1 Introduction

3D reconstruction is the process of capturing the shape and appearance of real objects. The conventional methods for 3D reconstruction is based on *multiple images*. In this method, 3D models are constructed from a set of images. It can be considered as a reverse process of obtaining 2D images from 3D scenes.
3D reconstruction has a wide variety of applications. It allows engineers and students to work more efficiently as they can generate copies of 3D objects easily. In orthopedics and joint replacement surgery, the complicated parts to be replaced can be made with high accuracy. Also, it can be used to generate replica of artifacts and fragile objects for further studies, without harming its integrity.

(a) Sparse reconstruction (b) Dense reconstruction

Figure 1: 3D reconstruction

Figure 9d shows the first step towards 3D construction i.e. *Sparse reconstruction*, which is done for some set of points. Figure 9h is the final reconstructed 3D model of the object. The *traditional pipeline* is shown in Figure 2.

The red-highlighted part of the pipeline is ***computationally expensive***. Thus, our project aims to reduce this computation and perform 3D reconstruction in near real-time.

The processing parts are:

- *Intrinsic and extrinsic parameters*: The camera projection matrix is a 3×4 matrix which represents the pinhole geometry of a camera for mapping 3D points in the world coordinates to 2D points on images. This matrix depends on extrinsic and intrinsic parameters. The intrinsic parameters mainly comprises of focal length, image sensor format, and principal points. The extrinsic parameters define the position of the camera center and the camera's heading in world coordinates in terms of a rigid rotation and translation.

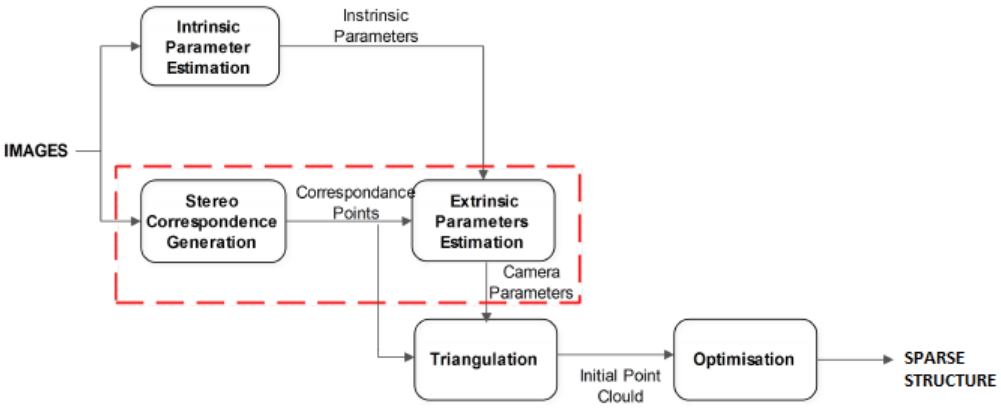


Figure 2: Traditional pipeline

- *Stereo correspondence generation*: Given two or more images of the same 3D scene, taken from different points of view, the correspondence problem refers to the task of finding a set of points in one image which can be identified as the same points in another image. To do this, points or features in one image are matched with the corresponding points or features in another image. The images can be taken from a different point of view, at different times, or with objects in the scene in general motion relative to the camera(s).

- *Triangulation*: Triangulation refers to the process of determining a point in 3D space given, its projections onto two or more images and their corresponding camera projection matrices. This point is found as the intersection of the two or more projection rays formed from the inverse projection of the 2D image points representing that 3D point in space.

- *Initial point cloud and 3D sparse reconstruction*: As the word suggests, *3D sparse construction* is done for only some set of data points in the given coordinate system called *initial point cloud*. Figure 9d illustrates a 3D sparse construction of a chariot. Figure 9h illustrates 3D dense construction of the same initial point cloud.

