Visual SLAM Using Monocular Camera

Ty Nguyen, Rachit Bhargava and Haoyuan Zhang

***Abstract*— The report mainly contains six parts, introduction, problem formulation, technical approach, training results, experiments with g2o and conclusion.**

We implement this visual SLAM based on the common SLAM which is based on lidar sensor data. Instead of using scan information from lidar data, we choose visual information to update the pose of camera as well as the 3D point cloud with respect to the global frame.

Eventually, we expect to build a 3D map with correct estimated trajectory of robot and 3D point cloud respected to the physical world.

I. INTRODUCTION

SLAM is one of the most common but also difficult problems in the Robotics realm. Nowadays, people come up with multiple approaches to perform SLAM well and apply that into various applications, such as car or drones. Due to the cost of lidar sensor, these days vision based SLAM plays more and more important roles which is efficient and also has good accuracy.

In the visual SLAM, feature plays a pretty important role. Unlike use scan data to locate and update the location of robot as well as implement mapping, we use feature information to achieve SLAM. Basically, we detect features for each frame and then track some of them in multiple camera frames, since those tracked feature represent the same physical point in 3D world, therefore, its reasonable to estimate both camera state and 3D point cloud via linear/nonlinear triangulation, linear/nonlinear PnP and bundle adjustment.

Below sections show some main problems in feature realm such as detection, matching and tracking issue, and optimization issues when implementing nonlinear triangulation, PnP and bundle adjustment. Finally, we raise corresponding technical approaches to improve the performance.

II. FEATURE DETECTION, MATCHING AND TRACKING

A. PROBLEM FORMULATION

1) Definition of Good Feature: Feature detection is the very first step of the visual SLAM pipeline. There are multiple types of feature that can be used for detection, such as SIFT, SURF, ORB, FAST, and so on. Each of them has its own properties so that performs different in various scenarios, therefore, its necessary and important to choose proper feature type given by the dataset.

2) Feature Distribution Issue in Feature Detection: The distribution of feature is the main issue in feature detection. Features with uniform distributions as well as various depth information truly do favor to the following update step where multiple 3D vision algorithms will be implemented.

However, different image frame has its own scenario, its common that most feature will appear in corner and bright region, while some flat or dark place might contain little detected result which highly possible makes feature distribution become into several clusters instead of the uniform one.

3) Outliers Problem: Feature matching plays an essential and important role in the following feature tracking. After feature detection, it's necessary to build label for each feature in order to distinguish them, and we call this label as the feature descriptor. Suitable descriptor is crucial.

What's more, outliers which bring mismatch problem will make bad performance of the whole system, one way to solve that is using RANSAC (Random Sample Consensus), however, parameters tuning in RANSAC is not easy but truly determines the performance of feature matching a lot.

4) Feature Tracking Difficulties: It's not difficult to track same physical point in two or three images. However, with the number of frames increase, due to the random property of RANSAC, tracking same feature along the trajectory seems quite difficult. With no enough tracked feature, we cannot expect good result from triangulation, PnP and bundle adjustment.

One possible solution is increasing the threshold of RANSAC which brings more detected feature in each frame and compare error of position in neighbor frames to find correspondences, but RANSAC is a trade off issue, that is, more features including means higher probability of having outliers. Therefore, it's necessary to figure out some advanced approaches to handle this problem.

B. TECHNICAL APPROACH

This section shows some several technical approaches to deal with mentioned problems above.

1) Feature Selection based on Scenario: As we know, there are multiple features can be used for detection, matching and tracking. However, each of them has its own properties and some of them may perform bad in certain scenario. Therefore, in order to select suitable feature detector for our project, we compare attributes of three main and common used features, SIFT, SURF and ORB, and analysis with the practical scenario.

According to the table I, SIFT is a good feature type but with heavy computational complexity, while ORB has reasonable speed as well as performance. Then we test the performance of those features in different scenarios.

According to the results in table II, in our case, there is no much scaling, fish eye and shearing problems between multiple images, while the image noise and computational complexity matter a lot, so finally we choose ORB feature.

TABLE I: Properties of Features

	SIFT	SURF	ORB
Properties	Efficient in object recognition but requires a large computational complexity	Similar as SIFT but much faster	Faster than SIFT with similar performance, but feature always locate at the center of images.

TABLE II: Feature Matching Performance in Different Scenarios

	SIFT	SURF	ORB
Intensity	Best match and longest time	Good match and short time	Bad match and shortest time
Rotation	Best match and longest time	Average match and short time	Bad match and shortest time
Scaling	Bad match and longest time	Bad match and short time	Best match and shortest time
Shearing	Best match and longest time	Good match and short time	Bad match and shortest time
Fish Eye	Best match and longest time	Bad match and short time	Bad match and shortest time
Noise	Good match and longest time	Bad match and short time	Best match and shortest time

However, not all ORB features are good features, in order to make KLT perform better, we define a good feature whose both eigenvalues are larger than certain threshold, that is, $\min(\lambda_1, \lambda_2) \geq \lambda_{threshold}$.

2) Adaptive NMS and Image Color Contrast: In order to make features distribute uniformly, the most common way is called the Adaptive NMS(Non-Maximal Suppression). Practically, We always use corner as good feature since it's eigenvalues are both large. The basic idea of Adaptive NMS is, if a pixel can be regarded as a good feature, it should be the largest one within some certain region. Essentially, for each image frame, we need to figure out its max radius requirement which a corner feature should satisfy in order to become a good one. Then based on our desired output feature points number, we can choose the max radius for each image, eventually we can achieve features to distribute uniformly.

However, the adaptive NMS approach cannot perform well if there exist no detected features in some region, such as dark region in image. In order to deal with that problem, we choose to reduce the image color contrast. The idea of that approach is pretty straightforward, that is, increase the brightness of dark region while decrease the intensity of some light places in image. After image contrast reduction, we can get more feature points in dark region. Figure 1 shows the effect of color contrast reduction.

