RGB-D Visual Odometry on TUM Dataset

Omar Elmofty
Institute for Aerospace Studies
University of Toronto, Toronto, ON

***Abstract* –A robot's ability to localize itself in an unknown environment is a very challenging problem. A lot of techniques and sensors have been developed over the years to solve this problem. One of those techniques is Visual Odometry, which is the process by which a robot localizes itself over time relative to its initial pose using only visual sensors. Visual odometry has a wide variety of applications, such as autonomous driving, mining, space exploration … etc. This paper presents a pipeline for performing visual odometry using RGB-D images. Specifically, the image sequences used in this work were sourced from the Technical university of Munich's RGB-D SLAM dataset. This paper presents the results of the proposed pipeline applied to the TUM dataset, as well as a method for estimating the uncertainty in the predictions using a Monte Carlo approach. Attached to this submission is the code of the pipeline as well as a demonstration video.**

I. Introduction

There are many sensors and techniques that can be used to retrieve a robot's location in an environment, examples are wheel odometry, global positioning system (GPS), global navigation system (GNSS), inertial navigation system (INS) and Visual odometry (VO) [2]. Most of the techniques mentioned have their own advantages and disadvantages, and although GPS tends to be the default go to for localization, it requires clear access to the sky which sometimes is not possible [2]. Hence, visual odometry provides an alternative, inexpensive and reliable solution to the localization problem with the absence of GPS.

In this paper, a pipeline for performing visual odometry using RGB-D images is presented. RGB-D is an acronym for a pair of images, with the first being a regular RGB image, and second being a depth map of the scene. Such images can be captured using inexpensive sensors such as Microsoft Kinect or ASUS Xtion [3]. The pipeline presented works by extracting features from the RGB-D images and then by performing scalar weighted point cloud alignment [10] between features from consecutive camera frames. The pipeline was tested on the technical university of Munich's (TUM) dataset [1].

II. Related Work

There are two primary approaches to performing visual odometry, feature-based or appearance based [2]. The feature-based approach works by extracting features from the scene (could be SURF, SIFT or ORB features [8]), then transforming the extracted features to point clouds in Euclidean space. This is performed for every frame of the image sequence, and then transformation from frame to frame is computed by minimizing the squared distance between point clouds of consecutive frames. Appearance based approaches work by monitoring the intensity changes of pixels between camera frames and using this information to estimate the motion of the vehicle, such techniques are also referred to as optical flow [2].

In this paper only feature based approaches were considered, for such approaches several popular algorithms exist for performing the scalar weighted point cloud alignment, such as Nister's algorithm [4] and others. The method selected for use in this pipeline has been referenced from T. D. Barfoot's State Space Estimation Book [10]. Although, the point cloud alignment method is a reliable and efficient solution for computing the transformation between camera frames, it doesn't account for the sensor error. Another more involved technique used for aligning point clouds is bundle adjustment [11], which considers the vehicles poses, and measured features as both uncertain, and optimizes for both poses and feature locations in space. Although, this technique is more accurate as it considers the sensor errors, it was not implemented due to the limitations in time and resources.

III. Experimental Setup

The setup consists of a Microsoft Kinect camera which has visual markers attached to it, the Kinect was attached to a pioneer robot (as shown in figure 1) and the robot was moved around in an enclosed environment. The ground truth poses for the camera were captured using a motion capture system, which uses the visual markers attached to the camera to determine its pose accurately. Credit for this setup is for the Technical University of Munich's Computer Vision Group [1].

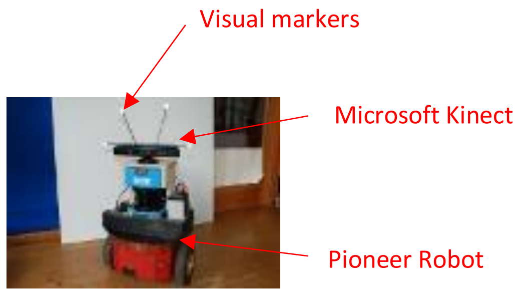


Figure 1: Setup used by TUM CV group to capture the video sequences [1]

The video sequence considered in this paper is 'freiburg2_pioneer_slam3' which is available on the TUM CVG website [1]. The ground truth trajectory is shown in figure 2.