2 Basic Concepts

2.1 Camera calibration

The camera parameters can further be subdivided into intrinsic and extrinsic parameters. **Camera intrinsic parameter** K is dependent on the focal length of the camera and principal point (which in most cases is the center of the image). The **camera extrinsic parameter** is composed of the rotation R

and translation t between camera coordinate system and the world coordinate system. Together they form the camera projection matrix P, a 3×4 matrix which describes the mapping of a pinhole camera from 3D points in the world to 2D points in an image.

$$P = K[R|t] \tag{1}$$

2.2 Sparse 3D reconstruction

Given two different images of the same scene from different angles, the position of a 3D point can be found as the intersection of the two projection rays which is commonly referred to as **triangulation**. For this first point correspondences have to be established. Then using this point correspondences a Random Sampling Consensus (RANSAC) based voting framework is used to estimate the camera intrinsic and extrinsic parameters. Finally, a joint non-linear optimization is used to further refine the camera parameters and the 3D points in a **bundle adjustment** framework. This method is computationally very expensive and hence done only for very sparse set of points. This is known as sparse 3D reconstruction.

3 Mobile IMU sensors

IMU (Inertial Measurement Unit) sensors are on-chip devices embedded in most of the smart phones or hand-held devices today. It mainly consists of a series of motion sensors: accelerometer, gyroscope, magnetometer and gravitation sensor. The data from these sensors can be fused to obtain the orientation and the position of the device in the world coordinate system.

4 Conventional versus mobile 3D reconstruction

In the case of a smart-phone or any hand-held device having a camera and IMU sensors, we wish to use the IMU sensors to obtain extrinsic camera parameters in real time. This will help in reducing the load on conventional 3D reconstruction methods and get it in near real time.

5 IMU Sensor Processing

"To get accurate position and orientation estimate based on readings of IMU sensors in smart-phones."

5.1 Initialization

Started by making an Android application on Android Studio. The following sensor were deployed:

- Accelerometer
- Gravity
- Gyroscope
- Magnetic Field

The application consisted of five buttons on the screen with the view from camera on the background

- Start: To begin the process of data collection from sensors and video capture from the camera
- Quit: To quit the application
- Image Click: To take various images of the concerned object after pressing Start
- Clear: To clear Cache
- About: About the application

On pressing the start button, the following data is received from the sensors and stored for further processing. Writing of data is done when the gravity sensor is active since it is the most frequent sensor.

- Acceleration: Raw acceleration data is quite crude and cannot be used directly and hence we needed to further process. Crude and processed data are compared in the results section.
- Magnetic field
- Gravity
- Gyroscope
- Rotation Matrix from inbuilt Android function using gravity and magnetic field (Calculated)- SensorManager
- Rotation Matrix from gyroscope (Calculated)
- ImageID corresponding to this data

5.2 Data Processing

:
Calculation of Rotation is started in parallel with collection of data. After all the data is collected, the following are done (in order):

- Acceleration Frame is changed from Ground Frame to Camera Frame. The produced acceleration is with respect to a static ground frame which is converted to camera frame using the Rotation matrix. Since we have two Rotation matrices available i.e. Default SensorManager and Gyroscope, thus an input was taken to choose the matrix
  Procedure:

```
1          def ChangeFrame(typeofmatrix,A_old):
2            # R and A_old are rotation and acceleration
     matrices respectively
3            R <- choose the appropiate matrix
4            A_old <- A_old*R
5
```

Listing 1: ChangeFrame()

- Gravity is removed from the acceleration data. Since the acceleration data also consisted of gravity, it needed to be removed before further processing it.
  Procedure:

```
1          def RemoveGravity(A_old, Gravity):
2            #A_old and Gravity are acceleration and gravity
     matrices respectively
3            A_old <- A_old - Gravity
4
```

Listing 2: RemoveGravity()

- *Static Bias Removal*:
  Assuming the device was at rest before the first image was clicked, static bias in the acceleration is removed. First, the mean of acceleration values before the first image was clicked is calculated and then subtracted from all the acceleration values. Note that image id is 0 before the first image is clicked.
  Procedure:

```
1          def StaticCorrection(A_old):
2
3            Tot_Acc <- 0
4            Num <- 0
5            #Sum of acceleration values with id=0
6            for acc in A_old having id=0:
7              Tot_Acc <- Tot_Acc + acc
8              Num <- Num + 1
9
10           Tot_Acc <- Tot_Acc/Num
```

```
11
12              A_old <- A_old - Tot_Acc
13
```