3) RANSAC to filter outliers: In order to match features correctly in multiple frames, we need to deal with two problems, first, create a good descriptors for each feature; second, construct RANSAC algorithm with suitable parameters in order to filter out outliers as much as possible.

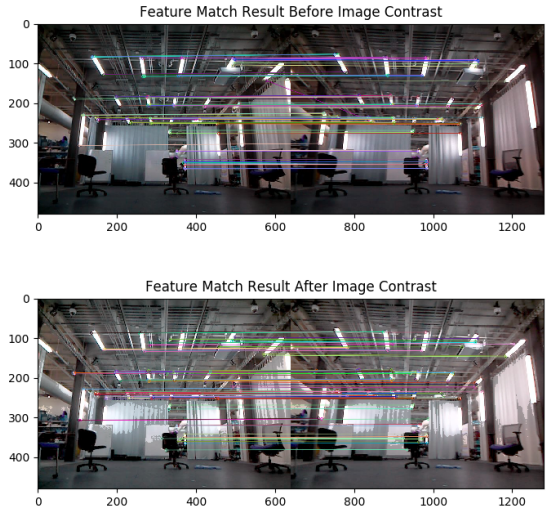


Fig. 1: The effect of image color contrast reduction

For descriptor construction, the intuition is simple, we create a window around each good feature with certain size such as 40 x 40; then down sample for the window to implement blurring which will improve the feature property of being invariant to scaling and other transformation; after that, normalize downsampled window with zero mean value and standard deviation of one; finally, unroll window into a vector as the descriptor of feature point.

RANSAC is the most common approach to deal with outliers. Since the transformation between two frames is not pure translation or rotation, so we cannot use homography to estimate projection error. Instead, we use fundamental matrix to compute reprojection error and set certain threshold to filter outliers. The pipeline shows below:

1) Randomly select 8 correspondences in two images.
2) Use those 8 points to estimate fundamental matrix.
3) Compute inliers where

$$X_2^T * F * X_1 < threshold$$

.

4) Count the vote number for current fundamental matrix as well as indices for all inliers, update max vote number if necessary.
5) Keep the largest inliers and iterate above procedures for certain times.
6) Repeat steps from 1 to 5, and if current count vote is larger than 95 percent of total correspondences, stop the iteration.
7) Use all inliers to compute fundamental matrix again and filter outliers.

According to the figure 2 and 3, RANSAC can truly improve the performance of feature matching.

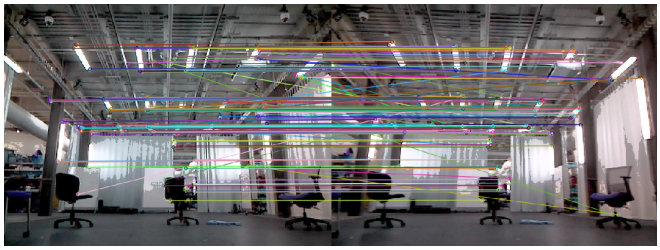


Fig. 2: Feature Match Result before RANSAC

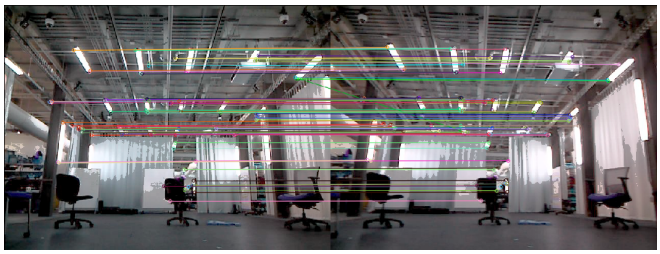


Fig. 3: Feature Match Result after RANSAC

However, for some extreme case, RANSAC still cannot filter outliers. Figure 4 is one of my failures, mismatch happens at bright lamp part, the possible problem might be the high intensity in those regions make these descriptors almost same. One possible solution is using Deep Neural Network to extract more information for each good feature..

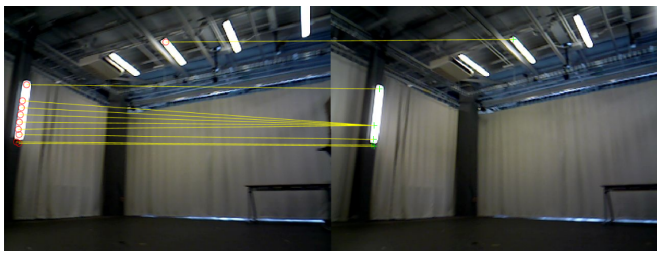


Fig. 4: Failures after RANSAC

4) Kanade-Lucas-Tomasi (KLT) Feature Tracker: In computer vision, the Kanade-Lucas-Tomasi (KLT) feature tracker is one of the common used approaches for feature extraction and tracking. As we know, the traditional image registration techniques are generally costly, while KLT makes use of spatial intensity information to direct the search for the position that yields the best match. This is faster than traditional ones for examining far fewer potential matches between the images.

Theoretically, KLT is based on the Lucas-Kanade Algorithm. The Lucas-Kanade Algorithm is essentially an optimization question and uses intensity error between origin and warped images to compute gradient so that converges warping parameters. Below shows the pipeline of the Lucas-Kanade Algorithm.

1) Warp image I(x) with warping function W(x;p) to compute warped version I(W(x;p)).
2) Compute image intensity error T(x)-I(W(x;p)) as well as the warp gradient ∇I with W(x;p).
3) Evaluate the Jacobian at (x;p) and its steepest descent images.
4) Compute the action parameter Δp to update original one until its steps smaller than certain threshold.

After implementation of above Lucas-Kanade Algorithm, we are supposed to obtain an optimal warp function between two images, therefore, continue the below pipeline and complete KLT algorithm.

