Visual Odometry for Corobots

Ruturaj Hagawane
Advisor: Dr. Zack Butler
Department of Computer Science
Rochester Institute of Technology

***Abstract*—Odometry plays a very important role in robotics for localization, mapping, navigation, SLAM. Traditional odometry approaches use encoders to measure distance traveled, stepper motors for precise movements by distance o visual odometry. Rochester Institute of Technology started corobtics project to inspire and aid students in learning computing fundamentals and robotics applications. Corobots uses encoders for odometry which face precision problems, since wheels slips and slides. Visual odometry is widely used technique in robotics to overcome this issue. Corobots have Kinect sensor which produces RGB image and depth data, which is used for visual odometry in this project.**

***Keywords*—Corobots, odometry, Kinect, key points, features, point cloud library, tracking, Robot Operating System.**

I. Introduction

Corobots at Rochester Institute of Technology are built to inspire students to learn computing fundamentals and robotics. Some basic nationalities are implemented on them for a student to focus more on algorithm part. Corobots are built using iRobot's Create 2 as base/legs, Chromebook running Ubuntu with ROS on it as main processing device/brain and Microsoft's Kinect as sensors/eyes. Corobots do localization using QR code placed at many places. While moving, when corobots don't have access to QR code they try to update current position and accumulates error over time because of encoders limitations.

The main idea of this project is to improve corobot's odometry and keep better track of current position. Visual odometry is widely used in robotics including mars rovers. Corobots have Microsoft's Kinect sensor which produces a RGB image and depth data. Using these, rotational matrix and the translational vector between frames can be obtained and used to track the change in position over time.

There are many approaches in visual odometry and generally, they follow similar steps. To understand those and our approaches, there few key terms we need to get familiar with.

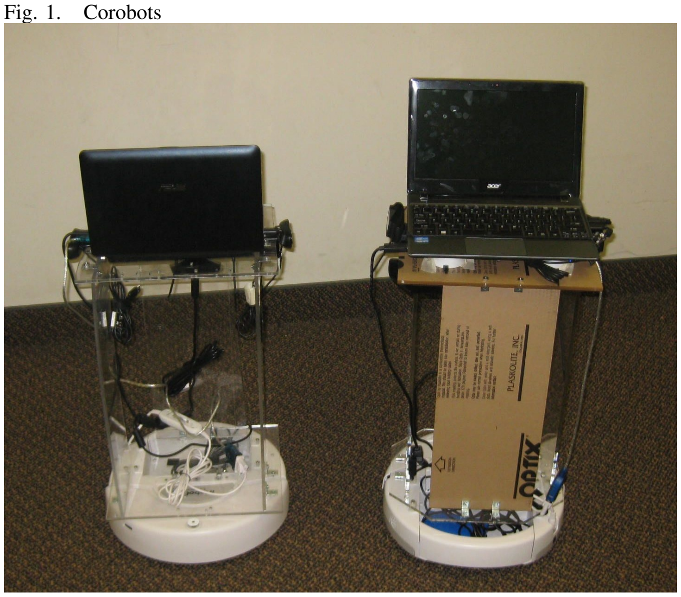
A. Point cloud

The point cloud is set of 3D points with X, Y, Z values with optional intensity value, RGB values, Normals (surface inclination) or a combination of these.

B. Key points

Key points are distinctive points in image or point cloud which are easy to track given a specified window around them.

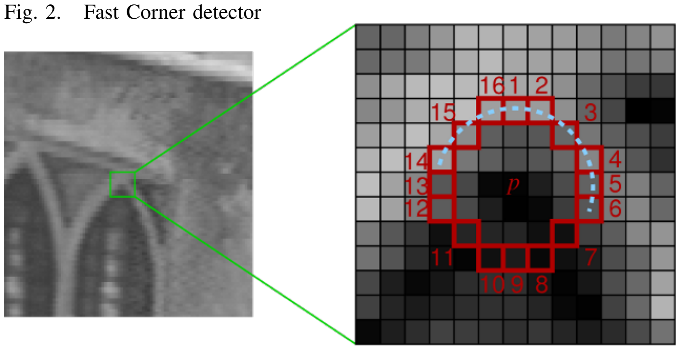
Fig. 1. Corobots



In computer vision, key points are also called as features, but we will avoid this term since term feature is used in Point cloud library differently. Generally, corners are good key points in the image. For finding key points there exists many key points detection mechanisms like SURF (Speeded up robust features), SIFT(Scale-invariant feature transform), FAST(Features from accelerated segment test) for 2D images and HarrisKeypoint3D, SIFT3D for point clouds.

We decided to use FAST corner detector[3] in final implementations of 2 of our approaches, because of computational efficiency as compared to other popular interest point detectors.