Listing 3: StaticCorrection()

- *Smooth Acceleration*:
  Two types of smoothening techniques were tried i.e Low-Pass Filter and 'LOESS' local regression. 'LOESS' local regression produced better results and hence was incorporated.
  Procedure:

```
1           def LowPassFilter(A_old, alfa):
2
3             for index in (1,A_old.size()):
4               A_old[index] <- A_old[index] + (1 - alfa) *
5                                       A_old[index-1]
6
```

Listing 4: LowPassFilter()

Procedure:

```
1           def LocalRegression(A_old, alfa):
2
3
```

Listing 5: LocalRegression()

- *Motion Zones Identification*:
  A data point was said to be a motion point if in its neighborhood lied a point having significant acceleration compared to the standard deviation of the acceleration points before the first image was clicked. All the contagious data points were clubbed together and identified as zones. Points outside these zones having non-zero acceleration were declared as rest points and hence noise was removed.

  Procedure:

```
1           def MovingPoint(std_dev, A):
2
3             #Array of same size as A having False values
4             mov_points <- [False] * (A.size)
5
6             for index in (0,A.size()):
7               if there exists a point in neighborhood of
    length 50 of index having A[nbh] > std_dev:
8                 mov_points[index] <- True
9
10            return mov_points
11
12          def MotionZones(A_old, alfa, std_dev):
13
14            mov_points <- MovingPoint(std_dev, A)
15
```

```
16          #compress all the contagious True Values in
    mov_points. Starting with empty array
17
18          mot_zones <- []
19
20          for index in (0, A.size()):
21            a,b <- find end indices of next contagious
    region of True values in mov_points
22
23            mot_zones.append(a,b)
24
```

Listing 6: MotionZones()

- *Velocity and Distance Calculation*:
  Velocity and distance were calculated taking account only the accelerations in the identified motion zones. First, velocity is calculated using Reimann Sums of Acceleration.
  Velocity Drift Correction: For each zone, to ensure that the velocity at the end is 0, a quadratic equation is subtracted having its end points at the beginning and end of the zone.
  Also, Distance is calculated using Reimann Sums of this Velocity
  Procedure:

```
1         def Velocity(A, delta_t):
2
3           #Calculating velocity using Reimann Sums
4           #Array of same size as of Acceleration(A), giving
    velocity at that point
5           velocity <- [0]*A.size()
6
7           for motion_zone in mot_zones:
8             (a,b) <- motion_zone:
9             tot_time <- 0
10            for index1 in (a,b):
11              velocity[index1] <- velocity[index1-1] + A[
    index1] * delta_t[index1]
12              tot_time <- tot_time + delta_t[index1]
13
14            #Subtracting a quadratic equation starting and
    ending at the end points of a zone
15            final_v <- velocity[b]
16            time_elap <- 0
17            m <- 2 * final_v / (tot_time * tot_time)
18
19            for index2 in (a,b):
20              velocity[index2] <- velocity[index2] - 0.5*m*
    time_elap*time_elap
21              time_elap <- time_elap + delta_t[index2]
22
```

Listing 7: Velocity()

Procedure:

