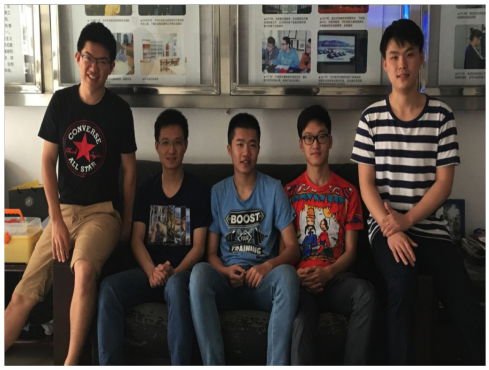
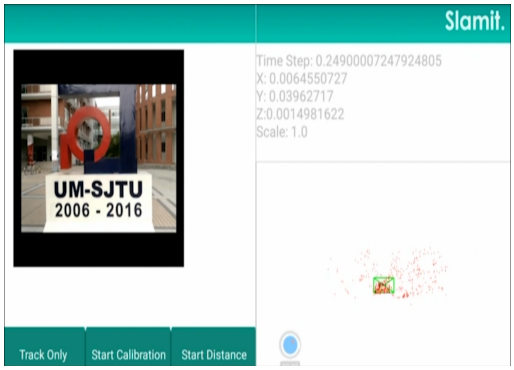
Final Report:
Smartphone-based Low-cost Simultaneous Localization and Mapping

Prepared by:
Team 11
Xiaobai Ma
Kangxu Wang
Zhenkai Wang
Luwei Yu
Zhi Zhang
University of Michigan - Joint Institute
Shanghai Jiao Tong University

Prepared for:
Zhijie Wang
2012 Laboratory @ Huawei Technologies Co. Ltd

Tuesday 9th August, 2016





Executive Summary

The smartphone-based low-cost simultaneous localization and mapping (SLAM) is a novel project that aims to implement the state-of-the-art robot guidance algorithm, SLAM algorithm,on the Huawei P9 smartphone platform. The motivation of this project is to take advantage of the various sensors on the smartphone like camera, magnetic field sensor, and inertial components to provide necessary mapping and localization data for SLAM algrithm. Due to the similarity in sensor system, this project is of great significance for the future study of driverless car.

Base on the requireents of our sponsor, 2012 Laboratory of Huawei Technologies Co. Ltd, and a study on the recent research on this topic and the P9 platform, we have a list of engineering specifications. Here listed the important ones:
(1)The working platform should be Huawei P9.
(2)The mapping and localization environment should focus on road situation.
(3)There should be dynamic objects like pedestrians and cars in the environment.
(4)The map building could be offline. The localization should be online with the latency less than 100ms.
(5)The CPU occupation while running the app on P9 should be less than 300MB.
(6)The app size should be less than 5MB.
(7)The average localization error should be less than 10cm.
(8)The 3G data used by the app should be less than 0.3MB/s.

A detailed explanation of our specifications is included in section 2. Based on these specifications, we did a literature search on difference SLAM algorithms and finally chose ORB-SLAM as the core algorithm of our project. The concept generation and selection process is introduced in section 3 and 4. The ORB-SLAM uses mono camera input and outputs a feature point map as well as camera position and trajectory. Section **??** gives a detailed explanation on it.

Our final design is a Android App called *Slamit*. It could take photos or use camera as its main input. The output of *Slamit* is a feature point map with camera position and trajectory shown on the screen. A detailed description of *Slamit* is given in secion 7.

The hardware cost of our project only contains the price of a Huawei P9 smartphone. Because *Slamit* doesn't need 3G data, there is no additinal cost on running the App. To check the completeness of our specifications, we performed several test on *Slamit*. Besides the fulfillment of the first 3 specifications, *Slamit* gives a satisfying result on memory usage, CPU occupation, app size, and 3G data consumption. However, its accuracy and respond time is remained to be improved and it currently only allows simultaneous online mapping and localization. Detailed test results are given in 8.
In conclusion, *Slamit* has realised the all the fundamental functions required by our sponsor. It reaches most of our initial engineering specifications and has a competitive performance on the memory occupation and 3G data consumption. However, the accuracy and response time is not that satisfying.

Table of Contents

List of Figures

List of Tables

1 Introduction

Simultaneous localization and mapping (SLAM) is a computational problem that describes the ultimate goal in robot navigation and guidance: mapping an unknown environment while simultaneously tracking the robot's position with respect to the surroundings and updating the map. The last two decades have witnessed the rapid and exciting development of the solutions to the SLAM problem since its first mention in early 1990s. Various algorithms, including MonoSLAM,[4] RGB-D SLAM,[3] and ORB-SLAM,[5] have been designed, implemented, and tested. The detailed algorithm introduction will be covered in section **??**.

The traditional SLAM algorithms are usually implemented in robot-based mapping systems, but the burgeoning driverless car industry can also be a perfect application of the SLAM algorithms. Combined with the electrical car and recycled energy field, this newly-developing but thriving industry will have very strong competitiveness in public transportation in the nearby future.

Apart from the SLAM algorithms and the driverless car, the 21th-century also witnessed the huge leap of the smartphone technology and our team project is actually the outcome of the combination of the above three fields. Nowadays, most cars are able to communicate with drivers' smartphones through wired or wireless connection devices. Moreover,the high uniformity between smartphone's hardware(camera, accelerometer, etc.) and sensors needed to solve the SLAM problem ensures smartphones like Huawei P9 an eligible carrier of the algorithms and hence a perfect bond between the driverless car and SLAM programs. Inspired by this idea, Huawei Co. hired our team to investigate the practicability of the implementation of SLAM algorithms in the android environment(Huawei P9).

