# Literature Review: Android 3D Scanner

By Joshua Tang

## Project Description

Software has always been considered as to how it affects the virtual world, often by optimizing real world tasks or creating new ways to solve problems. In recent years, with the rise of 3D printing, software can create new structures and parts for daily use. My project proposes to create a simple and accessible smartphone app which will allow users to scan objects in the real world and create 3D models of those objects. The app will allow users to utilize a sort of “copy-paste” methodology on physical objects. The app will also let users not only make models of objects that they encounter, but also let them modify those objects and make molds for those objects.

Currently all commercially available software that exists to create 3D models of objects on smartphones is very slow and clunky( see 123D Catch). The app we are proposing to make will be much more user friendly and much faster as it will create the model on the phone itself rather than sending the data out to a server to be computed. We will also be the first publicly available app to permit the users to directly edit the models resulting for the scan and create a mold for said models. In terms of research, in October, after I had proposed this project. Microsoft Research published the Mobile Fusion [14] paper which behaves similarly to our proposed app. While this is unfortunate, our app will being using a different methodology to obtain the 3D models and as such it will be interesting to compare and contrast our results with those of Microsoft's Oxford research team.

My app, will be extracting ORB features[5] from the video it is taking in order to perform monocular ORB-SLAM[6] on the scene that is being presented to it. Using the keyframes selected from the SLAM algorithm in order to create a semi-dense 3D mapping of the depth of each point from the frames [2]. The resulting mapping will be turned into a mesh which will then be available to view, edit and export to 3D printers, game engines, virtual worlds, anything really.

## Literature Review

*Feature extraction*

In order for our algorithm to work, we are going to need to be able to rapidly extract features from the scene in order for us to perform the SLAM algorithm. The features that are picked are going to need to be both scale, translation and rotation invariant, which is to say that they will remain the same regardless of the image's scale rotation or position. Looking through the literature, we find that the initial influential paper in this domain is the paper on SIFT [6], which describes exactly the kind of features we are looking for based on those feature's mathematical properties. SIFT is a little bit of a hack since the selected mathematical features that are picked were hand picked by researchers rather than being completely based on vision. Despite this fact SIFT works quite well. The main issue with using SIFT is that it requires a lot of computations and as such is very slow. Next, we look at SURF[7], which is an accelerated form of SIFT. SURF possesses the speed we are looking for, however it is less precise than SIFT and will break if sufficient noise is present in the scene. Which finally brings us to ORB[5] which is a combination of FAST and BRIEF. It has the speed and precision we are looking for and the only downside to using it is the fact that it breaks when exposed to pure rotations and like the other feature types, it doesn't track moving objects. I've decide to go with ORB for the project specially

since it is the feature used in the ORB-SLAM dense probability mapping[1][2].

*Tracking and Mapping*

Now we are going to need to track the invariant features we've extracted from the scene in order keep track of where they are even when they are obstructed by other objects. Additionally we need to be able to map these features to a 3 dimensional space, giving each of them a place in the physical space. All of the algorithms we will now be looking at functino by taking multiple keyframes throughout time, extracting some sort of feature from these keyframes and uses the information from the other keyframes to create a map. I took note of the work done in the Parallel Tracking and Mapping paper [8] which uses the FAST features and demonstrates how running multiple parallel processes to track and map all of the points is better than any method that exist prior to it and is very useful to helping a virtual environment interact with the real environment. The LSD-SLAM algorithm [11] get's closer to what we are looking for by creating a dense mapping of of it's environment which makes it good for our purposes, however, the algorithm's precision leaves something to be desired. We then come to DTAM[9], DPPTAM[10] and ORB-SLAM[1][3], all of which suit our purposes rather well in term of precision and speed. In the end I chose ORB-SLAM because it seemed like the easiest to implement in a relatively short time period and uses less processing power than the other two while still giving us great results. In addition we know that ORB-SLAM has been successfully been implemented on phones before [16] which, confirms the feasibility of this project, even though that example did not implement the semi-dense probability mapping [2] that we will be using.

*Pose Estimation, Relocalisation and Auto-calibration*

In order to get a good mapping of a scene, we are going to need to know precisely where are camera is. To do this, we are relying on good relocalisation [4] to tell us when our pose estimation is wrong and to allow us to find the correct pose by looking at the points we have already successfully mapped. This is a crucial part of both Tracking and Mapping the scene [3]. In addition, after some initial testing, we know that even for the same model of android phone, the camera is not necessarily calibrated the same and thus we need to calibrate the algorithm to the needs of the camera. While we could let the user calibrate the camera themselves using the method OpenCV has for calibrating cameras, this would be obnoxious to our users and would make the app less user friendly. Instead we can automate the calibration on the phone automatically[15], which should make using the app much easier.