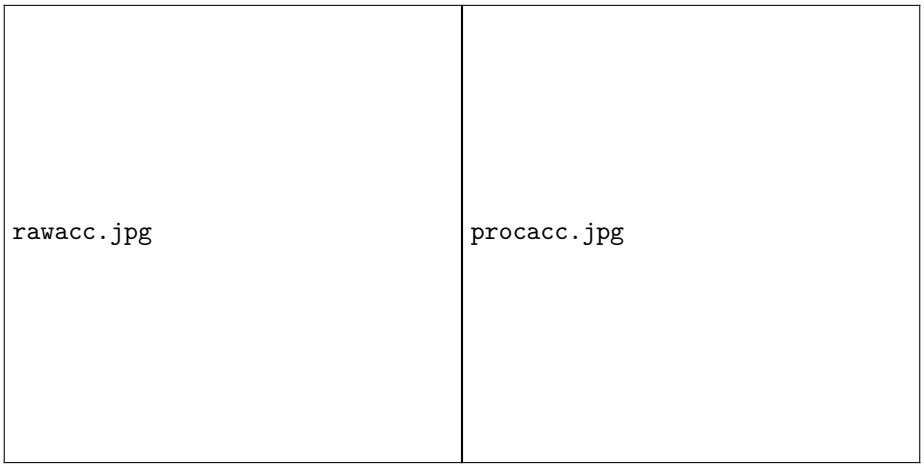
```
        def Distance(V,delta_t,mot_zones):

          #Calculating distance using Reimann Sums
          #Array of same size as of Velocity(V), giving distance upto that point
          distance <- [0]*V.size()

          for index in (1, V.size()):
            distance[index] <- distance[index-1] + V[index] * delta_t[index]

```

Listing 8: Distance()

5.3 Results:

The following results were compiled for **Xiaomi Mi4**.

- Figure 3 shows two graphs show smoothening of acceleration. The first graph represents raw value of acceleration from the sensors. The second graph shows the processed values of acceleration using our procedure.



(a) (b)

Figure 3: Acceleration Smoothening

The second graph has smoothened the kinks in the first graph thus our procedure is very successful in removing noise from raw acceleration data.

- Figure 4 shows two graph represents the value of distance obtained direct integration and our procedure for 30cm movmement for different trials

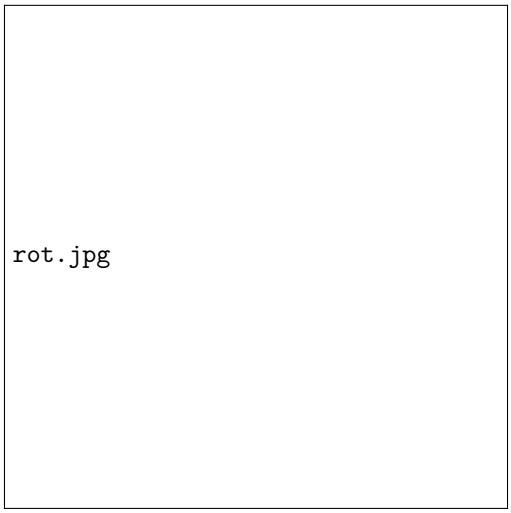
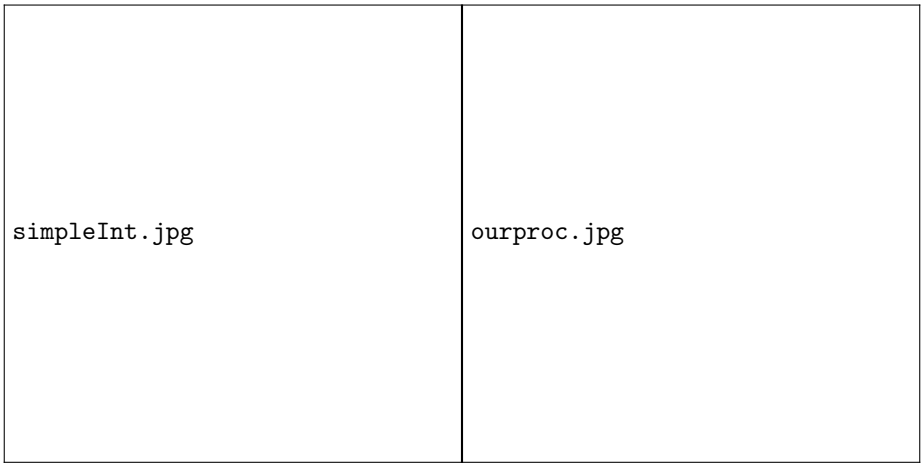


Figure 5: Rotation Values



(a) (b)

Figure 4: Distance Estimations

The above graphs show that our procedure clearly gives more reliable results.