Compared with traditional localization methods, this project has various advantages.First, with SLAM based localization, the accuracy of this project would be higher than GPS. Second, unlike depth camera based SLAM or radar based SLAM, smartphone-based SLAM makes fully use of the hardware environment of a smartphone and thus would not cause additional cost on the hardware. All these advantages would make our project competitive on the market. Figure **??** shows photos of the GPS ,lidar, and smartphone SLAM.

2 Specifications

Our sponsor, 2012 Laboratory of Huawei Technologies, proposed the basic requirements of this project. After discussion with sponsor, we classify the customer requirements as following:

- Accurate
- App works stable
- Respond fast
- Cost small CPU
- Low cost
- Adaptable for different weathers

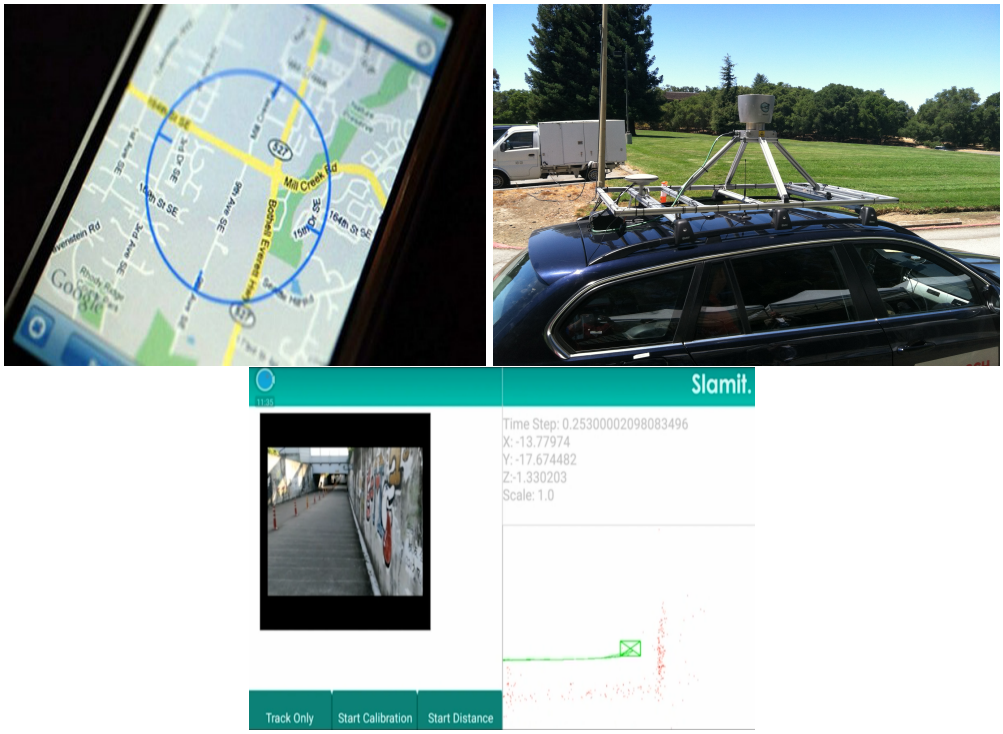


Figure 1: GPS[1](left), Lidar[2](right),and smartphone SLAM(bottom)

- UI looks good
- Show sign in map

The order of customer requirements is based on the importance of each requirement.

According to the customer requirements, we use the Quality Function Deployment(QFD) method to analysis the engineering specifications as well as the competitor products. First, we generate the following engineering specifications which are highly related to the customer requirements.

Photo Dimension: 544px×408px
The size of the photo will affect the time for calculation. We would like to make the photo size smaller as long as it shows the feature points. In that way we can increase the efficiency of calculation. We set the target size as 544px×408px because Andrew D. used the similar photo dimension for his research.[4]

UI Color
We do not propose a specific RGB color for our project. Although the color choice will not affect the functionality of the App, but the UI color will definitely be considered in order to make the App look good.

Time Complexity: $O(N^2)$
Time Complexity of the SLAM algorithm will affect the time for calculation directly. We set the target time complexity as $O(N^2)$ initially. Some SLAM algorithm has the time complexity of $O(N^3)$, which includes the computation for simultaneous mapping. Since we will build the map offline, we consider $O(N^2)$ will be a reasonable value for our project.

Memory: 300MB(10% for Huawei P9)
We want to optimize the App so that it would not take too much memory to execute. Considering the 3GB RAM for Huawei P9, we consider the upper limit for Memory usage is 300MB(10%).

App Size: 5MB
The size of App will influence the user experience. We consider 5MB is enough for an algorithm based App.

Accuracy: 10cm
The requirement for accuracy is fixed in proposal, which is 10cm.

Respond Time: 100ms/frame
We want the respond time to be as fast as possible. Comparing the research of single camera SLAM[4] and ORB-SLAM,[5] 100ms/frame is not good enough. We will try to modify the respond time.

3G Data Consumption: 0.3MB/s
The GPS Module and other data transfer might need Internet. As a limitation, we set the data consumption rate to be 0.3MB/s.