IV. RGB-D VO Pipeline

The formulation of the problem can be summarized using figure 3, where the trajectory of the camera is estimated relative to frame I, which is the initial position of the vehicle at time 0.

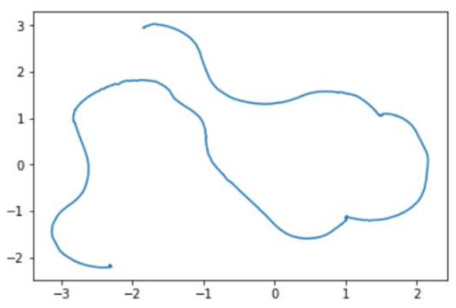


Figure 2: Ground Truth trajectory of camera for 'freiburg2_pioneer_slam3' video sequence

At every two consecutive camera frames (C_{k-1} & C_k), features are extracted and matched between the camera frames, the point cloud alignment algorithm is then used to recover the change in pose between the two frames, then the pose changes are accumulated in order to produce the final trajectory.

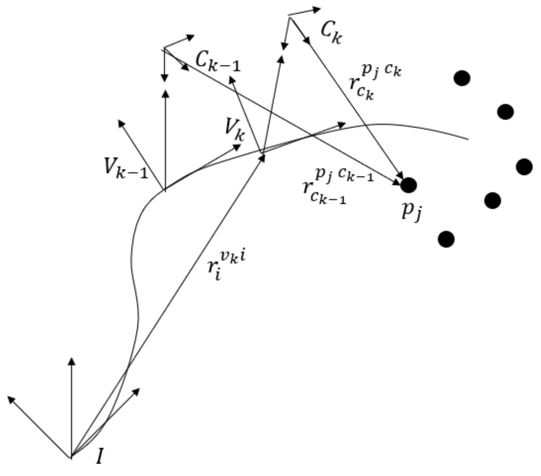


Figure 3: Diagram of reference frames

The sensor model for the RGB-D sensor is given by formula (1) below:

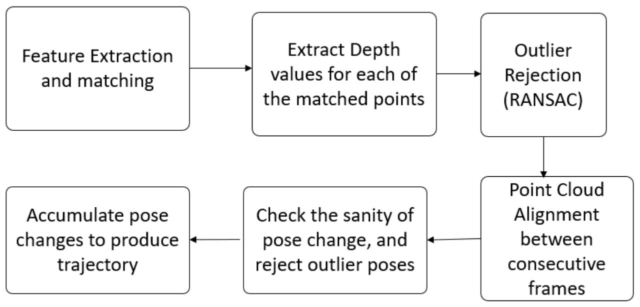
$$\begin{bmatrix} u \\ v \\ z \\ 1 \end{bmatrix} = \begin{bmatrix} f_u & 0 & 0 & c_u \\ 0 & f_v & 0 & c_v \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1/z & 0 & 0 & 0 \\ 0 & 1/z & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}_{C_k} \quad (1)$$

The transformation between the camera frame to the vehicle frame is given by formula (2) below:

$$\begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}_{C_k} = \begin{bmatrix} 0 & -1 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}_{V_k} \quad (2)$$

Note that the actual rigid transformation between the pioneer robot frame and the Kinect was not considered, since the ground truth trajectory was captured using visual markers attached to the camera itself not the vehicle. Formula 2 just rotates the camera frame to align it with the direction of motion.

The RGB-D pipeline proposed in this paper can be summarized in figure 4. The following sections provide a detailed explanation of the each of the steps presented in figure 4.



