Simultaneous Localization And Mapping (SLAM) is useful when a robot explores an unknown environment. Since this ability is required in many real-world applications, SLAM is an important topic in robotics. Visual SLAM is solving the SLAM problem by visual information such as images or video. Visual SLAM is important for robot navigation, but it can also be applied to mobile devices, such as smart phones or wearable devices like Google Glass. Indoor visual SLAM is different from conventional outdoor visual SLAM, since there are many occlusions within close range and moving landmarks in indoor scenes. Due to the fast motion of human and poor indoor illumination, significant motion blur also imposes a challenge for indoor SLAM. With the help of various sensors equipped in mobile devices such as accelerometer, gyroscope or Inertial Measurement Unit (IMU), we can approximate and refine the trajectory of the mobile device while reconstructing the indoor environment.

Using mobile devices to do SLAM can be very beneficial to many scenarios. For example, people can use their smartphones to record their surroundings in a museum or gallery. Moreover, our system can also help blind or visually-impaired personnel record their track and detect dangerous environments, such as potholes or stairs. If SLAM can run real-time on mobile platforms, it will also have real application prospects on micro robots whose computing ability is constrained by size.

II. RELATED WORK

Much work has been done to achieve indoor visual SLAM in recent years. Lee and Song propose autonomous detection of objects as visual landmarks for visual SLAM using an IR camera and RGB camera [5]. Hoffmann et al. present an autonomous mobile robot setting that automatically explores and maps unknown indoor environments. Their method needs additional information from an embedded event-based dynamic vision sensor and a ring of bump switches on the robot [3].

III. APPROACH

Our work is inspired by Parallel Tracking and Mapping (PTAM)\(^1\). PTAM is a camera tracking system for augmented reality, which has very fast feature tracking and matching ability and can be run real-time on mobile device. PTAM uses parallel-thread architecture, which has two threads asynchronously running at the same time: the first thread called “Tracker”, processes vision-related tasks, such as feature extraction and matching; and the second thread called “Mapper”, does Bundle Adjustment and builds the real-time landmark map. PTAM can run on mobile devices such as iPhone, but it’s sensitive to large motion and its map is fragile if the device’s motion is too large. Our system has a similar parallel-thread architecture, including “Tracker” and “Mapper” threads, however, we utilize an IMU to improve the performance. Figure 1 shows the workflow of our SLAM.

The “Tracker” thread takes input frames from camera, extracts features between two key frames and matches features, and then puts the features association information to the “Mapper” thread. The “Mapper” thread gets input both from raw sensor data from IMU and features projection information from “Tracker” thread, then solves the SLAM problem. In our “Mapper” thread, we use a factor graph as our tool to solve the SLAM problem.

Since the camera works at relatively lower frequency (about 20 to 30 fps for regular camera) than IMU (faster than 100Hz for regular IMU device), ”Mapper” thread will work at higher frequency than “Tracker” thread, and they are asynchronous. Since they are in one process and share the same address space, it’s easy to build the data pipeline between two threads. Considering more and more CPU on mobile devices such as smart-phone are Dual-core processors, parallel-thread architecture can utilize the CPU resource better.

A. Hardware

We build a mobile device prototype that equipped with a webcam and an IMU with a 3-axis accelerometer and a 3-axis gyroscope. The webcam we choose is Logitech HD Pro Webcam C920 with 1920x1080 resolution. The webcam is set to work at 640x480 resolution with frame rate at 30 fps. The IMU we use is ArduIMU V3. ArduIMU V3 has

\(^1\)http://www.robots.ox.ac.uk/~gk/PTAM/
one 6-axis inertial sensor MPU-6000 and one 3-axis compass HMC-5883. It also has one Arduino compatible ATmega328P micro-processor, and we modified its firmware to obtain high-frequency output. The IMU is set to work at 200Hz. The device was mounted on the head to get first-person view video. Figure 2 shows the hardware system we built.

B. Feature Extraction and Association

In feature extraction PTAM uses FAST [7], [6] feature detector, which has high response rate, yet the result is noisy and unstable. There are many new feature detectors and descriptors invented recently, such as ORB, BRISK, FREAK. Since our approach is designed to run in real time on embedded platform, we need to balance the efficiency and quality. We choose ORB feature, which is derived from FAST and BRIEF [1]. It is rotation invariant, resistant to noise, and efficient enough for embedded computing.

After we extract ORB features from images, we use Hamming-distance brute-force matcher to do initial descriptor matching. Therefore descriptors in two key frames can be roughly matched, but there are still some outliers in them. To further fine-tune the matched features, we use RANdom SAmple Consensus (RANSAC) [2] to filter all mismatched feature points. In this case, a fundamental matrix between two key frames can be generated as long as eight or more feature points are matched in both frames. Figure 4 shows matched features in two consecutive key frames after RANSAC.

When adding projection information to the factor graph, if the landmark is determined by a triangle with very small baseline, the estimated landmark positions will contain very large error.

We use OpenCV Library\(^2\) to build our “Tracker” thread. C. Factor Graph and iSAM2

To recover the trajectory of the mobile device and reconstruct the 3D environment, we use factor graph as our basic tool. GTSAM\(^3\) is a library of C++ classes that implement smoothing and mapping (SAM) in robotics and vision. It uses factor graphs and Bayes networks as the underlying computing paradigm.