Maximum Moving Speed: 10m/s
If the camera is moving fast, the video will not be very clear and the calculation might be not fast enough to track the position. Thus, we set the max moving speed as 10m/s. We could use bicycle to reach that speed, and 36km/s is a possible speed for vehicle.

Cost
The cost of this app should be as low as possible. In that case, we will try to minimize the 3G Data Consumption to make the App work free.

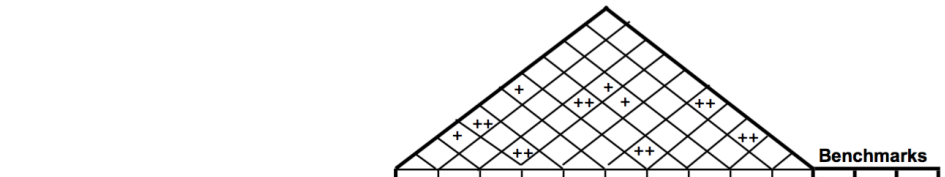
Item	Target Value
Photo Dimension	544px$\times$408px
Time Complexity	$O(N^2)$
Memory	300MB(10%)
App Size	5MB
Accuracy	10cm
Respond Time	100ms/frame
3G Data Consumption	0.3MB/s
Maximum Moving Speed	10m/s

Table 1: Engineering Specifications

Table 1 shows the engineering specifications with value. Figure 2 shows the details of QFD method. Customer requirements are listed in the first column. The weights are distributed according to the importance of each requirement (10 means very important, 1 means not important). The top column contains the engineering specifications. The correlation between each pair of engineering specifications are marked in the roof size grids (++ means strong positive).

The correlation matrix of requirements and specifications are at the middle (9 means strong relationship, 3 means medium relationship, and 1 means small relationship). After normalization of the score for each specification, we find out the Time Complexity is the most important one for our project.

We also make comparison for our competitors. The score is given from 1 to 5, which implies the satisfaction of each customer requirement. Since all of the three competitors are research based project, they do not aim to build an Android App or focus on UI design. Thus, some of the entries are left blank. After calculation and comparison, we find out that ORB-SLAM performs well on accuracy and respond time. We consider ORB-SLAM is the closest competitor for our project.



	Weight (1-10)	Photo Dimension	UI Color	Time Complexity	Memory	App Size	Accuracy	Response time	3G Data Consumption	Max Moving Speed	Cost	Benchmarks: *Andrew D. single camera SLAM*	*Pierre Martin Decoupled LAM*	*ORB-SLAM*
Accurate	10	3		3	3		9				3	2	3	5
Responds fast	9	3		9				9				5	3	4
UI looks good	5		9			1					1			
Cost small CPU	9	3		9	9				3			3	4	2
App works stable	10				3					9		2	3	3
Adaptable for different weathers	7	9					3							
Shows sign in map	4	3					9			3				
Low cost	8			1	1				1		9	5	2	3
Measurement Unit		px	RGB	BigO	mb	mb	cm	ms/f	mb/s	m/s	rmb			
Target Value		2E5		n^2	300	5	10	100	0.3	10				
Total		159	45	200	149	5	147	81	35	102	107			
Normalized		0.154	0.04	0.19	0.14	0.00	0.14	0.08	0.03	0.10	0.10			
Andrew D. single camera SLAM								25						
Pierre Martin Decoupled LAM		2E5												
ORB-SLAM							1	30						

Figure 2: Quality Function Deployment

3 Concept Generation

The overall project is divided into several stages which are shown in the figure 3. The core of the project is the mapping stage where the *Slamit* will use the data collected to build the map and localize the phone. You could see from the figure that we use ORB-SLAM in this stage. However, at the beginning, we actually got many concepts to choose from. This section will introduce our detailed concept generation process.

Before starting the SLAM development, we did the literature review of state-of-the-art SLAM algorithms. Some SLAM algorithms use laser range data and odometry information such as

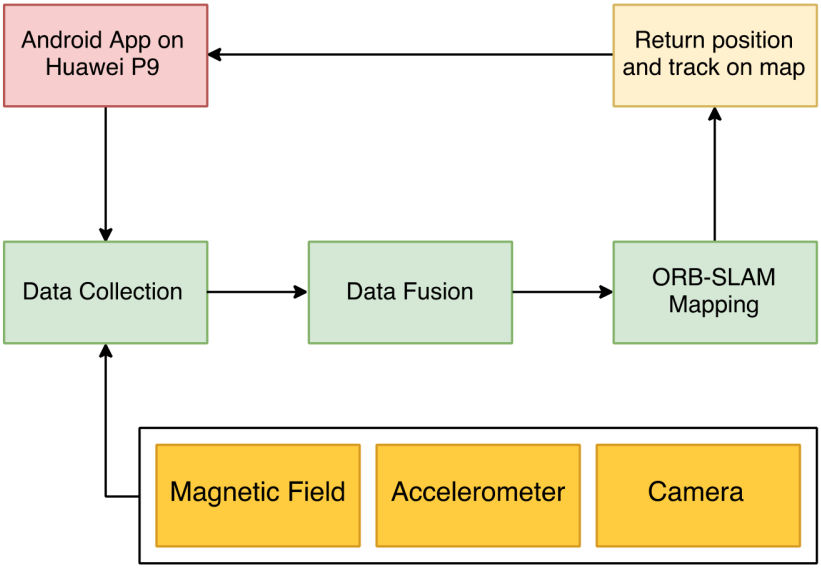


Figure 3: Software architecture of *Slamit*

UFastSLAM[6] and DP-SLAM.[7] These kinds of SLAM are usually very accurate and provide detail grid maps. However, they also require very expensive laser sensors for implementation. Since our project is to design a low-cost SLAM algorithm that can be used on a smartphone, we need to find a solution that is less expensive. After the literature search, we found that MonoSLAM,[4] RGB-D SLAM[3] and ORB-SLAM[5] use images as input data which are much more affordable. A pugh chart is given in table 2.