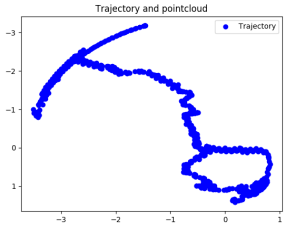
1) For two neighbor images, apply Lucas-Kanade Algorithm for each good feature position so that get corresponding optimal warp function, use that to compute the displacement from current position to next frame.
2) Store displacement of each feature and update its position.
3) (Optional) In order to avoid the reduction of feature numbers with the increase of camera frames, one alternative approach is adding more feature positions per M frames.
4) Iterate step 1 to 3 and finally return and plot the trajectory for each feature as tracking line.

Since in this case, we assume that a good feature has both larger eigenvalues, therefore, the performance of KLT in feature tracking is stable and pretty fine(Please follow instructions in 'Readme.txt' and check the result by running the demo code).

Finally, we use those tracking result as well as camera frame information to initialize the designed point cloud structure.

III. Visual State Correction and Sparse Pointcloud Building

The robot state prediction using robot's odometry has limitation on the accuracy, for example, in figure 5 the predicted trajectory is by far off from the ground truth trajectory. Thus, there is a need to correct this prediction. In this study, we



(a) Predicted trajectory

(b) Ground truth (red line)

Fig. 5: Prediction vs. ground truth

exam visual measurement using a monocular camera to make correction for the prediction step. We adopt a pinhole camera model and propose a pipeline as follows. In the diagram in figure 6, the estimation of the robot states (with respect to the world), after being transforming into the world-image frame transformation using intrinsic and extrinsic parameters that we omitted here, is used as inputs for the triangulation step. Note that time synchronization between camera images and the robot states is also performed. Therefore, from now on,

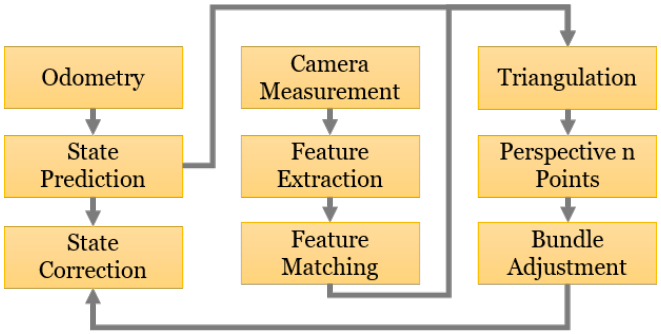


Fig. 6: Diagram of the whole process

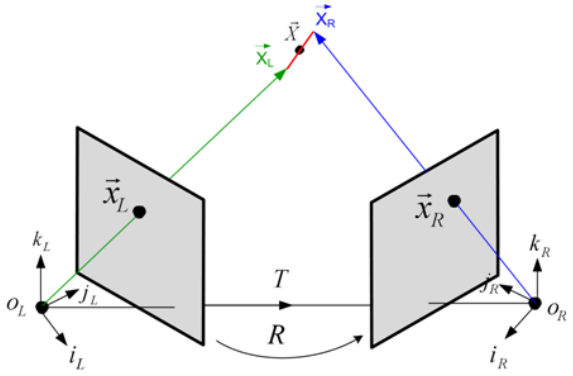


Fig. 7: Triangulation

we assert that each camera frame is one-to-one mapped with a robot state and when mentioning a camera pose, we refer to an image frame associated with a robot state that we need to correct. Moreover, our target is to refine the trajectory of camera poses or frames to correct our target, the robot trajectory in a sense that given the camera pose trajectory, we can use homogeneous transformation to compute the robot's trajectory in the world frame.

A. PROBLEM FORMULATION

After performing feature matching as described in the previous section, we can obtain information about 2D position of matching features in pairs of images. We are also given an estimation of corresponding camera poses with respect to the world after the state prediction step. Our first task is to refine these camera pose estimations to correct the robot state estimations. The second task is to generate the 3D point cloud of feature points as to generate a sparse 3D map.
Our problem of camera pose estimation and point cloud building can be formulated as an optimization problem in which we try to minimize the reprojection error. The reprojection error is expressed as the sum of square error of every single 3D-2D correspondence. Assume that there are m 3D points which are seen in n camera views and let v_{ij} denote a point j viewed by camera i and x_{ij} is the corresponding position in the image. Therefore, we can set $V_{ij} = 1$ if point j is visible in view i and 0 otherwise. In addition, assume that camera pose i is parametrized by a 6-element vector P_i $(r_{i1}, r_{i2}, r_{i3}, c_{ix}, c_{iy}, c_{iz})$ that encode rotation vector $r_i = (r_{i1}, r_{i2}, r_{i3})$ and translation $C_i = (c_{ix}, c_{iy}, c_{iz})$. 3D world point j is represented as a 3-element vector x_j, y_j, z_j. The reprojection error with respect to all 3D points and camera poses can be formulated as

$$\min_{\{C_i,r_i\}_{i=0}^{m},\{X\}_{j=0}^{n}} \sum_{i=0}^{I}\sum_{j=0}^{J} V_{ij}\left(\left(u_{ij} - \frac{P_{i1}^T\tilde{X}_j}{P_{i3}^T\tilde{X}_j}\right)^2 + \left(v_{ij} - \frac{P_{i2}^T\tilde{X}_j}{P_{i3}^T\tilde{X}_j}\right)^2\right) \tag{1}$$

where $(u_{ij}, v_{ij}$ is the 2D position of feature j^{th} on camera pose i^{th}. P_{ik}^T is each row of camera i^{th}'s projection matrix, P_i, which is computed by $KR_i[I_{3x3} \ -C_i]$, where K is the intrinsic camera parameter and R_i, t_i are rotation and translation from the world to camera, respectively and $C_i = R_i^T t$. Since this optimization problem includes a large number of variable, we propose to tackle it by a procedure as introduced in the figure 6. At first, we will use the camera poses obtained from the prediction step as an initialization for constructing a set of 3D point cloud, using triangulation (including linear and nonlinear triangulation). Then, linear and nonlinear PnP are applied to refine the estimation of camera poses. The results of these two steps are eventually used as initializations of bundle adjustment to refine the camera poses and 3D feature points.

