Measure It!

Yi Hua
The Robotics Institute
Carnegie Mellon University
yhua1@andrew.cmu.edu

Yu-Fang Chang
The Robotics Institute
Carnegie Mellon University
yufangc@andrew.cmu.edu

Abstract

As an example to understand and interact with the environment through mobile applications, we built an app for measuring distance between 3D points with depth sensor and color cameras on a tablet. The goal of our project is to deliver an iOS app for the user to measure 3D distances with the structured sensor. The challenge of this project lies in working with a noisy depth map and approximate user annotations.

1. Introduction

The challenge is to obtain accurate 3D measurement between two points the user is interested in. However, the depth map is noisy, and the user annotations are approximate. This formulates as a mapping question: given an approximate touch location on the screen, how can we intelligently map it to 3D point in the scene that is worth measuring, while taking into account the resource limitations of an mobile device?

In this report, we discuss the methods we explored for improving the accuracy, usability, and speed of this measurement app, as well as the results of those methods.

2. Background

Our projects are inspired by the Google Tango demo (`https://www.youtube.com/watch?v=5ZSjpw9t1r8`), which enables mobile devices to detect the position relative to the surroundings. We are focusing on physical space measurement that users can move the cursor to place the points and the app will show the measurement of the distances. The nature of mobile devices makes it handy as a measurement device, and computer vision makes it possible to input the surrounding environment into the mobile device for computation. Furthermore, efficient vision algorithms can be used to design mobile applications with simple, intuitive user interface.

3. Approach

3.1. 3D Point From Depth Map

The structure sensor that supplies us depth map is a structured-light sensor. It obtains depth information about the scene by projecting infrared light patterns and capturing the result with a infrared camera. The structured sensor provides calibration software for locating the depth camera in iPad's color camera coordinates or vice versa.

In the depth camera frame, we can obtain 3D point location corresponding to depth map location $p = [p_x p_y]^T$ by

$$\tilde{P} = K^{-1} p^T$$
$$P = \frac{depth(p)}{\tilde{P}_Z} \tilde{P}$$

where K is the intrinsics of depth camera and $depth(p)$ denotes the depth at location p. Then the distance between two 3D points can be calculated.

Together with the touch to select points functionalities, this way of mapping two screen points to 3D points gives us a naive implementation of the measuring distance with depth sensor on a mobile device.

3.2. SLAM

We added SLAM to our app because we wanted to know about the camera pose and the environment structure.

First, the use case of mobile devices is to be held in users hands and often move around when the app runs. When camera location moves the selected points on screen maps to different 3D points in the environment while the user wants to measure the point they initially pointed at. SLAM provides us with the camera pose.

Second, length measurements are often concerned with object's size, and depth edges created by the boundary of objects are of special interest to us. However, the depth map obtained from structured light is often unreliable in those areas; we observe that depth map flickers on the edge of objects. SLAM implementation in the structure SDK pro-

vides a mesh representation of the environment, and integrates many observations of the depth maps across time.

With SLAM, we can obtain the 3D point by intersecting ray from camera $\tilde{P}$ with the world mesh to obtain 3D point coordinate P. The distance between two points in depth camera coordinate P_1, P_2 with corresponding camera pose T_1, T_2 is then measured as

$$||P_2 - T_2 T_1^{-1} P_1||$$

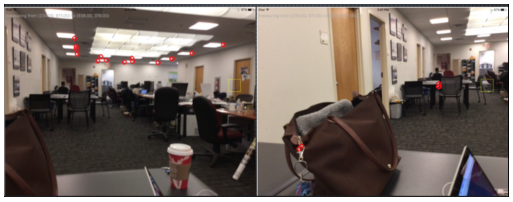
3.3. Accurate User Annotation

One challenge of the project is to get the accurate point that user is interested in. For example, to measure a desk, we aim to select the corner of the desk while it might be the case that we actually select the nearby point on the ground, which results in the wrong measurement. We approach this problem by attempting to extract only points that are meaningful for measurement, and snapping the user selected point to the closest meaningful point. We tried different methods for understanding points that meaningful for measurement.

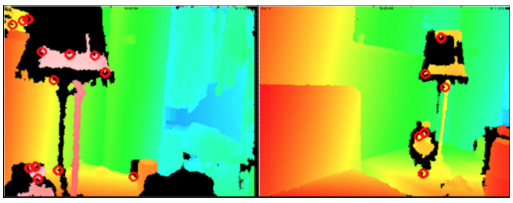
First, geometric corners in a scene are often desired end points for measurement, we tried to apply Harris corner detector directly on depth maps. However, as shown in figure 1 (b), the detector pick up the irregular-shaped holes in the depth map, instead of actual geometrical corners in the scene. Our second thought was to apply Harris corner detector on hole-less color image, and filter out corners that resulted from texture. However, Harris corner detector detects image patches with large gradient change in both direction, but not 'structural' corners where two edges meet, like table corners, as shown in figure 1 (a).

Then we decided to implement our understanding of structural corners, where two edges meet. We extract edges with Canny edge detector and then Hough line transform to get possible line segments. However, we found that unreliable line segments gave more unreliable corners from the their intersections. So we resort to keeping all the line segments as potentially meaningful measurement points.

Since we extract all the possible lines from the frame, as the blue lines shown in Figure 2, once we select or move a point, we can derive the white line segment which is the closest to the selected point. Different from common problem to find the distance between a line and a point, here we are trying to find the distance with a line segment instead of a infinite line. Let that the line segment X be defined by point A and point B and line Y is perpendicular to line X and pass through point P and intersect with line X at point Z. If Z is not on the segment, we return the point A or B which is closer to point Z. Therefore, every time we select a point, we can see the App snapping it to a most possible point on the line for better measurement. It is simply done by project the selected point to the line segment. Additionally, we set

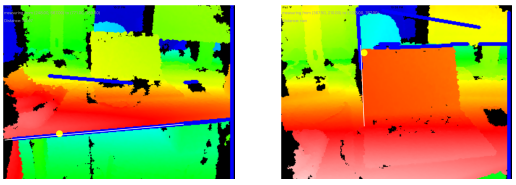


(a) Harris corner detection on color image.