Design Criteria	Weight	MonoSLAM	RGB-D SLAM	ORB-SLAM	UFastSLAM	DP-SLAM
Accuracy	2	++	+	++	++	++
Process Time	2	+	-	+	+++	++
Required Input	2	0	-	+	---	---
Cost	3	+	0	+	---	---
Generated Map	1	0	+	0	+	+
+		9	3	11	11	9
0		2	0	1	0	1
-		0	4	0	15	15
Total		9	-1	11	-4	-6

Table 2: Concepts Pugh Chart

The further selection process between MonoSLAM, RGB-D SLAM, and ORB-SLAM is introduced in section **??**.

4 Concept Selection

With the three candidate SLAM algorithms, we did further research and list the important factors of the performance of them in Table 3.

	Accuracy (cm) (Indoor) Spec: < 10cm	Process time/ frame (ms) Spec: < 100ms	Required Input	Generated Map
MonoSLAM	~1-2	19	Monochrome images	Feature map
RGB-D	~9	350	Color images + depth images	Colored point cloud
ORB-SLAM	~1-2	11	Monochrome/ Color images	Feature map

Table 3: Performance comparison of SLAM algorithms

Among these algorithms, RGB-D SLAM has best performance in terms of map quality. It can acquire 3D models of objects and scenes and generate a pose graph with colored point clouds. A map generated by RGB-D SLAM is shown in Figure 4. However, the computational cost of RGB-D SLAM is the highest which largely limits its performance on a smartphone. Besides, it is tailored to a RGB-D camera which provides both colored images and depth images, while a normal smartphone such as Huawei P9, is only able to take colored images. Furthermore, its accuracy is about 9cm for indoor environment, which is not as good as MonoSLAM and ORB-SLAM. Its average process time for each frame is 350ms which is also much slower than the other two algorithms. Thus, it is not the best choice for our purpose.

By contrast, MonoSLAM and ORB-SLAM are expected to be a more feasible solution for an android smartphone. They only take monochrome images as input data and have very high accuracy for indoor environment, which is less than 2cm. For the average tracking time, MonoSLAM takes 19ms and ORB-SLAM takes 11ms. Both algorithm generated feature maps as shown in Figure 5 and Figure 6. After comparing the two algorithms, we found that ORB-SLAM has lower-computational cost, thus being able to deal with more features and have better performance in larger outdoor environment.

Figure 4: 3D map generated by RGB-D SLAM[3]

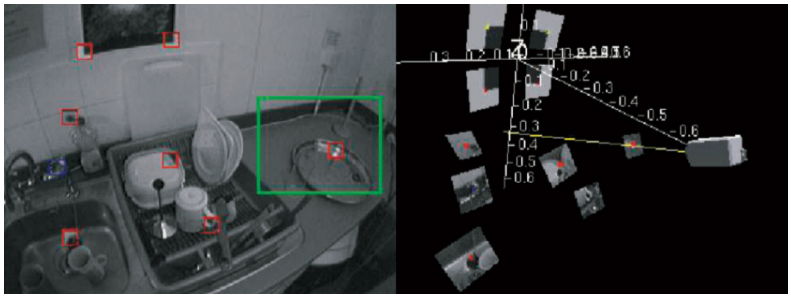


Figure 5: Map generated by Mono SLAM[4]

Based on the reasons above, we decide to use ORB-SLAM as the base visual SLAM algorithm for the implementation on Huawei P9. To increase the localization performance, using Extended Kalman Filter, we will also fuse the inertial data from gyro and GPS to improve the accuracy and reliability. The novelty of our project is that it is realized on Android smartphone platform and combines the visual SLAM and inertial navigation. With our project, it is very possible that smartphone can become a part of the Advanced Driving Assistance Systems in the future.

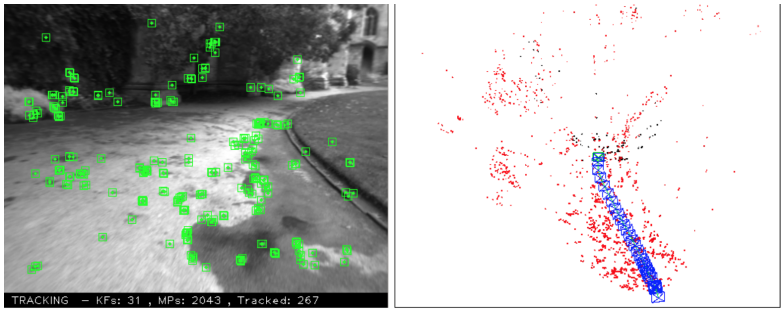


Figure 6: Map generated by ORB-SLAM[5]

5 Concept Description

We designed an Android Application *Slamit* to realize simultaneous localization and mapping. The Application is composed with three subsystems, as shown in Figure 3.

