Creating a Geospatial Macro Level Feature Database

Jason Brewer

Final Report

Submitted to

The University of Liverpool

in partial fulfilment of the requirements
for the degree of

MASTER OF SCIENCE

September 5, 2017

Contents

Chapter 1

Problem Summary

Blind or visually impaired (BoVI) individuals face many day to day issues regarding the ability to navigate the outdoor world with confidence. As highlighted by Quinonoes et al. [1], a given hardware and software system for BoVI users must address navigation focussing on recovery from mistakes on a given route, losing one's way, unfamiliar routes and changes to a route which the system must react to efficiently and robustly.

By understanding the needs of the BoVI user, it becomes apparent that a system must be able to identify meaningful waypoints on a given outdoor route whilst at the same time not supplying an overwhelming amount of information. In effect, the system must act like a background process with an occasional element of interactivity: gathering useful information about the world as it would be perceived by a sighted person and, when required, relaying relevant information back to the BoVI user when required.

The system previously proposed to tackle this problem, known as Siloam, is comprised of four software components:

- SiloamSee (the data collection subsystem)
- SiloamLearn (the machine learning subsystem)
- SiloamSilo (the online data storage subsystem)
- SiloamRoute (the navigation subsystem)

Together these components aim to define a standard whereby the information of street level macro level features can be collected and analysed in any arbitrary location.

The project is being developed in conjunction with another University of Liverpool student. It is they who will deal with the development of SiloamRoute. Because of this and issues that will be covered in chapter 3, only details of the development of SiloamSee and SiloamLearn will be explored in this document.

Chapter 2

Outputs

2.1 SiloamSee

The original plan for the SiloamSee component, as discussed in section 2.2.3 of the proposed design and specification, was to extend the open source stock Android camera application from the KitKat version of the operating system. This was the last to feature so called "photo sphere" functionality as standard: that is, the ability to capture a 360° panoramic spherical image. This is represented in two dimensions in an equirectangular Plate Carre projection, which is how the photo sphere is stored on any given device. The proposed extension would utilise aspects of the Google Project Tango C++ API [2] to include the collection of depth information whilst encoding the RGB data captured by the device that has been used to test the final implementation of SiloamSee, the Lenovo Phab 2 Pro (henceforth referred to as "P2P").

However, this approach proved unsuitable because most of the Android source would need to be built along with the stock camera application resulting in a non modular result. Efforts were made to extend the open source Android application Focal [3] which also features photo sphere functionality. These efforts also proved fruitless both because of the target device of the application, the LG Nexus 4, and the lack of active development, the last commit being on 20/02/2014. The application built and ran on the P2P but was too unstable.

A more actively developed open source application is Matthieu Labbè's RTAB-Map [4]. A design choice was made to move away from the photo sphere plus depth information idea and focus on the full 3D model capturing that RTAB-Map offers natively. Labbè's creation was extended to include GPS and magnetometer data which is shown in figure 2.1. Primarily using Java, three extra buttons were added to the main activity of the application (GET HEADING, START GPS and STOP GPS) as well as real time updates of the last GPS coordinate captured by the P2P. The core of RTAB-Map and these subtle extensions form the SiloamSee subsystem.

As soon as the application starts, the GPS sensors are polled at a minimum of two second intervals. The actual interval can be larger that this due to the nature of the Android GPS polling method and various multithreading behaviours inherent in most modern operating systems. When the user has finished capturing the current outdoor location with SiloamSee, it is recommended the press the STOP GPS button to ensure the OBJ 3D model file export, another feature included as standard in RTAB-Map, executes without error.