In visual SLAM problem we need to solve for camera pose and landmark position. We use variable symbol \(x\) to represent the pose of camera, so that the initial pose of camera is \(x_0\), and second is \(x_1\), etc, and \(l\) to represent the position of landmarks.

To constrain variables, we put several kinds of factors in the graph. The first constraint that we impose is IMU factor, which is already implemented in GTSAM. It constrains camera pose, camera velocity, and IMU internal bias. To use this factor, we need to add two "hidden" layers in the graph — velocity and bias. Velocity is represented as symbol \(v\) and bias is represented as symbol \(b\). IMU factors give the probability of given neighbouring poses, velocity and bias like \(P_i(\vec{a}_i, g_i | x_i, x_{i-1}, v_i, v_{i-1}, b_{i-1})\). \(\vec{a}_i, g_i\) are measurements of IMU.

Since the sensor’s bias cannot change rapidly, there is a constraint between neighbouring biases. We put a factor between biases, and its probability is \(P_i(b_i, b_{i-1})\). we also put prior factor on \(x_0, v_0, b_0\) to constrain the extra freedom, whose probability is \(P(x_0), P(b_0), P(v_0)\). To Constrain landmarks’ position, we apply a pinhole-camera projection factor. For landmark \(l_j\), measurement (feature position in the frame) \(z_k\) in the frame taken in \(x_i\), and we have a factor whose probability is \(P_k(z_k | x_i, l_j)\).

The final graph is shown in the Figure3. To solve this problem and get \(x_i, l_j\), we need to optimize the joint probability shown in equation(1).

\[
\arg \max_{x,l} P(x_0)P(b_0)P(v_0) \prod_i P_i(b_i, b_{i-1}) \prod_k P_k(z_k | x_i, l_j) \\
\prod_i P_i(\vec{a}_i, g_i | x_i, x_{i-1}, v_i, v_{i-1}, b_{i-1})
\] (1)

Because the graph will be bigger and bigger during time, spending time of traditional method will be increase fast. To reduce the effect of increasing graph size, we utilize the iSAM2 [4] module in GTSAM, which is Incremental Smoothing and Mapping by Bayes Tree. It use Bayes Tree

\(^2\)http://opencv.org/

\(^3\)https://borg.cc.gatech.edu/download
as basic math tool to deal with incremental information and significantly reduce the consuming time of solve factor graph.

IV. Evaluation

To test our system, we create a simple scene with several objects presenting in it, such as books on the desk. The dataset contains 1849 IMU data and 271 images. Figure 5 shows the sample key frames of the scenario.

![Figure 5: Sample key frames of the scene we built for test experiment.](image)

Figure 6 shows the result of our method, including a comparison between results by IMU integration by IMU data only. The world coordinate’s origin is the starting point, and Z axis is parallel to gravity. Since all objects in the scene we capture are on a flat desk, the landmarks (black dots in the Figure 6) should be a plane parallel to X-Y plane. However, the landmarks are not exactly in the same plane, and it looks like a "cluster" instead, which means our result still has a lot of noise.

![Figure 6: Trajectories Result. Purple trajectory is by our method, cyan trajectory is by integration of IMU data only. Camera’s poses are shown, red/green/blue means x/y/z axis of the camera, and black dots are landmarks.](image)

Since our method is designed for real-time application, we should analyze efficiency of our method. First the ORB feature detecting/matching and RANSAC takes about 30 to 40ms per step. Considering the frame-rate could be lower than 30fps to make sure very key frame have enough motion, the "Tracker" thread can be real-time on laptop. Second the time consuming of iSAM2 of "Mapper" thread is shown if Figure 7. The time needed every step is increasing because the graph is being larger and larger. The last several steps need more than 10ms, means it’s not real-time at 200Hz. If we want to use it in real-time application, we should lower the IMU frequency and limit the map’s size.

![Figure 7: Time consuming of every step in iSAM2.](image)

V. Discussion

We propose a monocular SLAM algorithm and a portable device that can reconstruct 3D environment and recover camera pose, and we implemented state-of-art methods to improve the speed and accuracy of the system.

First our method is quite unstable in certain scenarios, such as forward moving and keeping still for a long time. Second is our method is not real "real-time" now. Although it can run real-time for several seconds on laptop, it’s still too heavy for mobile CPU, and the method cannot stable work for more than several seconds.

Since micro IMU’s output is noisy due to change of environment, such as temperature. The cyan line in Figure 6 shows the trajectory output by IMU. It still gives us a trajectory, but it is significantly effected by the drift of the sensors. Visual information can significantly reduce the side-effect of sensors drift, and improve the trajectory result since visual information of landmarks has no drift caused by time and environment.

In the future we will try more accurate feature extracting and matching method, especially for feature descriptor since the ORB feature descriptor has low inlier rate. We will also refine our graph model, make it compact and more solvable in a certain map size, and try more incremental SLAM solver other than iSAM2 to get better timing performance in large-scale problem.

REFERENCES