- Figure 5 shows a graph which represents the value of rotation obtained from our procedure against various ground truths. As can be seen from the graph, that the values obtained are very close to the ground truth. Thus, this shows that our procedure is highly accurate.

Figure 6: Motion Zones

- Figure 6 shows a graph identifying the motion zone using our procedure. The detection of the motion zone is highly accurate and efficient using our procedure

6 Generating point correspondences

We exploited the video feed from the camera to generate correspondences through visual tracking of points. It was done in the following way.

1. Feature points were identified in the first frame using Good features to track.
2. These points were tracked in the following video frame using sparse optical flow (KL Tracker)
3. As a result, from the first frame a chain of points was established till the next image was captured and hence correspondences were identified.
4. Due to quick movement taking place, the number of tracked points would drop. To ensure that this is corrected, more feature points were added after every 10 frames so that enough points are available.

6.1 Results

- The above procedure works in real time on the device.

- The correspondences obtained are multi image correspondences.
- To identify the frame rate required for this method to work, an off-line program was run on various frame rates. It was seen that frame rate above 6-7 was able to produce accurate correspondences.
- The average number of correspondences obtained from a dataset of images is as follows:

Number of Images	Average Number of Correspondences
2	TODO
3	TODO
4	TODO
5	TODO
6	TODO
7	TODO
8	TODO

- The quality and number of correspondences can be improved by incorporating a kalman filter to allow points dropped in a frame to re enter the procedure.
- The correspondences obtained contain some errors which can be seen in the following:

Figure 7: Incorrect Correspondence

Figure 8: Correct Correspondence

7 3D Reconstruction

Using the correspondences and position orientation data obtained previously, 3D reconstruction is performed.

1. Conversion the Orientation Matrix (Mobile Sensors Frame) to Rotation Matrix (Camera Frame)

$$\mathbf{R} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & 1 \end{bmatrix} * \begin{bmatrix} 0 & 1 & 0 \\ -1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix} * \begin{bmatrix} R_0 & R_1 & R_2 \\ R_3 & R_4 & R_5 \\ R_6 & R_7 & R_8 \end{bmatrix} \tag{2}$$

   - The first matrix brings a flip of the y axis.
   - The second matrix aligns the x and z axis of the frame of sensors with the camera frame.

2. Conversion of Position Matrix to Translation Matrix

$$\mathbf{T} = -\mathbf{R} * \mathbf{C} \tag{3}$$

   The above equation changes the position of the camera in ground frame to the translation in the frame of the image.

3. Triangulation

   - The intrinsic camera matrix is precomputed from the focal length of the camera. For the case of Xiaomi Mi4 the matrix is as follows:

$$\mathbf{K} = \begin{bmatrix} 8.695 * 10^{-4} & 0 & 0 \\ 0 & 8.695 * 10^{-4} & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{4}$$

   - Using equation 1 and the R, T, K matrices for a given set of corresponding points, a system of linear equations is solved to obtain the 3D coordinates of the point.

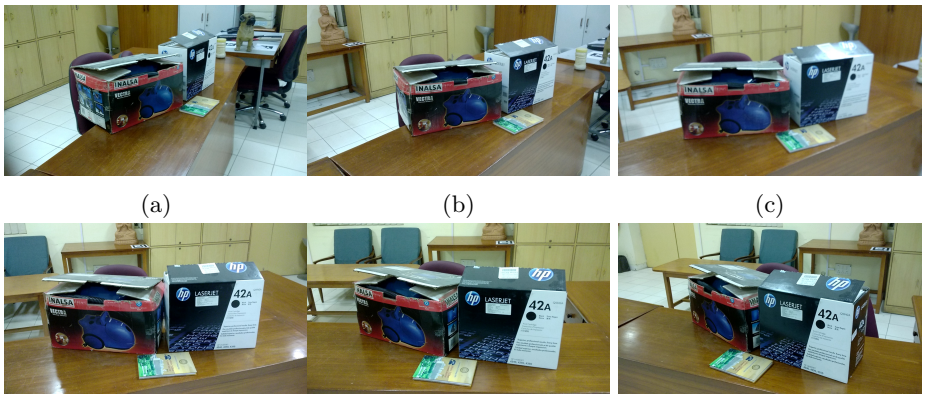
4. Bundle Adjustment
   After the triangulation has taken place from the estimated R,T matrices, a bundle adjustment is done to improve the 3D point coordinates, rotation and translation matrices to yield the final result.