B. TECHNICAL APPROACH

1) Triangulation: The purpose of this step is to estimate a 3D position of a feature point, given its 3D correspondences in images and the relative location and orientation of camera poses. Figure 7 gives an illustration on this.

2) Linear Triangulation : Suppose there is a relationship between a 3D point X and a 2D point x in an image: $x = PX$, where $P = KR[I \ -C]$, K is the intrinsic parameter matrix, R is rotation matrix transformation from the camera to the world in the camera frame, C is the translation vector from the world to the camera in the world frame. We can obtain $x(PX) = 0$, which gives us

$$u(p^{3T}X) - (p^{1T}X) = 0 \tag{2}$$
$$v(p^{3T}X) - (p^{2T}X) = 0 \tag{3}$$

where p^{iT} is row i of P and $x = (u, v)^T$. Given n point $x's$, we can stack these equations into the form $AX = 0$ and solve it using SVD method, with

$$A = \begin{bmatrix} u_1 p^{3T} - p^{1T} \\ v_1 p^{3T} - p^{2T} \\ \dots \\ u_n p^{3T} - p^{1T} \\ v_n p^{3T} - p^{2T} \end{bmatrix} \tag{4}$$

We did experiments on the data provided in project 3-SLAM. After obtaining 3D point cloud, we projected back to 2D to obtain the reprojection error of all feature points and showed in figure 8.

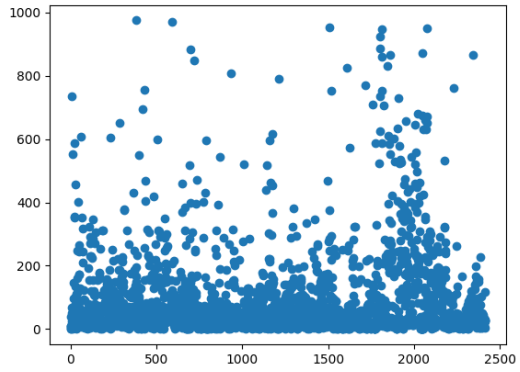


Fig. 8: Reprojection error after linear triangulation

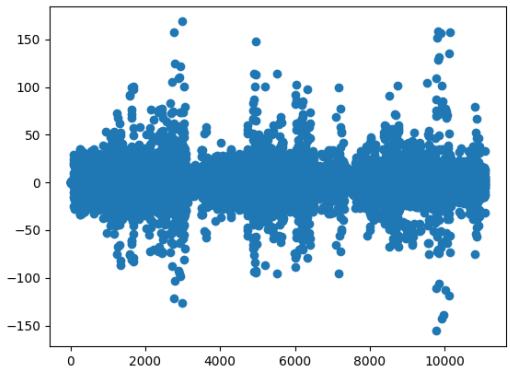


Fig. 9: Reprojection error after nonlinear triangulation

3) Nonlinear Triangulation: As can be seen in figure 8, the linear triangulation method results in a huge error. Besides the fact that the camera pose estimation is inaccurate due to the noise in odometry, the triangulation problem is a nonlinear problem so solving it using a linear least square method will probably suffer from inaccuracy. However, the linear triangulation step is important because it can provide an initialization for the nonlinear triangulation algorithm which try to optimize the following cost function

$$\min_X \sum_{j=0}^{n-1} \left(u_j - \frac{P_{j1}^T \tilde{X}}{P_{j3}^T \tilde{X}} \right)^2 + \left(v_j - \frac{P_{j2}^T \tilde{X}}{P_{j3}^T \tilde{X}} \right)^2 \tag{5}$$

Using result from the linear triangulation step, we executed the nonlinear triangulation step to refine the 3D position of feature points then project back 2D to obtain the reprojection error of all feature points and showed in figure 9. As the figure shows, the projection error is much less than that of the linear triangulation step. However, it is still high so we need to move on on improvement.

4) Perspective-n Points: In the triangulation step, we do nothing with the camera poses. We can actually use them as initialization for bundle adjustment but it will result in a large inaccuracy since the provided camera poses are too off from the ground truth. Thus, in here, we propose to apply PnP method to increase the accuracy of camera poses after using bundle adjustment. In this step, we are given 3D position of feature points from triangulation step and our task is to estimate the 3D position and orientation of camera poses.

5) Linear PnP : The general formulation is to find the transformation of the camera pose with respect to the world that minimizes the reprojection error

$$\arg\min_{R,C} \sum_j ||p^j - \hat{p}^j|| \tag{6}$$

where $\hat{p}^j$ is the reporjection of 3D point X^j into the image I with the transformation R,C. Let $X = (x,y,z)^T$ be the 3D point and $x = (u,v)^T$ be its reprojection on the image, we have the following relation

$$\begin{aligned} u &= \frac{P_{11}x + P_{12}y + P_{13}z + P_{14}}{P_{31}x + P_{32}y + P_{33}z + P_{34}} \\ v &= \frac{P_{21}x + P_{22}y + P_{23}z + P_{24}}{P_{31}x + P_{32}y + P_{33}z + P_{34}} \end{aligned} \tag{7}$$

Thus, each 3D-2D correspondence can provide two equations while in the transformation matrix, there are 6 degrees of freedom (DOFs). Thus, wee need at least three 3D-2D correspondences. In general, if we have $m \geq 3$ 3D-2D correspondences, we stack the corresponding constraints, form $AX = 0$, where A is

$$\begin{bmatrix} 0_{1x4} & -x_1 & -y_1 & -z_1 & -1 & x_1v_1 & y_1v_1 & z_1v_1 & v_1 \\ x_1 & y_1 & z_1 & 1 & 0_{1x4} & -x_1u_1 & -y_1u_1 & -z_1u_1 & -y_1 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ 0_{1x4} & -x_m & -y_m & -z_m & -1 & x_mv_m & y_mv_m & z_mv_m & v_m \\ x_m & y_m & z_m & 1 & 0_{1x4} & -x_mu_m & -y_mu_m & -z_mu_m & -y_m \end{bmatrix} \tag{8}$$

and the variable X is an $1x12$ column vector constructed by stacking rows of P. We then use SVD to solve X. Eventually, the rotation and translation part can be easily extracted from $P = KR[I \ -C]$.