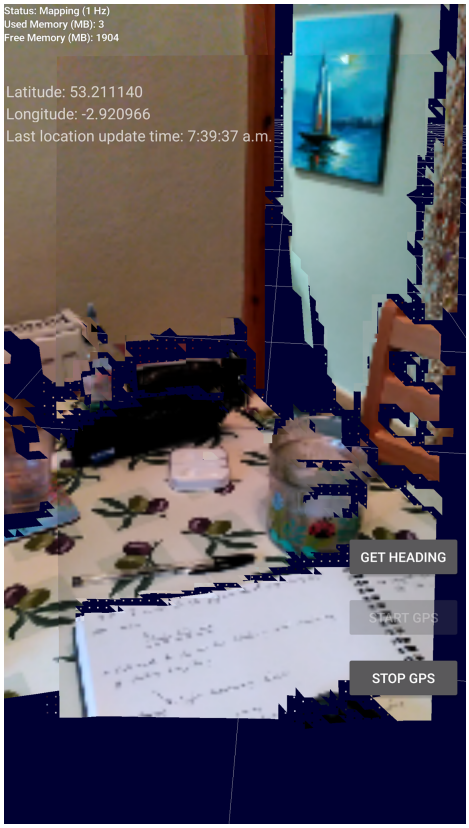

Figure 2.1: Sample view of the extended RTABMapActivity that forms the basis of the SiloamSee Android application

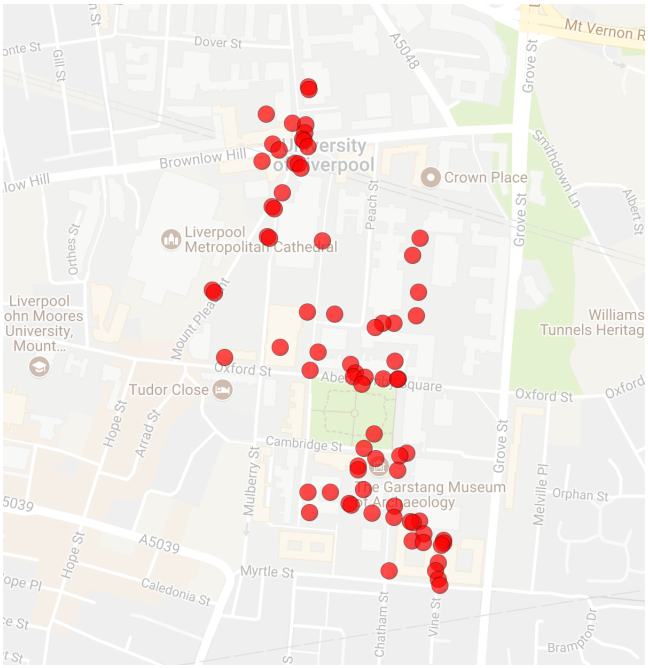

Figure 2.2: Map showing the locations of the 76 locations captured by SiloamSee

The locations of the 76 models captured by a P2P running SiloamSee are showed in figure 2.2. Each model was then labelled as either test, train or ground truth data to correspond to the areas shown in figure 2.6 of the proposed design and specification.

2.2 Data Preparation

Taking inspiration from the methods presented by Fehr et al. [5] , the next step in development was to generate an RGB-D dataset from the 3D models generated by SiloamSee. The paper discusses using not just RGB and depth images ("D" images) but also RGB representations of the surface normals ("N" images) of a given 3D scene, the grayscale images ("I" images) and various applications of the Sobel filter.

With its powerful Python scripting interface, the open source 3D modelling application Blender [6] was a good candidate to render RGB/D/I/N representations of a variety of views of the the 76 models captured by SiloamSee. To this end a script, Blender_ImportAndRender.py, was produced that contains functions to automatically import and render the RGB/D/I/N data for 16 views of each of the 76 models, totalling to 4864 2D images in total. The placement of the cameras that were produced by the script are shown in figure 2.3.

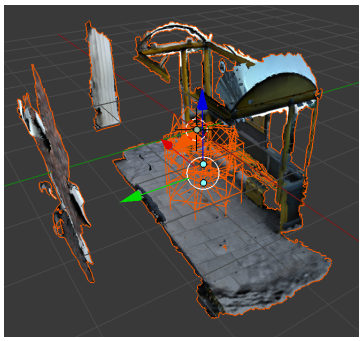

Figure 2.3: Sample bus stop model. The orange mass centred around the axis representation in the centre of the image is the 16 cameras. They face outwards away from the central point. Each are offset 45° from eachother to ensure correct overlapping so no macro level features will be lost

The project also makes use of 3D models downloaded from the Trimble Warehouse [7] to better train the support vector machine (SVM) in the SiloamLearn subsystem. Therefore, Blender_ImportAndRender.py also features functionality to define a square/rectangular perimeter of cameras around an imported DAE 3D model. This is shown in figure 2.4. RGB/D/I/N representations are also generated for these imported models.