The data collection subsystem will collect the magnetic field data, acceleration data and camera images (or pre-stored images) from the smartphone and send them to the data fusion subsystem. The data fusion subsystem will calculate the orientation (roll, pitch and yaw) of the camera in the world frame and resize the images. It will then send the resized images to the ORB-SLAM subsystem. ORB-SLAM subsystem will localize the camera position and build a local map from the images. Then it will return the map points data and camera position data. *Slamit* will then draw the map points and the camera frame and display position data on the screen. The details of working diagram of *Slamit* is shown in Figure 7.

6 Parameter Analysis

6.1 Camera Calibration Parameters

Camera calibration parameters include the camera focal lengths, optical center, and distortion parameters. These parameters are important for SLAM algorithm to get undistorted image and improve the map building accuracy.

To calibrate the Huawei P9 camera, we took images of calibration chess board and resize it to 544px$\times$408px. The we use openCV calibration programs to calibrate the collected images. We did multiple calibrations and use the average value as our final calibration parameters. Then we put these parameters back into the calibration file in Huawei P9. Figure 8 shows one of the images we used.

Notice that we disable the auto focus function of the P9 camera to keep the camera parameters consistent while running the SLAM algorithm, so we did the same in calibration which causes a little vague on the image.

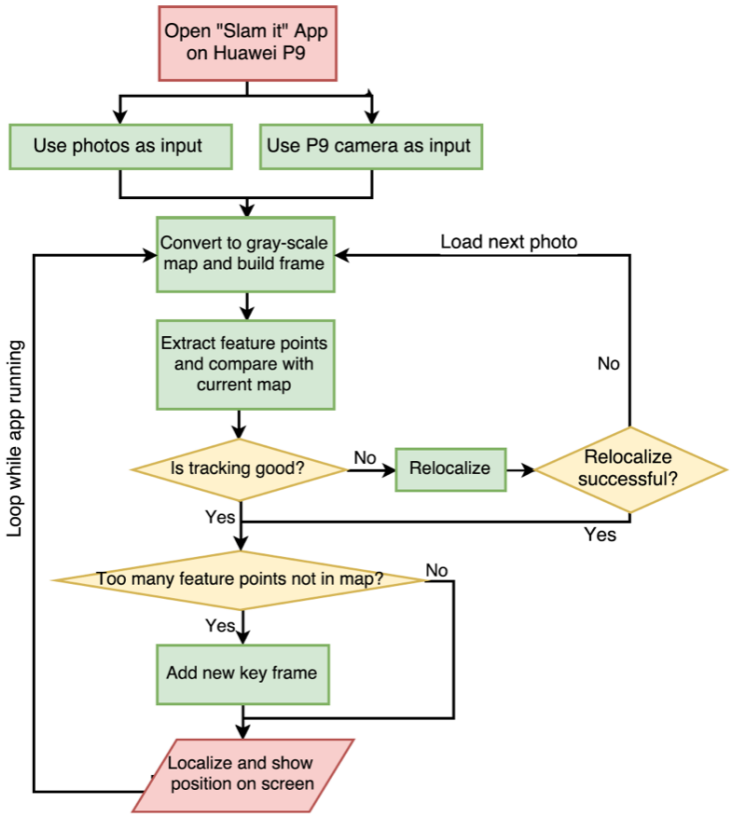


Figure 7: Working diagram of *Slamit*

6.2 Scale Factor

The customer requires the accuracy of the localization to be less than 10cm. However, the map given by the original ORB-SLAM algorithm is a reference map, which means the scale of the map and the camera position are not the same as the ones in the real world. Thus, we need a scale factor which indicates the ratio between the distance in real world and the calculated distance in ORB-SLAM.To get this factor, we introduce the following calibration process.

The basic idea of the calibration is to let the user move along a direction for 0.5 meters. The initial position is recorded as $\vec{x}_i$ and the final position is recorded as $\vec{x}_f$. The scale factor will be calculated as:

$$f = \frac{0.5}{|\vec{x}_i - \vec{x}_f|}$$

After the calibration, all the position data returned by the ORB-SLAM later will be multiplied by the scale factor and displayed on the screen.

In addition to the map scaling, we also want to align the SLAM world frame with the earth frame so that the position displayed is more meaningful. To do so, we collect the orientation data from the smartphone to generate the rotation matrix. Then we multiply the position in the camera frame by the rotation matrix so that the result is the position of the camera in the earth frame.