*Creating and Smoothing a 3D Mesh*

Finally, once we have the dense probability mapping of an object, we are going to need to turn it into a mesh so that we can edit it. There are many different ways to do this but using spherical parameterization [13] seems like a good idea due to fact our point cloud won't be particularly dense. Other options [12] are available to us, but currently it seems like spherical parameterization will be our choice for this project. Editing the resulting model and displaying it should then be easily feasible. We should save the model as an STL or .obj file so that it is easier to share, print or use in other projects.

## References


[1] R. Mur-Artal, J. Montiel and J. Tardos, 'ORB-SLAM: A Versatile and Accurate Monocular SLAM System', *IEEE Trans. Robot.*, vol. 31, no. 5, pp. 1147-1163, 2015.

[2] R. Mur-Artal and J. Tardos, 'Probabilistic Semi-Dense Mapping from Highly Accurate Feature-Based Monocular SLAM', Robotics: Science and Systems. Rome, Italy, July 2015.

[3] R. Mur-Artal and J. Tardos, 'ORB-SLAM: Tracking and Mapping Recognizable Features',Robotics: Science and Systems (RSS) Workshop on Multi VIew Geometry in RObotics (MVIGRO), Berkeley, USA, July 2014.

[4] R. Mur-Artal and J. Tardos, 'Fast Relocalisation and Loop Closing in Keyframe-Based SLAM.', IEEE International Conference on Robotics and Automation (ICRA), Hong Kong, June 2014.

[5] Rublee, Ethan, Vincent Rabaud, Kurt Konolige, and Gary Bradski. 'ORB: an efficient alternative to SIFT or SURF.' In *Computer Vision (ICCV), 2011 IEEE International Conference on*, pp. 2564-2571. IEEE, 2011.

[6] D. Lowe, 'Distinctive Image Features from Scale-Invariant Keypoints', *International Journal of Computer Vision*, vol. 60, no. 2, pp. 91-110, 2004.

[7] H. Bay, A. Ess, T. Tuytelaars and L. Van Gool, 'Speeded-Up Robust Features (SURF)', *Computer Vision and Image Understanding*, vol. 110, no. 3, pp. 346-359, 2008.

[8] Klein, Georg, and David Murray. "Parallel tracking and mapping for small AR workspaces." In *Mixed and Augmented Reality, 2007. ISMAR 2007. 6th IEEE and ACM International Symposium on*, pp. 225-234. IEEE, 2007.

[9] Newcombe, Richard A., Steven J. Lovegrove, and Andrew J. Davison. "DTAM: Dense tracking and mapping in real-time." In *Computer Vision (ICCV), 2011 IEEE International Conference on*, pp. 2320-2327. IEEE, 2011.

[10] Concha, Alejo, and Javier Civera. "DPPTAM: Dense piecewise planar tracking and mapping from a monocular sequence." In *IEEE/RSJ international conference on intelligent robots and systems*. 2015.

[11] Engel, Jakob, Thomas Schöps, and Daniel Cremers. "LSD-SLAM: Large-scale direct monocular SLAM." In *Computer Vision–ECCV 2014*, pp. 834-849. Springer International Publishing, 2014.

[12] Fabio, Remondino. "From point cloud to surface: the modeling and visualization problem." *International Archives of Photogrammetry, Remote Sensing and Spatial Information Sciences* 34, no. 5 (2003): W10.

[13] Zwicker, M., and C. Gotsman. "Meshing point clouds using spherical parameterization." In *Proc. Eurographics Symp. Point-Based Graphics*. 2004.

[14] P. Ondruska, P. Kohli and S. Izadi, 'MobileFusion: Real-Time Volumetric Surface Reconstruction and Dense Tracking on Mobile Phones', *IEEE Trans. Visual. Comput. Graphics*, vol. 21, no. 11, pp.

1251-1258, 2015.

[15] Saponaro, Philip, and Chandra Kambhamettu. "Towards Auto-calibration of Smart Phones Using Orientation Sensors." In *Computer Vision and Pattern Recognition Workshops (CVPRW), 2013 IEEE Conference on*, pp. 20-26. IEEE, 2013.

[16] Shridhar, Mohit, and Kai-Yuan Neo. "Monocular SLAM for Real-Time Applications on Mobile Platforms." (2015).