7.1 Results

1. The accuracy of the rotation and translation matrix obtained from the IMU sensors compared against the R,T matrix obtained from computer vision techniques is as follows:

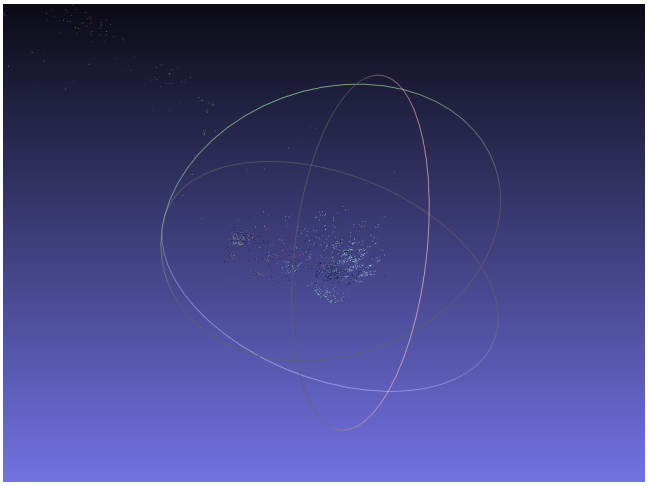
S.No.	No. of Images	Avg. Error in Translation	Avg. Error in Rotation
1	TODO	TODO	TODO
2	TODO	TODO	TODO
3	TODO	TODO	TODO
4	TODO	TODO	TODO
5	TODO	TODO	TODO
6	TODO	TODO	TODO
7	TODO	TODO	TODO

2. Consider the images shown

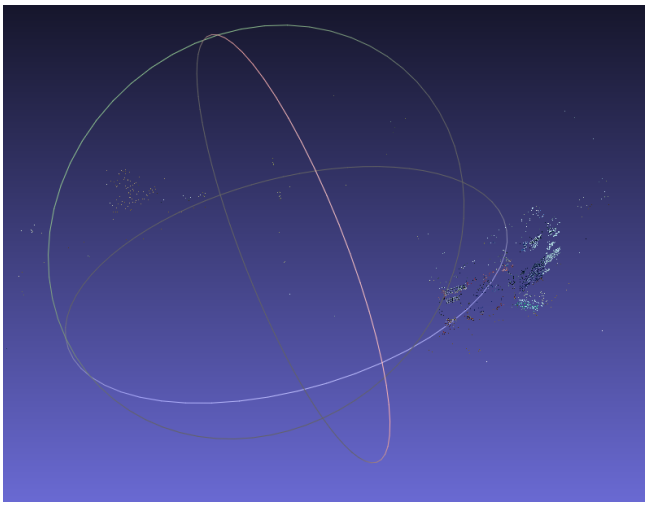


(a) (b) (c)

(d) (e) (f)



(g) Reconstruction from sensors



(h) Bundle Adjustment

Figure 9: Images

- Figures a-f capture the images of 2 boxes placed on a table from different positions
- Figure g is the output of triangulation using the information obtained from tracking and sensors. The points can be seen to represent a rough structure of the boxes shown in the images. Different set of layers can be observed which are supposed to represent the same flat surface.
- After running bundle adjustment, the 3D structure of the boxes can be visualized clearly in Figure h

8 References