Feature Extraction and matching
Extract Depth values for each of the matched points
Outlier Rejection (RANSAC)
Point Cloud Alignment between consecutive frames
Check the sanity of pose change, and reject outlier poses
Accumulate pose changes to produce trajectory

Figure 4: RGB-D Visual Odometry Pipeline

A. *Feature Extaction & Matching*

The first step in the proposed VO pipeline is feature extraction from the RGB images (assuming that the images are already calibrated). ORB features [7] were selected for use since they are extremely robust [8], and their computation time is much faster than other popular features. In every camera frame, 5000 ORB features were generated, the reason for the large number is because as the features get filtered through the pipeline, their number gets severely reduced, hence starting with a large number guarantees the survival of some features through the pipeline. Now that features from two consecutive camera frames have been generated, they are matched using their descriptors. A Brute Force Matcher [12] is used for this step with Hamming distance as its criterion.

B. *Depth extraction*

For each of the matched pair of features (in the two consecutive camera frames), the depth is extracted from the Microsoft Kinect Depth map. The features are then filtered using the below conditions:

$$if\ z = 0\ or\ if\ z > 5 \rightarrow \text{reject feature}$$

The z=0 condition is to filter out features with undetected depths. As for the z>5 condition, it was noticed that when features with depths larger than 5 m were included, the accuracy of the predictions was degraded significantly, that's because the uncertainty in the Kinect's depth measurement grows quadratically with depth as proved by K. Koshelham & S. Elberink in [5]. Although, they recommended to use depths that only lie within 1 to 3 meters, that range was found to be very tight for the application at hand, at numerous camera frames almost no features were detected between 1 to 3 m. Hence, as a compromise, the range from 0 to 5 meters was selected and was found to produce acceptable results.

C. *Outlier Rejection*

The outlier rejection algorithm used is variation of the popular RANSAC algorithm [9]. The steps for the outlier rejection process are summarized below:

```
For 36 iterations:
a.  Randomly select 3 of the matched features
b.  Perform point cloud alignment between consecutive
    frames using the 3 selected features (algorithm explained
    in the following section)
```

c. Use the pose change resulting from the point cloud alignment to transform all features in frame k to frame k-1.
d. Calculate the norm distance between detected features and transformed features in frame k-1
e. If norm <0.08 m, keep as inlier, else discard as outlier
f. Calculate the variance of the inliers
g. Keep the set that has the largest (size x variance)

Note that steps a to e are the standard for the RANSAC algorithm, where 3 features are selected randomly and used to perform point cloud alignment from frame k to frame k-1 (where k stands for time index). The resulting pose change from the point cloud alignment is then used to transform all the features from frame k to k-1. The transformed features are then compared to the detected features in frame k-1, and the Euclidean norm is then computed. Features with a norm less than 0.08 m are kept as inliers. Note that 0.08 m was selected based on the analysis performed in [5], where it was deduced that the standard deviation of the Kinect's depth sensor is around 0.018 m at max, hence using a 4-sigma envelope, 0.08 threshold was selected.

Steps f and g slightly differ from the standard RANSAC algorithm. The purpose of computing the variance of the inliers in step f is to measure how spread out the inliers are in the image, since it is undesirable to have all the inliers concentrated in one location in the image. Therefore, step g has an objective to maximize the number of inliers as well as the variance of the inliers, therefore the inlier set with the largest size multiplied variance is kept. Note that the variance computation for step f is done only in the u and v directions of the image, the depth is not considered for the variance computation.

The number of iterations was computed using formula (3) below, based on estimating the success probability and the inlier probability:

$$k = \frac{\ln(1-p_{suc})}{\ln(1-p_{in}^4)} = \frac{\ln(1-0.9)}{\ln(1-0.5^4)} = 36 \quad (3)$$

D. Point cloud alignment

The outlier rejection step outputs a set of inlier features, which are used to recover the pose change between consecutive camera frames using the scalar weighted point cloud alignment algorithm [10]. Below is a brief summary of the algorithm:

1. For two sets of point clouds (sets k-1 & k), the sensor measurements are converted to euclidean point measurements in vehicle frame:

$$p_{k-1}^j = T_{CV} T_{sensor} \left[u_{k-1j}\ v_{k-1j}\ z_{k-1j}\ 1\right]^T \quad (4.1)$$
$$p_k^j = T_{CV} T_{sensor} \left[u_{kj}\ v_{kj}\ z_{kj}\ 1\right]^T \quad (4.2)$$