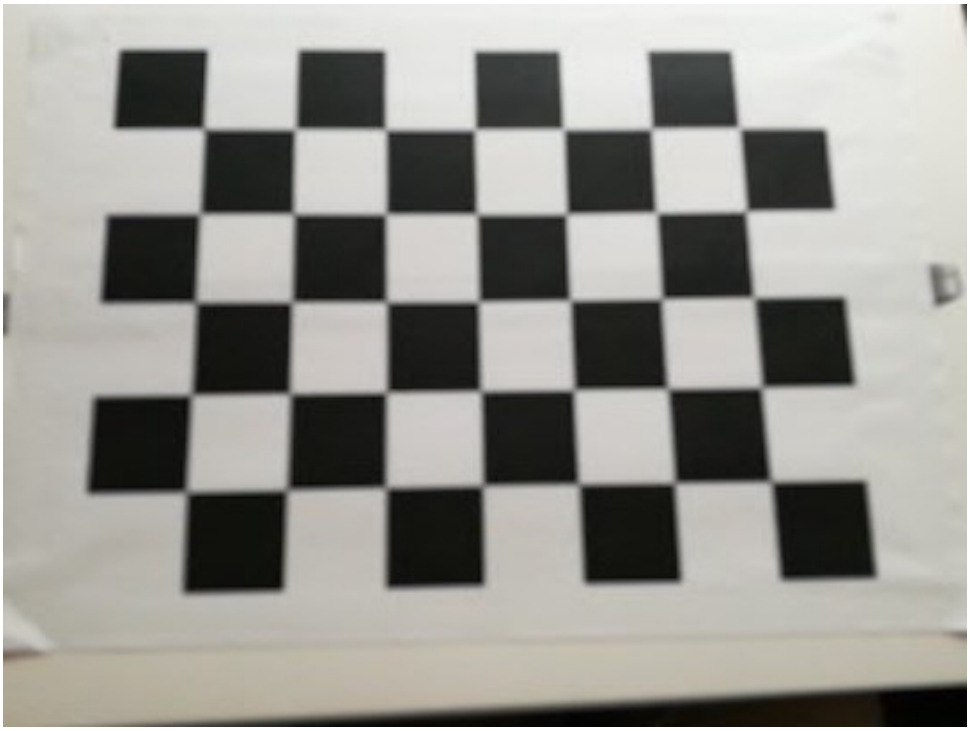


Figure 8: Camera Calibration Image

7 Final Design

Our final design is an android app named *slamit*. We will introduce the design procedure and operating procedure in the following sections.

7.0.1 Design Procedure

After concept selection, we implemented the ORB-SLAM2 on an Android app. Besides the algorithm, we need our app to take photos, get motion sensors data as input.

Literature research and select ORB-SLAM2
We searched and compared different SLAM algorithms and finally chose ORB-SLAM2.

Test the feasibility of ORB-SLAM2 on PC using online datasets
We ran the sample code using the online dataset as input. We also made a dataset by recording video on campus and cut it into a set of pictures. It worked good for both datasets.

Design and build the app layouts and structures
We designed the basic layouts including buttons and views. With these buttons and views, the app can achieve different functions.

Port C++ version ORB-SLAM2 as a core module
The ORB-SLAM2 contained a bunch of C++ files, but we were programming in Java. Thus we used the Java Native Interface (JNI), which allowed us to use C/C++ files in Java. In detail, we used the Android Native Development Kit (NDK) to compile the C++ files into .so files. We

made a java file to use these .so files.

Add OpenCV as another module
We need OpenCV to manage the camera and draw the map. Similar to ORB-SLAM2 code, we used the C++ version OpenCV code. We also used these functions through JNI.

Complete permission requirement for camera and motion sensors
After SDK 23, the permission of camera and sensors are required at runtime instead of installation. The code for permission request is different. We added the permission request for camera and sensors.

Design and complete the code of data fusion to increase accuracy
Only with ORB-SLAM2, we could only get a map without true scale. The scale depends on the focus of camera. We decided to add a calibration process to get the scale. Also, we used the data from accelerometer to further support the calibration process.

Design user interface and test the app
The final step is to make the app look attractive. We designed the app icon (Figure 9) and user interface (Figure 10, 11).

Figure 9: Icon Design

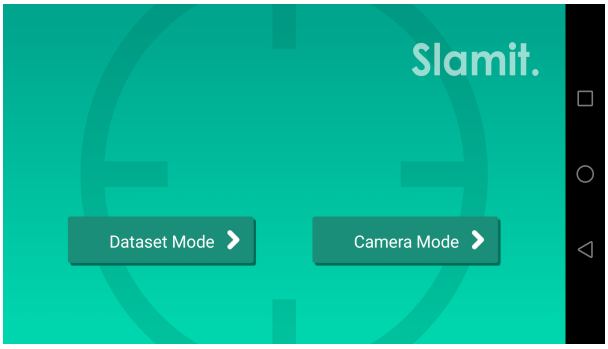


Figure 10: Main Interface

7.0.2 Operating Procedure

- Open *Slamit*
- Click ”Camera Mode” button to begin real time SLAM
- Wait for initialization finished

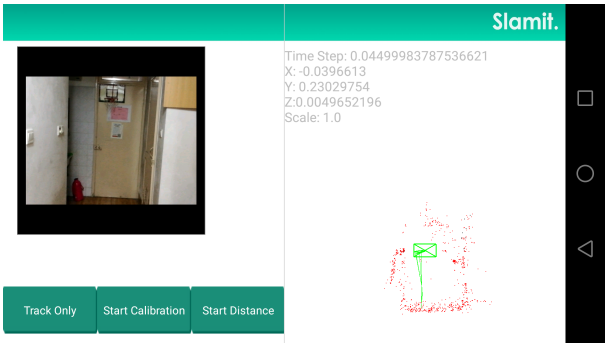


Figure 11: Function Interface

- After building a simple map, click ”Start Calibration” to begin the calibration process. Move the camera for 0.5 meters and click ”End Calibration”. The scale factor is shown on the right.

If the calibration is accurate enough, the location shown on the right is the real location of the camera relative to the origin. The green frame in the map is the camera, and the red points are the feature points. If a point is not in our view, it will shown in black.

If you click ”Track Only” button, the map will stop adding new feature points. It will only do the localization. So it will improve the efficiency and eliminate the time need for calculating each frame. However, if the map does not contain enough points, then it is possible to lose track frequently.

If you click ”Start Distance” button, and move for a distance, then click ”End Distance” button, the distance will show on the top right part of the screen.