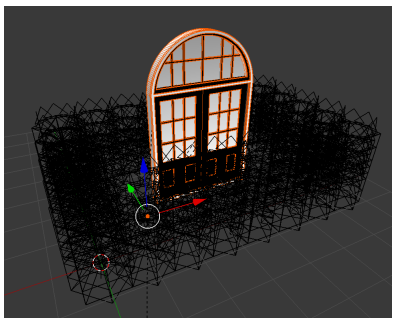

Figure 2.4: The model here is the so-called "ArchDoor". It is surrounded by a rectangular structure of pillars of cameras of dimension 7 units by 4 units. In an interactive fashion, after the user has imported the model and specified the dimensions of the camera rig, the imported model is selected so that it can be moved into the centre of the rig

2.3 SiloamLearn

Again following the methods of Fehr et al. [5], the SiloamLearn C++ application developed thus far creates a database of all of the paths to the 4864 images produced by Blender_ImportAndRender.py and uses the DBSCAN [8] algorithm to segment the objects of interest from the floor plane. The abstract version of this algorithm can be condensed as follows:

- 1. Process the bottom quarter of an N image to get an idea of possible RGB values of floor normals
- 2. Process the rest of the image and find all pixels that fall within the range of RGB values found in step 1. Use this to create a "floor estimation" pixel set
- 3. Remove all the pixels in the "floor estimation" set from the D image
- 4. Run the DBSCAN algorithm and add all pixels in clusters of less than size "kNormalNoiseThreshold" (experimentally set at 2000) to the "floor estimation" set
- 5. Make all the pixels in the "floor estimation" set black for each of the RGB/D/I/N images

The implementation of DBSCAN is provided by the pyclustering C++ library [9]. A graphical representation and a sample result is show in figure 2.5.

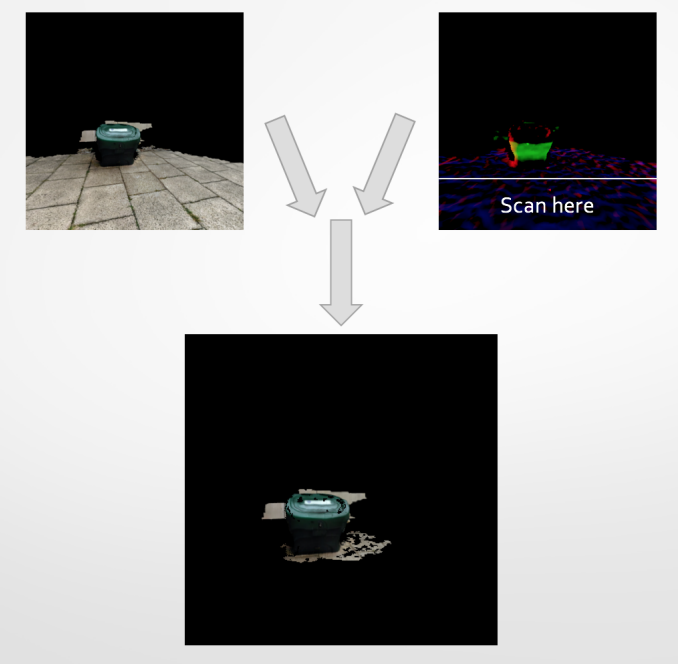

Figure 2.5: Sample results of the experimental floor removal algorithm

It is important to mention that development of SiloamLearn is still ongoing. Currently in development is the generation of various applications of the Sobel filter to I and D images to complete the dataset needed to form the covariance descriptors used to train the SVM outlined in Fehr et al. [5]. The final application will use the trusted Sobel functionality from OpenCV [10].

Chapter 3

Evaluation

Although a working Android application has been completed, initial teething difficulties with the development of SiloamSee meant that in reality development took four weeks instead of one, as shown by the arrow extending from the allotted time in figure 3.1. The light gray circle shows the actual completion date. Following this, development of SiloamLearn started 3 weeks late and continues to the day of writing. This means that, while essentially a simple text processing task, the interface between SiloamLearn and SiloamSilo has yet to begin. Having said that the library needed, cpprestsdk [11], is in place and is well maintained and tested.