2. Then the centroids of both point clouds are computed:

$$p_{k-1} = \frac{1}{w}\sum_{j=1}^{N} w_j p_{k-1}^j \ , \ p_k = \frac{1}{w}\sum_{j=1}^{N} w_j\, p_k^j \quad (4.3)$$

$$w_j = 2/(z_{kj} + z_{k-1j}) \quad (4.4)$$
$$w = \sum_j w_j \quad (4.5)$$

Note that the weight w_j is essentially $\frac{1}{z_{avg}}$ for points in frames k, and k-1. This will result in closer points with lower depth variance to be weighted higher than far away points that have a higher depth variance, hence enhancing the accuracy of the results.

3. Compute W matrix:

$$W = \frac{1}{w}\sum_{j=1}^{N} w_j (p_k^j - p_k)\ (p_{k-1}^j - p_{k-1})^T \quad (4.6)$$

4. Construct pose change by performing singular value decomposition of W:

$$C_{k,k-1} = V \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & detUdetV \end{bmatrix} U^T, \ where\ VSU^T = W \quad (4.7)$$

$$r_{k-1}^{k,k-1} = -C_{k-1,k}^T p_k + p_{k-1} \quad (4.8)$$

$$T_{k-1,k} = \begin{bmatrix} C_{k,k-1} & r_{k-1}^{k,k-1} \\ 0 & 1 \end{bmatrix} \quad (4.9)$$

E. Sanity checks of resultant pose change

Now that the resulting pose change has been estimated using the method presented in the previous section, one must conduct some sanity checks on the pose change in order to ensure that it is realistic. It was found that these checks were crucial when the camera frames were blurry or corrupted. Below is snippet of pseudo code illustrating of the checks applied:

```
n=1
while (k+n) < K:

    #Check 1:
    If n>10:
        k=k+1
        n=1

    Perform feature extraction, matching and outlier rejection
    between frames k & k+n.

    #Check 2:
    If number of final features <10:
        n=n+1
        Jump to beginning of loop

    Perform point cloud alignment between frames k & k+n.

    #Check 3:
    if $|r_k^{k+n,k}| > 0.2$m :
        n=n+1
        Jump to beginning of loop

    k=k+n
    n=1
```

Figure 5: Pseudo Code of sanity checks

In summary, check 2 in figure 5 is for ensuring that at least 10 points are used for recovering the pose change between frames. Although, only 3 points are needed to solve for the pose change in closed form, it was found that enforcing at least 10 points enhances the accuracy of the pose estimate. As for check 3, that is to ensure that the magnitude of the estimated pose change is not too large (more than 0.2m for each time step), since such large motions did not take place when the data was gathered. Check 3 was found to fail when the detected features were concentrated in one spot in the image, so the pose change estimate was highly inaccurate. If either check 2 or 3 fail, then n is indexed by one, and the same steps are repeated for frames k & k+n (essentially skipping n frames). If the n gets too large (higher than 10), then k is indexed by one and the whole process is repeated, essentially ignoring frame k (assuming its pose change is identity).

F. Pose Accumelation

After estimating the pose between consecutive frames, the pose relative to the initial frame I can then be estimated by accumulating the pose changes over time.

$$T_{kI} = T_{k,k-1}T_{k-1,I} \quad (5)$$

V. Uncertainty Esimtation

In order to estimate the uncertainty in the predicted trajectory, a Monte Carlo Simulation was performed on randomly generated trajectories. A Monte Carlo approach was used since it is hard to solve for the uncertainty analytically (no clear solution was found in the literature for RGB-D VO), that is because the propagation of error from the actual feature locations to the final pose estimates encounters many non-linearities. The approach used to get an estimate for the covariance matrix at each time step is summarized below:

For 100 iterations:

1. Generate a random Trajectory using by sampling a random Δx and $\Delta\theta$:

   $$\Delta x_k = rand\ unif\ [0,0.03] \quad (6.1)$$
   $$\Delta\theta_k = rand\ unif\ [-0.1,0.1] \quad (6.2)$$
   $$T_{k,k-1} = \exp\left([dx_k\ 0\ 0\ 0\ 0\ d\theta_k]^{T^\wedge}\right) \quad (6.3)$$
   $$T_{k,I} = T_{k,k-1}T_{k-1,I} \quad (6.4)$$

   Note that Δx was selected to be within 0.0 to 0.03 which is the actual range of motion of the pioneer robot at every timestep (avg 0.015 m).
2. At each time step k, sample 50 random features in the camera frame:

   $$u_k = rand\ unif[0,640] \quad (6.5)$$
   $$v_k = rand\ unif[0,480] \quad (6.6)$$
   $$z_k = rand\ unif[0.5,5] \quad (6.7)$$
   $$y_k = [u_k, v_k, z_k, 1]^T \quad (6.8)$$
3. Generate previous frame measurements, and add noise to it:

   $$y_{k-1} = T_{sensor}T_{v,c}T_{k,k-1}T_{v,c}^{-1}T_{sensor}^{-1}y_k \quad (6.9)$$
   $$u_{k-1} += rand\ normal(0,\ \sigma_{pix} = 1) \quad (6.10)$$
   $$v_{k-1} += rand\ normal(0,\ \sigma_{pix} = 1) \quad (6.11)$$
   $$z_{k-1} += rand\ normal(0,\ \sigma_z = 0.02) \quad (6.12)$$

4. Perform point cloud alignment using the simulated features (generated in steps 2 and 3) to get the estimated pose.
5. Compute the error between the estimated pose (from step 4), and the actual pose (from step 1).

   $$\delta\epsilon_k = \ln\left(T_{k,I}T_{k,I}^{est^{-1}}\right)^{\vee} \quad (6.13)$$

After all iterations are done, compute the expected covariance P_k at every time step by averaging all the errors.

$$P_k = E[\delta\epsilon_k \delta\epsilon_k^T] \quad (6.14)$$

Figure 6 below show an example of the output of the algorithm, the red lines shows the 3-sigma envelope derived from P_k estimated using formula 6.14. Figure 7 shows the 3 sigma envelops plotted vs time for both x and y directions.

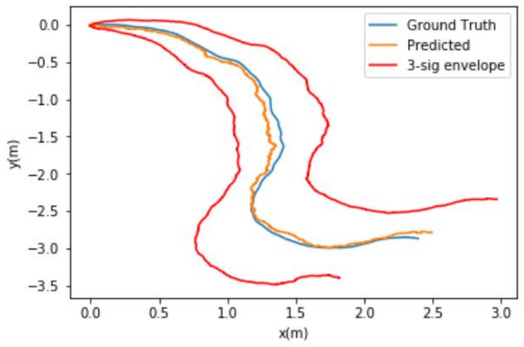


Figure 6: An example of the output of the Monte Carlo Covariance estimation

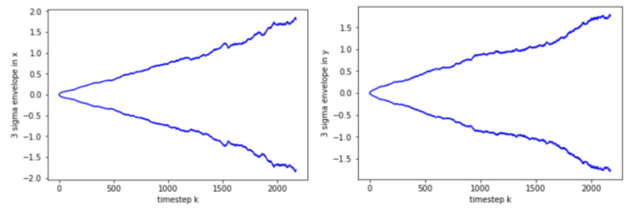


Figure 7: Three-sigma envelope for x and y values

Note that the three-sigma envelope increases in somewhat linear fashion vs time (as shown in figure 7). The slope of such line can be estimated using linear regression, and instead of repeating the simulation again, the approximate line can be used as the covariance estimate. The slope of the line can also be expressed as a function of the input noise applied to the features.

VI. Experimental Results

The pipeline was tested on the video sequence 'freiburg2_pioneer_slam3' available on the TUM RGB-D slam dataset [1]. The resultant trajectory is shown figure 8 along with the ground truth trajectory.

