic and doesn't require surgeon's intervention, except manual configuration at the beginning of the operation. Quick configuration for the communication between the device and the remote server (entering the IP addresses).

We still see error and inaccuracy at the visualization on the device, due to the communication latencies and the drifting errors. The HoloLens 2 is still a pretty new device, because of that, bugs might pop up and there is a lack of the documentation online. Updating the project to a newer version of the device (when it will be released) might improve it significantly.

Future work might find our architecture convenient and improve it using other communication methods. As well, different rigid registration problems can be solved using it.

As an example, we have an implementation for the Long Throw sensor (other depth sensor instead of AHAT sensor), which might be helpful for rough objects registration at a longer range (effective range: 1-6 meters).

HoloLens 2 visualization

Our Code

https://github.com/nir6760/HL2proj

Resources

1. HoloLens' Research Mode
   https://docs.microsoft.com/en-us/windows/mixed-reality/develop/advanced-concepts/research-mode
2. Intrinsic & matrices in Apple developer website.
   https://developer.apple.com/documentation/avfoundation/avcameracalibrationdata/2881135-intrinsicmatrix
3. OpenCV's Cascade Classifier
   https://docs.opencv.org/3.4/db/d28/tutorial_cascade_classifier.html
4. Rapid Object Detection using a Boosted Cascade of Simple Features
   https://www.cs.cmu.edu/~efros/courses/LBMV07/Papers/viola-cvpr-01.pdf
5. Open3D's Poisson Surface Reconstruction
   http://www.open3d.org/docs/latest/tutorial/Advanced/surface_reconstruction.html#Poisson-surface-reconstruction
6. Poisson Surface Reconstruction Paper
   https://www.cs.jhu.edu/~misha/MyPapers/SGP06.pdf
7. Camera Calibration
   https://www.mathworks.com/help/vision/ug/camera-calibration.html
8. Global Registration
   https://www.comp.nus.edu.sg/~lowkl/publications/lowk_point-to-plane_icp_techrep.pdf
   http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.106.3035&rep=rep1&type=pdf
   http://www.open3d.org/docs/release/tutorial/pipelines/global_registration.html
9. HoloLens 2 for CV
   https://github.com/microsoft/HoloLens2ForCV
10. Stream Recorder
    https://github.com/microsoft/HoloLens2ForCV/tree/main/Samples/StreamRecorder