Furthermore, while figure 2.5 shows a fairly successful application of the experimental floor removal algorithm this is not the case for all images tested. In some cases very few areas of the floor plane are removed and in other cases large parts of the objects of interest in the image are removed. At present it is not a general solution; further experimentation with "kNormalNoiseThreshold" as well as the parameters of DBSCAN, ϵ and minPts is required.

While too late in the day to change the course of the project completely, future development may make use of PCL [12] to process coloured point clouds instead of RGB-D data. 3D OBJ files are readily converted to point clouds using the library and may lead to greater insights into the data captured by SiloamSee.

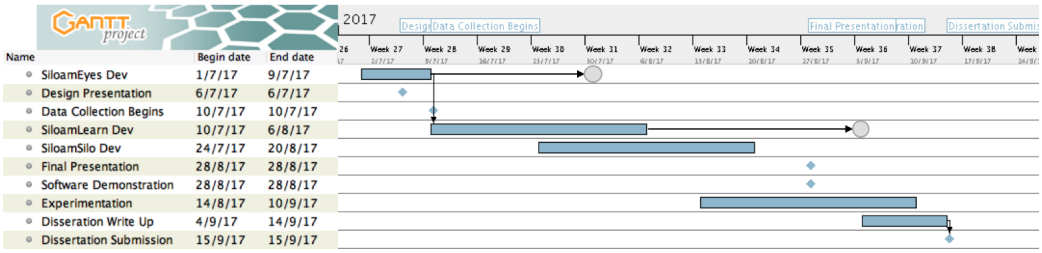

Figure 3.1: The original gantt chart from the proposed design and specification with indications of actual development timescales. The light gray circle in the first row signifies the real completion date while the light gray circle in the SiloamLearnDev row corresponds to the present day

Bibliography

[1] P.-A. Quinones, T. Greene, R. Yang, and M. Newman, "Supporting Visually Impaired Navigation: A Needs-finding Study," in *CHI '11 Extended Abstracts on Human Factors in Computing Systems*, ser. CHI EA '11. New York, NY, USA: ACM, 2011, pp. 1645–1650. [Online]. Available: http://doi.acm.org/10.1145/1979742.1979822

[2] "Tango." [Online]. Available: https://get.google.com/tango/

[3] XpLoDWilD, "android_packages_apps_focal: Open Source Android Camera App," Sep. 2017, original-date: 2013-05-20T19:43:36Z. [Online]. Available: https://github.com/xplodwild/android_packages_apps_Focal

[4] "rtabmap: RTAB-Map library and standalone application," Jun. 2017, original-date: 2014-08-11T18:38:14Z. [Online]. Available: https://github.com/introlab/rtabmap

[5] D. Fehr, W. J. Beksi, D. Zermas, and N. Papanikolopoulos, "Covariance based point cloud descriptors for object detection and recognition," *Computer Vision and Image Understanding*, vol. 142, pp. 80–93, Jan. 2016. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S1077314215001368

[6] B. Foundation, "blender.org - Home of the Blender project - Free and Open 3d Creation Software." [Online]. Available: https://www.blender.org/

[7] T. Inc., "3d Warehouse." [Online]. Available: https://3dwarehouse.sketchup.com/?hl=en

[8] M. Ester, H.-P. Kriegel, J. Sander, X. Xu *et al.*, "A density-based algorithm for discovering clusters in large spatial databases with noise." in *Kdd*, vol. 96, no. 34, 1996, pp. 226–231.

[9] "annoviko/pyclustering: pyclustring is a Python, C++ data mining (clustering, graph coloring algorithms, oscillatory networks, neural networks, etc.) library." [Online]. Available: https://github.com/annoviko/pyclustering

[10] "OpenCV library." [Online]. Available: http://opencv.org/

[11] "cpprestsdk: The C++ REST SDK is a Microsoft project for cloud-based client-server communication in native code using a modern asynchronous C++ API design. This project aims to help C++ developers .." Jun. 2017, original-date: 2014-08-21T20:57:45Z. [Online]. Available: https://github.com/Microsoft/cpprestsdk

[12] "PCL - Point Cloud Library (PCL)." [Online]. Available: http://pointclouds.org/