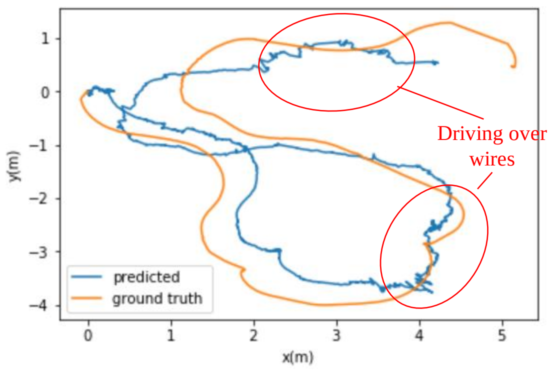


Figure 8: predicted trajectory

As it can be seen in figure 8, the estimated trajectory resembles the ground truth trajectory closely, however there is quite a significant amount of drift. Also, note that the predicted trajectory seems to fluctuate quite significantly at some areas, which correspond to where the robot drove on some wire placed on the ground. Driving on the wires (figure 9) caused the camera to shake quite significantly, and the images are quite blurry. This has resulted in not enough ORB features being detected and matched, and such blurry frames were skipped for the most part by the sanity checks in section IV E. Another issue that was encountered was that in some frames, most of the ORB features detected were far away and did not meet the depth less than 5m constraint, resulting in such frames being skipped. Such frame skipping has led to some accuracy degradation, however if a good number of close features were detected, the algorithm was found to perform well.

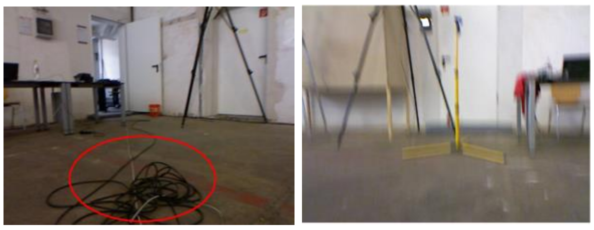


Figure 9: Driving over wires placed on the ground (left image) causing blurry images (right image)

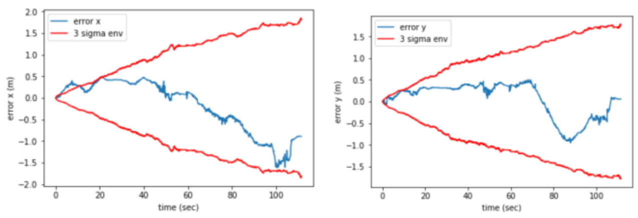


Figure 10: Errors in x & y directions along with their respective 3 sigma envelopes

Figure 10 shows the errors in both x & y directions relative to the ground truth trajectory, with their respective 3 sigma envelopes (derived from section V of this paper), the 3 sigma envelopes are the same ones showen in figure 7, but overlaid on top of the error. Note how both errors lie within their 3 sigma envelopes for the most part, which indicate that the estimate of the 3-sigma envelope is reasonable.

Although, the overall performance of the pipeline presented in this paper seems reasonable, improvements can be made to further enhance it. For example, currently the raw RGB images are used right away to extract the ORB features, but they could be processed before the feature extraction step, to increase the number of accurate features detected. In addition, better handling of the noisy depth sensor could be implemented, currently any points with depth higher than 5 meters are excluded, which limits the number of features detected. Also, the weights in the point cloud alignment algorithm are inversely proportional to the depth of the feature, however, one can try different heuristics such as having a soft threshold depending on the camera frame, or having weights that are inversely proportional to the square of the depth measurements. Finally, Bundle Adjustment [11] could be used rather than scalar weighted point cloud alignment for predicting the pose change between the camera frames.

As for the runtime of the algorithm, the code attached to the submission of this paper was not optimized to run real-time. Further enhancements to the methods and data-structures used could be applied to reduces the processing time per frame. For example, vectorized operations could be used instead of for loops, also number of ORB features detected could be decreased to enhance performance.