If you click ”Start Calibration” button, you need to move for a specific distance (0.5 meters) and click ”End Calibration”. The scale factor will show on the top right part of the screen. We highly recommend you do multiple time for calibration until the value of scale factor is stable.

8 Test Results

Our team decided to evaluate the functions of *Slamit* from four most important aspects, which are localization accuracy, response time, memory occupation, and 3G data consumption. Their results are shown as well.

8.0.1 Localization accuracy

As the core function of *Slamit*, accurate localization is not only of great importance for the slam process, but also for the ultimate goals like path planing and obstacle avoidance. To quantitatively verify the accuracy of *Slamit*, one extra function is added to the app to calculate the displacement of two different positions. As is shown in Figure 11, by clicking the button ”Start Distance”, *Slamit* will record the current position. The displacement will be calculated and shown on screen as we click the button again. Then our team will compare the result with the

actual distance measured by tapeline.

To conduct the experiments, a map will be firstly built and calibrated. Then the smartphone will move for 2 meters for several times and the corresponding displacements will be recorded and the localization accuracy will be calculated.

Currently, the best accuracy we can reach for outdoor environment is about 10% of the moving distance. It is larger than the predicted value, but our team will keep improving *Slamit* for the rest time.

8.0.2 Response time

To realize the simultaneous mapping and localization, the time needed to process one frame, i.e., the response time must be reasonably small. As is shown in Figure 11, our team add the function of calculating the response time into *Slamit* and the result will be shown in the function interface.

To conduct the experiments, we just need to run the app and record the instant response time and then calculate the average value.

The current average response time is 150 ms. Although it is a little bit larger than the expected value, *Slamit* can still achieve simultaneous mapping and localization.

8.0.3 Memory occupation

A qualified android application should not occupy too much memory while running. Our team used the application monitor in Android Studio to trace the memory occupation.

To conduct the experiment, we just need to connect the smartphone with the Android Studio and run the application. The memory occupation will be automatically shown on the screen.

The average memory occupation now is 30 Mb, which is around 1% of the total memory of Huawei P9. This result is much better than our predicted value, 300 Mb.

8.0.4 3G data consumption

Previously, our team planned to add functions like sharing map data and communication with computers into *Slamit*. Under this circumstance, 3G data consumption must be considered since it is related to the cost of this app. However, we currently have not added these functions. In fact, *Slamit* can still work even without Internet connection. In this case, we believe *Slamit* consumes no 3G data.

9 Engineering Changes Notice

From Design Review 3, we made several changes to improve the performance of *Slamit* and make it complete. We also make an effort to realize the map save and load.

We use .bin vocabulary file instead of .txt which reduce the initialization time by 95%. We complete the function of Track Only.Now the first click would disable mapping and the second click would re-enable it.

After final delivery of our sponsor, we realized that the amp save and load is an important function. Thus, we made an effort towards it and successfully realized the map save and load on PC. However, due to time limit, this function is not achieved on the *Slamit*.

10 Discussion

An important advantage of *Slamit* is its low cost on localization. A well designed user interface and reasonable respond time ensures the user experience. The camera view and built map are shown simultaneously on the screen makes the function of *Slamit* delivered clearly.A simple calibration method makes the coordinates calculated in ORB-SLAM related to real world scale. However, due to scale dependent localization error, the accuracy of *Slamit* is relatively large in large map scale. Another limitation of *Slamit* is that it could not save the built map, which makes it less effciecnt in real-time practice. We realized the importance of this function at the end of this semester and we could only achieve it on PC due to time limitation.

For the future, there are several aspects could be improved. First, a fusion of inertial components of smartphone and Rob-SLAM could be used to calculate the map scale automatically. Second, as mentioned before, the map save and load is an important function which is worth to be added into the App.

11 Recommendations

Based on the real time experiments our team conducted, we are confident to deliver the following two parts of recommendations. The first part will focus on how SLAM algorithms could be applied in future driver-less car systems and the another part will be the offline map construction.

11.1 SLAM algorithm in future driver-less car

The performance of ORBSLAM2 algorithm on Huawei P9 is much better than our expectation, especially after we changed the format of "ORBvoc" file from ".txt" to ".bin". This result shows that smartphones are definitely an eligible "sensor" for driver-less cars. Based on this, our team comes up with the following recommendation that smartphones should not only use their embeded cameras, but also those built in the cars. Nowadays many cars have several cameras assembled in their right side and back to keep people safe when turning right and parking. If powerful smartphone like future "Huawei P10" could get access to those cameras and make use of the images, more accurate SLAM of car surroundings can be achieved and our team

believes that Huawei Technology can cooperate with car companies to build the smartphone-car communication. In addition, with the motion sensors assembled in cars, more accurate displacements can be calculated and data fusion can then be combined into the ORBSLAM2 Algorithm.

11.2 Offline map construction

Due to some misunderstandings of the project before the final delivery, our team can only realize the PC-based "save/load map" functions in the last week. The problem we met when implementing the two functions on Huawei P9 is that the we cannot successfully build the "Boost" library in android environment, but we believe it can be solved if given more time. Despite of this, the success of PC-based saving and loading shows that it is feasible to build offline maps through the ORBSLAM2 algorithm.