6) PnP RANSAC: The least square method is highly sensitive to outliers so we apply RANSAC algorithm to get rid of outliers to perform linear PnP better. The algorithm is shown in figure 10.

7) Nonlinear PnP: Using results obtained from the linear PnP step as an initialization, we solve an optimization problem over R_i, C_i to refine the estimation of camera poses using 3D feature points.

$$\min_{R,C} \sum_{j=0}^{m-1} \left(u_j - \frac{P_{j1}^T \tilde{X}}{P_{j3}^T \tilde{X}} \right)^2 + \left(v_j - \frac{P_{j2}^T \tilde{X}}{P_{j3}^T \tilde{X}} \right)^2 \tag{9}$$

8) Bundle Adjustment: After the above steps, we can obtain a set of 3D points and camera poses. The bundle adjustment, then, is used as a means to simultaneously refine the 3D point cloud and the parameters of the camera poses, by minimizing the reprojection error between the observed and reprojected point positions. The cost function is formulated exactly as in equation 1.

Algorithm 2 PnPRANSAC

$n \leftarrow 0$
for $i = 1 : M$ **do**
  Choose 6 correspondences, $\hat{\mathbf{X}}$ and $\hat{\mathbf{x}}$, randomly
  `[C R] = LinearPnP(`$\hat{\mathbf{X}}$`, `$\hat{\mathbf{x}}$`, K)`
  $\mathcal{S} \leftarrow \emptyset$
  for $j = 1 : N$ **do**
    $e = \left(u - \frac{\mathbf{P}_1^\intercal \tilde{\mathbf{X}}}{\mathbf{P}_3^\intercal \tilde{\mathbf{X}}}\right)^2 + \left(v - \frac{\mathbf{P}_2^\intercal \tilde{\mathbf{X}}}{\mathbf{P}_3^\intercal \tilde{\mathbf{X}}}\right)^2$
    if $e < \epsilon_r$ **then**
      $\mathcal{S} \leftarrow \mathcal{S} \cup \{j\}$
    end if
  end for
  if $n < |\mathcal{S}|$ **then**
    $n \leftarrow |\mathcal{S}|$
    $\mathcal{S}_{in} \leftarrow \mathcal{S}$
  end if
end for

Fig. 10: PnP RANSAC algorithm

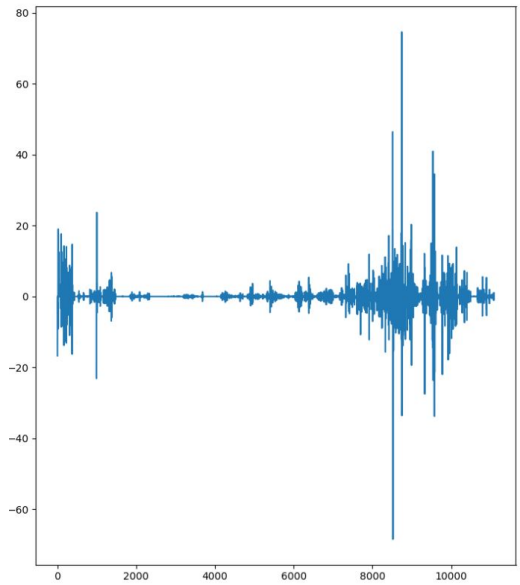


Fig. 11: Reprojection error after bundle adjustment

Using result from the PnP step, we executed the bundle adjustment step and obtain the reprojection error of all feature points and showed in figure 11. As can be seen, the PnP and bundle adjustment steps have helped reduce the reprojection error significantly.

IV. Experiment: Back End formulation for the Toy problem

Before implementing the bundle adjustment on real data, implementation was done on synthetic data for proof of concept. The first task was to observe the performance of a non linear least squares solver on 2D pose-landmark data to get an idea of how the problem is solved. The problem is formulated as a factor graph where the nodes or vertexes represent the pose of the key frames (pose of the robot) or the positions of the landmarks. The edges represent the constraints between 2 poses or between a pose and a landmark. The following data was provided to the linear solver g2o.

- The initial estimate of the 2D positions of the landmarks.
- The initial estimate of the 2D pose (position and orientation) of the camera / robot.
- Constraint between pairs of key frames. (Represented by the blue edges in the pose graph in Fig). This represented the pose of one of the robots as viewed in the local frame of the other.
- Constraint between a key frame and a landmark. It provides the estimation of the 2D position of the robot as viewed in the local frame of the robot.

Thus it possible to attain two estimates of the position of the robot and the landmarks, one from the initial estimates and one from the constraints. Thus there is a residual that can be constructed from each of these errors weighted by the information matrix (inverse of the covariance) for each measurement. The problem is formulated as,

$$x^* = argmin_x \sum e_i^T \Omega e_i \tag{10}$$

Here x, represents the entire set of poses of the robot and the locations of all the landmarks in the 2D space. e_i represents the instance of the the error between the two estimates of the pose or position. Ω represents the information matrix which is the inverse of the covariance matrix. The minimization can be thought of as adjusting the pose and positions of the robot and the landmark so that the constraints represented by the edges are best satisfied. Hence the goal is to estimate the pose / positions that minimize the cumulative sum of squares of residuals. The g2o solver is an excellent solver for this problem. For the Victoria Park Dataset shown in Fig. 12 containing 6969 poses, 151 landmarks and 10608 constraints, the g2o solver reduces the sum of squared residual from 5e+7 to 3e+4 as shown in Fig. 13. The result at various iterations of optimization can be seen in Fig. 12.