(b) Harris corner detection on depth map.

Figure 1



(a) (b)

Figure 2: Detected lines (blue) and the closest line (white) for snapping the point.

a threshold for searching nearby edges. We only snap the point to an edge if it is close to the line segment under a small distance.

3.4. User Interface

We considered different types of user interfaces: 1) user tap anywhere on screen to select 3D points imaged at that location, 2) user choose one from a couple of interest points shown on screen, 3) user tap to select the 3D point imaged at the center of the screen, similar to the Project Tango MeasureIt interface.

We soon found out, with inaccurate interest point detection, the second option is rather limiting - it will not let users measure the points they actually intend to measure. Furthermore, when snapping to nearby interesting point is implemented, the first option is rather expensive as we will be attempting to understand the scene covering the whole frame. The third option saves significant computation by taking advantage of the fact that users can move their mobile device such that their desired location is near the center of the screen. However, the third option requires accurate estimation of camera poses across frame.

4. Results

For a comparison between a naive implementation versus our final implementation, please see https://youtu.be/qlRAthArTmc.
Below we will briefly discuss different aspects of the performance:

4.1. Accuracy

Direct Measurements From Depth Map Without implementing snapping to edge or SLAM, what gets measured very largely depends on how stable the user can hold the tablet and how patient they are about getting close to the edge of the object. Additionally, the flickering distance calculated from each new frame of depth map means the user need to mentally average the readings to infer the true distance being measured. However, when the we take care of those factors and avoided holes in the depth map, we carefully and repeatedly measured small objects of known size (laptops, iPads, monitors), and obtained measurements that are precise up to 1 centimeter. It was impossible to measure larger objects like doors, because the depth sensor has a depth limit of 3.5 meters [2], while this naive implementation requires fitting both measurement points in the iPad screen.

Measurements With SLAM and Snapping After implementing snapping to edges, it became much easier for the user to select the same point, since now points will be consistently selected from depth edges in the scene. Additionally, tracking camera positions makes measuring large distance possible as we can move and rotate iPad between point selections. However, repeatedly measuring objects of the same size (tables, doors, whiteboards) gave us poor precision, up to 0.2 meters. This is due to the fast yet imprecise SLAM implementation from the structure sensor SDK. While the underlying implementations are not public, our suspicion is that it does not perform loop closure when new depth and color frame is integrated to save computation; for example, the mesh it built from looping around a foosball table can be seen in figure 3. For structured sensor sample applications, the refinement step is probably done in batch after the capturing is finished, in the finalizeTriangleMesh function. To improve the accuracy of measurements, we will need to dig into the implementations of the SLAM algorithm. We think that implementing SLAM with depth and color image inputs while respecting the resource limitations on a mobile device could be a interesting yet challenging course project for the future.

4.2. Speed

The computationally intensive parts of the SLAM, like integrating a new depth and color frame into the current mesh, are handled on new threads. All the functionalities provided by the structure sensor SDK runs smoothly, though they do drain battery.

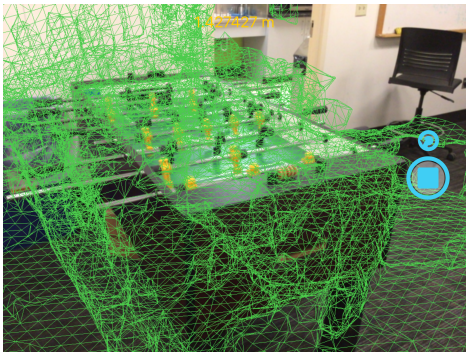


Figure 3: Failed mesh reconstruction from SLAM.

Since we used OpenCV functions including corner detectors and edge extractors to understand the scene, this part was more time-consuming. When we extracted line segments from the whole depth image by Hough line transform, and computed the distance between user-selected points to each of those lines, the frame rate dropped noticeably to around 6 fps. Implementing user interface in the style similar to Google Project Tango MeasureIt [1] greatly reduces the area of scene we need to understand. This cuts computation on 320×240 pixels to selecting the best pixel out of 100 points near the center of the screen, about 0.1% of the original computation. Additionally, limiting the selection meant it becomes beneficial to store the ray directions for each potential selection point, and the 3D point location within depth camera frame can be obtained with 2 multiplications. The conclusion here is mobile applications should take advantage of the mobility of the device, and careful design of the user interactions can greatly reduce the computations needed for the task.

5. List of Work

Equal work was performed by both project members.

6. Github

Github link:
https://github.com/jujudydy8196/Measure-it
Since the project checkpoint, we integrated SLAM functionalities that came with the structured sensor SDK, experimented with more ways of snapping touched screen point to edges, and implemented a user interface similar to the Tango MeasureIt[1].

7. Tips for Working with Structure Sensor

1. Structure sensor USB hacker cable uses the lighting port, so it is not possible to run your app while iPad is connected to the Xcode on computer. To debug, use STWirelessLog and setup writing to your desired IP address and port number in AppDelegate.mm application didFinishLaunchingWithOptions.

2. Structure sensor outputs depth frame at the resolution of 320×240, different from the size of view that displays it.

3. Intrinsics to the depth camera can be found in this thread: `http://forums.structure.io/t/getting-colored-point-cloud-data-from-aligned-frames/4094/2`.

References

[1] Google. Google measure it demo with tango.

[2] O. Inc. Structure sensor technical specifications.