VII. Conclusion

In this paper, a pipeline for performing Visual Odometry using RGB-D images is presented. The pipeline utilizes scalar weighted point cloud alignment algorithm to predict the pose change between consecutive camera frames, and the pose changes are then accumulated to produce an estimate of the final trajectory. The pipeline was tested on TUM RGB-D dataset and was found to produce reasonable results, however further enhancements could be applied to the pipeline to improve its accuracy and runtime performance. In addition, a method for estimating the covariance of the drift in trajectory is presented, the method utilizes a Monte Carlo approach along with simulated measurements to output an average covariance matrix for each time step of the trajectory. The uncertainty estimate was found to be reasonable given the actual errors observed on the TUM RGB-D dataset. Finally, future work could include optimizing the pipeline presented for real-time performance, as well using the Bundle Adjustment algorithm instead of point cloud alignment to recover the pose change more accurately.

VIII. References

[1] Computer Vision Group - Datasets - RGB-D SLAM Dataset and Benchmark, 08-Mar-2016. [Online]. Available: https://vision.in.tum.de/data/datasets/rgbd-dataset. [Accessed: 31-Dec-2019].

[2] M. O. A. Aqel, M. H. Marhaban, M. I. Saripan, and N. B. Ismail, “Review of visual odometry: types, approaches, challenges, and applications,” SpringerPlus, 28-Oct-2016. [Online]. Available: https://www.ncbi.nlm.nih.gov/pmc/articles/PMC5084145/. [Accessed: 31-Dec-2019].

[3] “RGB-D Sensors Kinect” Computer Vision Group - RGB-D Vision, 08-Mar-2016. [Online]. Available: https://vision.in.tum.de/research/rgb-d_sensors_kinect. [Accessed: 31-Dec-2019].

[4] D. Nistér, O. Naroditsky, and J. Bergen, “Visual odometry for ground vehicle applications,” Wiley Online Library, 26-Jan-2006. [Online]. Available: https://onlinelibrary.wiley.com/doi/abs/10.1002/rob.20103. [Accessed: 31-Dec-2019].

[5] K. Khoshelham and S. O. Elberink, “Accuracy and resolution of Kinect depth data for indoor mapping applications,” Sensors (Basel, Switzerland), 2012. [Online]. Available: https://www.ncbi.nlm.nih.gov/pmc/articles/PMC3304120/. [Accessed: 31-Dec-2019].

[6] Fast visual odometry and mapping from RGB-D data - IEEE Conference Publication. [Online]. Available:

https://ieeexplore.ieee.org/abstract/document/6630889. [Accessed: 31-Dec-2019].

[7] E. Rublee, V. Rabaud, K. Konolige, G. Bradski, “ORB: an efficient alternative to SIFT or SURF” [Online]. Available: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.370.4395&rep=rep1&type=pdf [Accessed: 31-Dec-2019].

[8] S. A. Perera, “A Comparison of SIFT , SURF and ORB,” Medium, 18-Aug-2018. [Online]. Available: https://medium.com/@shehan.a.perera/a-comparison-of-sift-surf-and-orb-333d64bcaaea. [Accessed: 31-Dec-2019].

[9] M.A. Fischler and R.C. Bolles. Random sample consensus: A paradigm for model fitting with applications to image analysis and automated cartography. Communications of the ACM, 24(6):381–395, 1981.

[10] T. D. Barfoot “State Space Estimation for Robotics”, Cambridge UK, Cambridge University Press, 2019, pp. 297-319

[11] Triggs; P. McLauchlan; R. Hartley; A. Fitzgibbon (1999). "Bundle Adjustment — A Modern Synthesis". ICCV '99: Proceedings of the International Workshop on Vision Algorithms. Springer-Verlag. pp. 298–372

[12] D. Gada, “Feature Matching using Brute Force Matcher,” *A developers notebook*, 13-Mar-2019. [Online]. Available: https://thedevnotebook.wordpress.com/2019/03/13/feature-matching-using-brute-force-matcher/. [Accessed: 31-Dec-2019].