Following this, a toy problem was setup to implement bundle adjustment to find the optimal 3D positions of points in the world and 3D poses of the cameras. The initialization of the point cloud is shown in Fig. 8. The point cloud initialization was constructed by adding random noise to the original point cloud. A camera starting from origin moves towards the point cloud and clicks pictures of the point cloud. The pictures were attained by projecting the original point cloud from the original pose. An initialization of the pose was also attained by adding noise to the pose estimates. Given the intrinsic parameters of the robot K, the pose of the robot $[R|t]$ the projection of the 3D point $P = [X,Y,Z]$ on the image plane can be found out by the function f:

$$f(K,R,t,P) = \hat{p} = [u,v,1]^T = K * [R^T | -R^T t] * [X,Y,Z,1]^T \tag{11}$$

where [X,Y,Z,1] and [u,v,1] are the homogeneous coordinates of the 3D point and its 2D projection on the image plane respectively. $\hat{p}$ is hence the pixel coordinates of the 3D point as estimated from the corresponding 3D coordinates. Given the actual pixel coordinates in the image plane p, the squared

error can be estimated as

$$||p - \hat{p}||_2^2 \tag{12}$$

This error is known as the re-projection error. Thus, given the initialization of the point cloud, the camera poses, the intrinsic parameters of the camera and the 3D-2D correspondences for each frame, the set of optimal robot poses and point cloud locations $\chi^* = ([R_1,t_1],[R_2,t_2],.....,P_1,P_2,.....)$ can be found by

$$\chi^* = argmin_\chi \sum ||p_k - f(K,R_i,t_i,P_j)||_2^2 \tag{13}$$

This can done by first order linearization of the residuals followed by collecting of the Jacobians into a large sparse matrix. This reduces the problem a linear least square estimation with the change in parameters as the variables which can easily be solved by sparse matrix factorization techniques like QR factorization or Cholesky Factorization. Alternatively ,it can also be solved by setting the derivative of the first order linearization with respect to the parameters as 0 to get a sub optmial solution of the parameters. The residuals and hence the cost are recalculated using the updated parameters and the process is repeated. This process is done iteratively till the value of the cost function converges.

The solver used for this process was the Google Ceres Solver. The results of the point clouds attained after implementing the Bundle Adjustment are shown in Figs. 14 and 15 for the cow and lamp dataset respectively. As can be seen, the reprojection error reduces drastically after convergence and the point cloud looks visually appealing after the adjustment indicating that the algorithm worked correctly.

Even though both the toy problems gave very positive results, these results cannot be used as a benchmark for performance on the real dataset. This is because in the case of the pose-landmark graph optimization scenario the measurements i.e. the estimates of the landmark positions in the local frame of the robot were pretty accurate. Similarly in the 3D bundle adjustment toy problem, the initialization of the pose was quite close to the actual optimal value. Hence, in both optimization, the local minima turned out to give extremely good results. But in the real world, getting a good initialization on the trajectory of the robot and the point cloud is extremely difficult. Measurements in the real world are often very noisy and the optimization done using this data often results in convergence to a high cost value at an incorrect local minima.

V. Results

After doing experiments with the synthetic dataset in section IV, we performed experiments with a real dataset adopted from the SLAM project as seen in figure 16. There are indeed various challenges when dealing with this data. The first challenge is the inaccuracy of the odometry information which results in a predicted trajectory which is by far off from the ground truth as can bee seen in figure 16. The second challenge is that there are numerous transformation that we need to perform from the world to the robot body, from the robot body to the head, then from the head to the camera frame before ending up from the camera frame to the image frame. These transformation make the debugging step become really challenging.

Due to these hardships and a limitation in time, our results in the SLAM dataset is less appealing. As can be seen in figure 17, our obtained trajectory has been improved after performing nonlinear PnP (figure 18a) and bundle adjustment (figure 18b) as compared with that obtained from the robot state prediction. However, there is still a need to improve it. Figure 18 shows the point cloud.