To illustrate our ideas, some definitions must be firstly given. The offline map we want to construct here is series of data of "mappoints", "keyframes", and their connections rather than a visible 3D map (The connection here means which mappoints are seen by a certain keyframe). The reason for this is that we believe it is much more important to let cars understand a map than build a colorful but useless painting. In addition, "mappoints" and "keyframes" are much easier to save and load than RGB data information and requires less space to store. For example, the original "ORBvoc.txt" file is about 100MB and takes 20s to load, but after we transferred it to a binary file "ORBvoc.bin", it is only about 40MB and takes 1s to load. As a results, we can save all "mappoints" and "keyframes" as binary files and load them when entering the same place.

So how can we build offline map based on these "save/load map" functions? Our team comes up with the following ideas. People can upload the maps their apps built to Huawei Technology everyday. Huawei's engineers can combine these data to update existing maps and construct new maps. At the beginning, Huawei Technology may need to build some "standard keyframes" with high accuracy by some road tests and measurements. Then map data from app users will gradually enrich the map and its accuracy will increase as the data uploaded by users increases. As for combining data from different users, this will be the problem for another project.

12 Conclusions

Due to the various problems on cost and accuracy of current localization methods, smartphone SLAM is proposed to achieve low-cost and accurate localization. With the ORB-SLAM as the core algorithm, our final design, *Slamit*, successfully meets the specifications on cost, app size, CPU occupation, 3G data consumption. The accuracy and respond time is a little below the specified value. Due to time limitation, the map save and load is only realized on PC. For the future, we recommend more work localization accuracy and map save and load.

13 Acknowledgements

The completion of *Slamit* could not have been possible without the help of Mr. Gang Ye and Prof. Chengbin Ma. As the direct supervisor from sponsor Huawei Technology, Mr. Ye has

provided us valuable advice on technical issues, and our course instructor,Prof. Ma, gives us a great help on project management.

In addition, we also want to give our sincerely acknowledgement to Ms. Dannuo Feidi and Ms. Yufei Chen for their profession advice on our video cutting and UI design.

We thank you.

A Camera Calibration

Here is a copy of the camera calibration .yaml file we used for *Slamit*

- list3.yaml

```
# Camera Parameters. Adjust them!


# Camera calibration and distortion parameters (OpenCV)
Camera.fx: 564.930718
Camera.fy: 569.571047
Camera.cx: 271.5
Camera.cy: 203.5

Camera.k1: 0.436844
Camera.k2: -3.336257
Camera.p1: -0.005644
Camera.p2: -0.001433
Camera.k3: 9.021231

# Camera frames per second
Camera.fps: 30.0

# Color order of the images (0: BGR, 1: RGB. It is ignored
   ↪ if images are grayscale)
Camera.RGB: 1


# ORB Parameters


# ORB Extractor: Number of features per image
ORBextractor.nFeatures: 1000

# ORB Extractor: Scale factor between levels in the scale
   ↪ pyramid
ORBextractor.scaleFactor: 1.2

# ORB Extractor: Number of levels in the scale pyramid
ORBextractor.nLevels: 8

# ORB Extractor: Fast threshold
# Image is divided in a grid. At each cell FAST are
   ↪ extracted imposing a minimum response.
# Firstly we impose iniThFAST. If no corners are detected
   ↪ we impose a lower value minThFAST
# You can lower these values if your images have low
   ↪ contrast
```

```
ORBextractor.iniThFAST: 20
ORBextractor.minThFAST: 7


# Viewer Parameters

Viewer.KeyFrameSize: 0.05
Viewer.KeyFrameLineWidth: 1
Viewer.GraphLineWidth: 0.9
Viewer.PointSize:2
Viewer.CameraSize: 0.08
Viewer.CameraLineWidth: 3
Viewer.ViewpointX: 0
Viewer.ViewpointY: -0.7
Viewer.ViewpointZ: -1.8
Viewer.ViewpointF: 500
```

B Bill of Materials

he only hardware we used are a Huawei P9 smartphone. Table 4 gives details of it.

Item	Quantity	Catalog Number	Cost
Huawei P9	1	EVA-AL10	3,688

Table 4: Bill of Materials

References

[1] “”lost in tracking” or why mobile gps is inaccurate?.”

[2] V. Mike, “Lidar: Coming soon at a price you can afford (part 2),” 2015.

[3] F. Endres, J. Hess, N. Engelhard, J. Sturm, D. Cremers, and W. Burgard, “An evaluation of the rgb-d slam system,” in *Robotics and Automation (ICRA), 2012 IEEE International Conference on*, pp. 1691–1696, May 2012.

[4] A. J. Davison, “Real-time simultaneous localisation and mapping with a single camera,” in *Computer Vision, 2003. Proceedings. Ninth IEEE International Conference on*, pp. 1403–1410 vol.2, Oct 2003.

[5] R. Mur-Artal, J. M. M. Montiel, and J. D. Tardós, “ORB-SLAM: a versatile and accurate monocular SLAM system,” *CoRR*, vol. abs/1502.00956, 2015.

[6] C. Kim, H. Kim, and W. K. Chung, “Exactly rao-blackwellized unscented particle filters for slam,” in *Robotics and Automation (ICRA), 2011 IEEE International Conference on*, pp. 3589–3594, May 2011.

[7] A. Eliazar and R. Parr, “Dp-slam: Fast, robust simultaneous localization and mapping without predetermined landmarks,” in *Proceedings of the 18th International Joint Conference on Artificial Intelligence*, IJCAI’03, (San Francisco, CA, USA), pp. 1135–1142, Morgan Kaufmann Publishers Inc., 2003.