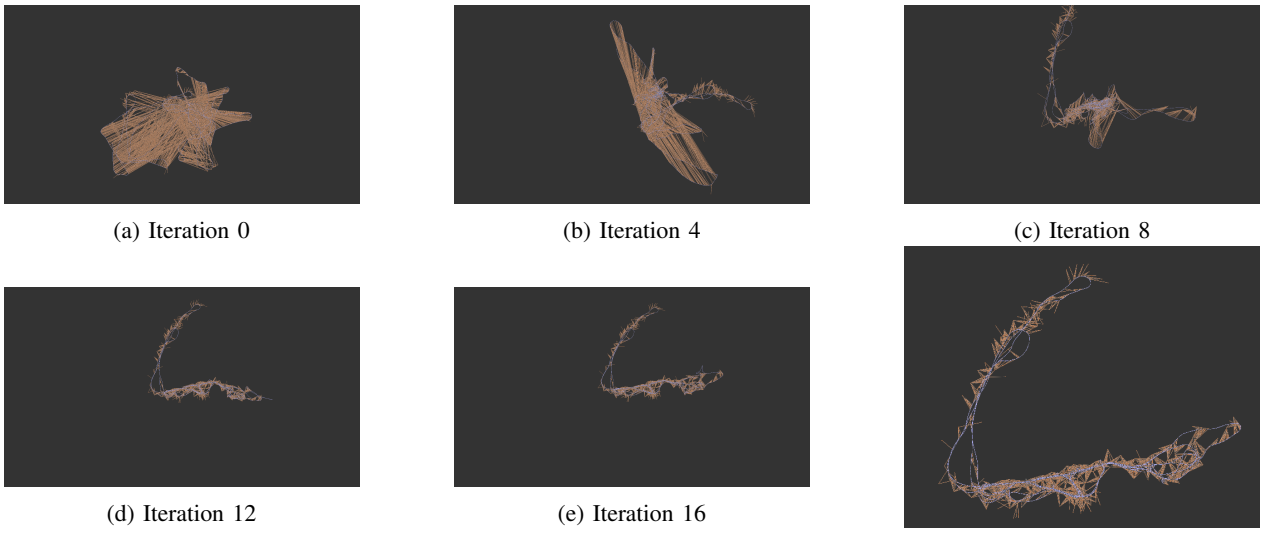
VI. CONCLUSION

In the modern SLAM problem, there are multiple approaches to give good performance, some of them use lidar sensor and scan information to achieve localization and mapping, however, due to the expensive cost of those lidar sensor equipments, visual SLAM which only uses cameras becomes more and more popular. Our final project is based on the visual SLAM base to achieve robot localization and 3D mapping.

Theoretically, given multiple image frames, we can detect features as well as matching and tracking them to build the connections between image frame world with 3D physical world. The most common used approaches including triangulation, PnP and bundle adjustment.

However, there truly exists some constraints to get good result using above methods. The quality of features matters a lot, for example, in order to get reasonable estimation in bundle adjustment, it's better to make tracking features distribute uniformly in image frame so that they can bring multiple dense information.

To be honest, our estimated results are not good since we may have bad prediction of camera pose and position, linear / nonlinear triangulation may also have some problems. But we have implemented extra experiments using g2o and other data sources to verify that the above technical approaches are pretty reasonable. And this topic is worth further researching.



(a) Iteration 0 (b) Iteration 4 (c) Iteration 8

(d) Iteration 12 (e) Iteration 16 (f) Final Iteration

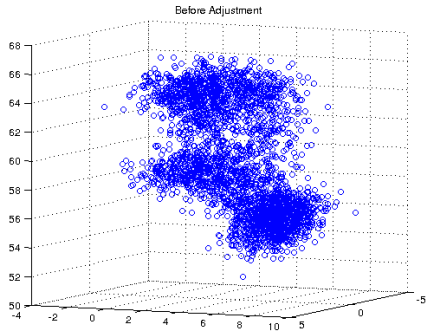
Fig. 12: Optimization of pose-landmark graph using g2o for 2D SLAM. The dataset used is called Victoria Park. The subfigures represent the graph through various stages of optimization.

```
loaded g2o_landmark/victoriaPark-full.g2o with 7120 vertices and 10608 measurements
graph is fixed by node 0
# Using CHOLMOD poseDim -1 landMarkDim -1 blockordering 0
Preparing (no marginalization of Landmarks)
iteration= 0 chi2= 50827603.062346 time= 0.0471231 cumTime= 0.0471231 edges= 10608 schur= 0
iteration= 1 chi2= 123473350.178086 time= 0.0259882 cumTime= 0.0731113 edges= 10608 schur= 0
iteration= 2 chi2= 46291063.116528 time= 0.0183713 cumTime= 0.0914826 edges= 10608 schur= 0
iteration= 3 chi2= 26184948.380259 time= 0.0186635 cumTime= 0.110146 edges= 10608 schur= 0
iteration= 4 chi2= 30557528.202998 time= 0.0183103 cumTime= 0.128456 edges= 10608 schur= 0
iteration= 5 chi2= 11108363.578965 time= 0.0185402 cumTime= 0.146997 edges= 10608 schur= 0
iteration= 6 chi2= 3425378.336913 time= 0.0174533 cumTime= 0.16445 edges= 10608 schur= 0
iteration= 7 chi2= 2065088.548901 time= 0.0176221 cumTime= 0.182072 edges= 10608 schur= 0
iteration= 8 chi2= 509607.053733 time= 0.0173261 cumTime= 0.199398 edges= 10608 schur= 0
iteration= 9 chi2= 468715.958786 time= 0.0176801 cumTime= 0.217078 edges= 10608 schur= 0
iteration= 10 chi2= 432750.586014 time= 0.0174217 cumTime= 0.2345 edges= 10608 schur= 0
iteration= 11 chi2= 192310.084152 time= 0.0176632 cumTime= 0.252163 edges= 10608 schur= 0
iteration= 12 chi2= 46209.356581 time= 0.0172754 cumTime= 0.269439 edges= 10608 schur= 0
iteration= 13 chi2= 34841.077600 time= 0.0175721 cumTime= 0.287011 edges= 10608 schur= 0
iteration= 14 chi2= 33640.002614 time= 0.0182995 cumTime= 0.30531 edges= 10608 schur= 0
iteration= 15 chi2= 31474.480251 time= 0.0185763 cumTime= 0.323886 edges= 10608 schur= 0
iteration= 16 chi2= 31202.165520 time= 0.0182249 cumTime= 0.342111 edges= 10608 schur= 0
iteration= 17 chi2= 31126.285761 time= 0.0186085 cumTime= 0.36072 edges= 10608 schur= 0
iteration= 18 chi2= 31072.770063 time= 0.0182144 cumTime= 0.378934 edges= 10608 schur= 0
iteration= 19 chi2= 31015.404864 time= 0.017936 cumTime= 0.39687 edges= 10608 schur= 0
```

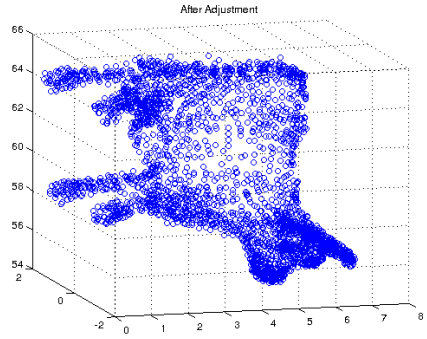
Fig. 13: Convergence of chi-squared error using g2o for the Victoria Park dataset

```
Parameters: mode = 4 w3Dv2D = 0.1       Meta Info: nCam = 20 nPts = 2903 n0bs = 58060 n_objects = 0     Camera Intrinsic: fx = 718.856 fy = 718.856 px
<------------------------------- In Mode 4 ---------------------------------->
iter      cost      cost_change  |gradient|   |step|    tr_ratio  tr_radius  ls_iter  iter_time  total_time
   0  2.865088e+05    0.00e+00    2.29e+05   0.00e+00   0.00e+00  1.00e+04        0    5.01e+00    5.13e+00
   1  1.183561e+03    2.85e+05    2.12e+05   6.30e+01   4.51e+00  3.00e+04        1    5.33e+00    1.05e+01
   2  4.794817e+02    7.04e+02    5.92e+03   2.36e+01   1.02e+00  9.00e+04        1    5.27e+00    1.57e+01
   3  4.783539e+02    1.13e+00    1.53e+01   2.25e+00   1.00e+00  2.70e+05        1    5.25e+00    2.10e+01
Ceres Solver Report: Iterations: 3, Initial cost: 2.865088e+05, Final cost: 4.783539e+02, Termination: CONVERGENCE
```

(a) Convergence of cumulative squared reprojection error



(b) Point cloud before Bundle Adjustment

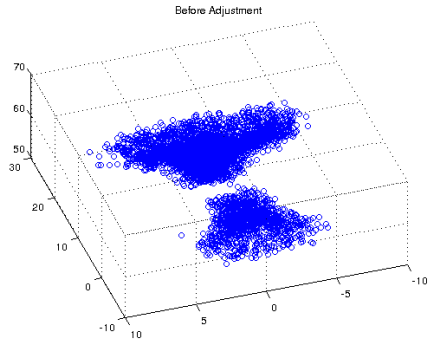


(c) Point Cloud after Bundle Adjustment

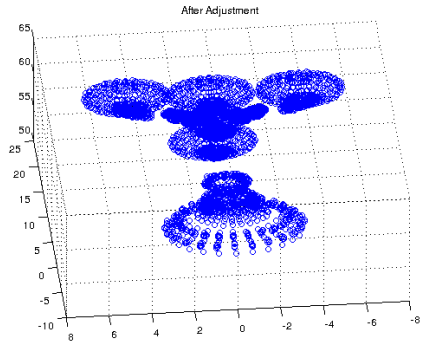
Fig. 14: Bundle adjustment for 20 camera poses on the cow dataset

```
Parameters: mode = 4 w3Dv2D = 0.1       Meta Info: nCam = 20 nPts = 4440 n0bs = 88800 n_objects = 0     Camera Intrinsic: fx = 718.856 fy = 718.856 px
<------------------------------- In Mode 4 ---------------------------------->
iter      cost      cost_change  |gradient|   |step|    tr_ratio  tr_radius  ls_iter  iter_time  total_time
   0  4.279782e+05    0.00e+00    3.18e+05   0.00e+00   0.00e+00  1.00e+04        0    8.05e+00    8.21e+00
   1  2.041342e+03    4.26e+05    7.20e+05   8.05e+01   4.42e+00  3.00e+04        1    8.45e+00    1.67e+01
   2  7.369503e+02    1.30e+03    5.83e+03   1.91e+01   1.04e+00  9.00e+04        1    8.31e+00    2.50e+01
   3  7.365077e+02    4.43e-01    1.25e+01   1.05e+00   1.00e+00  2.70e+05        1    8.50e+00    3.35e+01
Ceres Solver Report: Iterations: 3, Initial cost: 4.279782e+05, Final cost: 7.365077e+02, Termination: CONVERGENCE
```

(a) Convergence of cumulative squared reprojection error



(b) Point cloud before Bundle Adjustment



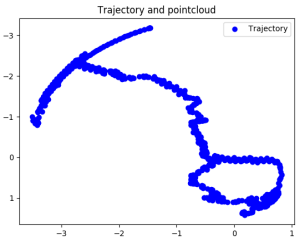
(c) Point Cloud after Bundle Adjustment

Fig. 15: Bundle adjustment for 20 camera poses on the lamp dataset

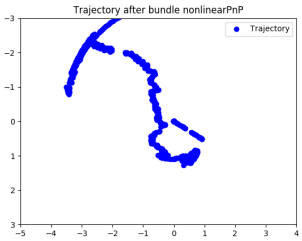
(a) A screenshot of the scene in dataset used in the experiment

(b) Ground truth trajectory (red line)

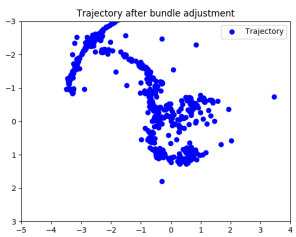
Fig. 16: Dataset



(a) Trajectory after prediction

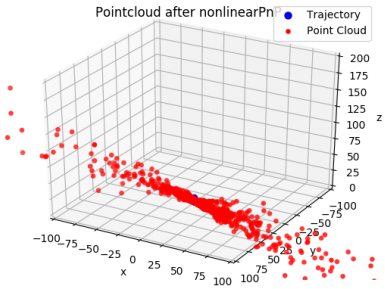


(b) Trajectory after nonlinear PnP

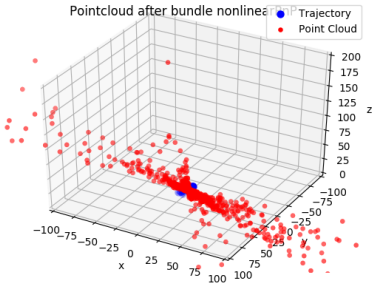


(c) Final trajectory

Fig. 17: Trajectory improvement



(a) Point cloud after nonlinear PnP



(b) Final point cloud

Fig. 18: Point cloud