Fig. 2. Fast Corner detector



As in Fig 3, if we want to test point p if it is a corner or not, FAST draws a circle of 16 pixels circumference around point ’P’. For each pixel on the circumference of this circle, if a set of continuous pixels exists whose intensity exceeds the intensity of the pixel ’P’ by ’I’ and for another set of contiguous pixels if the intensity is less by at least I, If ’P’ satisfies this criterion, then we mark ’P’ as a corner. A heuristic is used for rejecting the majority of non-corners, in which pixel at position 1,9,5,13 are examined first where at least three of them must have a higher intensity difference of I for the point to be a corner. This approach is selected due to its computational efficiency as compared to other popular interest point detectors such as SIFT.

C. Feature descriptor

Feature descriptor represents a point in point cloud using geometrical and/or RGB/Intensity information available of that and neighboring points. Feature descriptor is used to calculate match between two points when finding correspondences.

D. Key point tracker

Key point tracker, when provided with an image with key points, tracks similar key points in the second image using local information around the point. Kanade–Lucas–Tomasi feature tracker is widely used and favored tracker for visual odometry as it takes optical flow into consideration. It is faster than traditional techniques for examining far fewer potential matches between the images.

E. Correspondences

Correspondence is a structure with 2 integers and float value. It stores which key points (query) from one set matches which key point (match) in another set with the difference between their feature descriptor (distance). This is used in the case where key point tracker is not used.

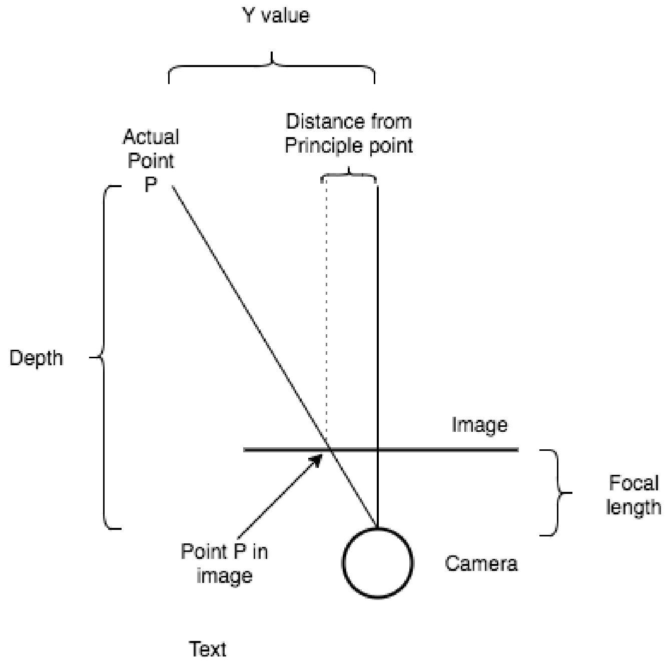
F. 3D point triangulation

To calculate a real-world coordinate of a pixel is possible if we know the depth of point, focal length f and Principal point (cx,cy) of the camera. The principal point of the camera is X and Y value of center point in an image. Let’s take the following image as an example to understand the geometry of triangulation. The fig 3. is the top view of the system. Assume there is point P (RX, RY, RZ) at depth d from the camera in the real-world captured in an image at (ix,iy). Since the triangle formed by the camera, point P in image and image is similar triangle to triangle formed by the actual point P, and depth d. So the ratio between actual x value RX to distance between point P x coordinate in image ix and principal point’s x coordinate cx is equal to the ratio of depth d to focal length f.

$$\frac{RX}{(ix-cx)} = \frac{d}{f}$$

$$RX = (ix-cx)*d/f;$$

Fig. 3. Triangulation geometry



G. RANSAC

Random sample consensus (RANSAC) is an iterative method. At every iteration, it randomly samples five correspondences from given correspondences, estimates transformation and then checks how many rests of the points are inliers when using this transformation. After a fixed number of iterations, it terminates and returns transformation with a maximum number of inliers if inliers are above threshold or returns unit matrix. The default inlier threshold is 5% of all points.

H. Transformation matrix

Transformation matrix consists of relative rotational matrix R and translation vector T, between two images or points sets. Suppose A and B are two images of the point cloud of the same scene from two different angles. If we calculate transformation matrix T for A to B then by applying transformation T on Point A_i in A will convert point A_i to the corresponding point B_i in B.

$$B_i = R * A_i + T$$

II. Similar Work

There has been a lot of work in Visual odometry using a monocular camera, stereo camera, RGB and depth camera. Most approaches are based on following stages:

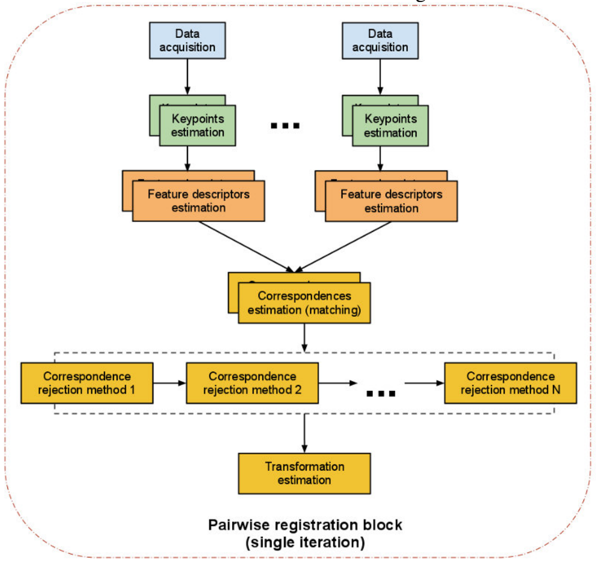
1. Acquire input images
2. Image processing techniques like filtering, converting formats, lens distortion removal
3. Feature detection
4. Feature tracking/matching

5. Camera motion estimation

Lucas–Kanade did great work in feature tracking which is widely used for calculating optical flow. Kanade–Lucas–Tomasi feature tracker[4] uses spatial intensity information to searching for the position for the best match. Making it efficient in faster approach than brute force matching. It works efficiently over the movement of a single camera taking picture of the same scene from slightly different viewpoints, but it assumes illumination of the scene is constant, which leads to failure in some cases.

Dirk Holz, Alexandru E. Ichim, Federico Tombari, Radu B. Rusu and Sven Behnke[5] published an overview on registration algorithms, usage examples for Point Cloud Library (PCL)[2] and the tools therein available. Their registration estimation pipeline is as in fig 4.

Fig. 4. Transformation estimation in PCL block diagram


Data acquisition
Data acquisition
Keypoints estimation
Keypoints estimation
Feature descriptors estimation
Feature descriptors estimation
Correspondences estimation (matching)
Correspondence rejection method 1
Correspondence rejection method 2
Correspondence rejection method N
Transformation estimation
Pairwise registration block (single iteration)

They explained different modules in PCL and 3 approaches for the different type of data. For their work on RGB-D data obtained from Kinect, they used surface normals which represent inclination of the surface at that and surrounding points. Then they used normal space sampling which consists of binning each point into M^3 bins based on the normal angle around each of the x, y, and z-axis, and then extracting fixed number of samples randomly from each bin. This sampling ensures that the resulting point cloud has samples from all the differently oriented surfaces in the scene, increasing the probability that the registration will converge to the global minimum. There correspondence estimation used nearest neighbor search using kd-tree. So they were comparing a point in one cloud to every other cloud but by putting And then they used iterative error minimization algorithm to calculate transformation. Since their objective was to register/fuse already recorded RGB-D data together, they used all frames of data taken at 30 frames per second. They talked about to reduce accumulating error during each registration is to use loop closing via graph optimization or forms of global registration.

This approach can be used to build a 3D cloud map by recording data first, but it is not real time. Finding correspondence by comparing feature with every point in other point cloud is not a good idea when we have

III. Implementation

The implementation of this technique was done using 3 different approaches. The first method was tested on KITTI data set only since it required scale information. For other two, Camera calibration for Kinect was done for both RGB and Infrared camera. For calibrating Infrared camera, the Infrared blaster was covered with stick page to avoid noise. Camera calibration package from ROS was used to calibrate RGB and IR camera. Freenect library in ROS was used to receive data from Kinect. The RGB image received using freenect node is in bayer format which uses a single plane with storing value for one color per pixel. Image from Bayer format was converted to BRG format and then bilateral filter was applied before converting 3 planner BRG image to single planner intensity image. Depth images have from freenect holds depth values for a pixel in mm. Since RGB camera and IR camera on Kinect are 4 cm apart, the registered flag of the freenect launcher was set to 1. Registered depth image is aligned depth image to RGB image instead of original depth image. Since Freenect publishes RGB and depth image separately, ApproximateSync policy in ROS was used, so images received in the pair of RGB and Depth who have an approximately same time stamp.

A. Approach 1

The first approach was motivated from general approach of visual odometry. Experiments with different key points estimation were done in this approach. This approach was implemented for 2D images with external scale information provided since transformation obtained from monocular camera contains unit vector as translation vector which needs to be scaled. Images were read from the storage device. For implementing this approach following algorithm was followed

Algorithm 1: Monocular visual odometry algorithm with scale information provided

```
1  function track ($image$);
   Input  : A image pointer in Bayer format $image$
   Output: $transformationMatrix$
2  Convert image to BRG image;
3  Filter BRG image;
4  Convert Filtered image to Gray Scale image;
5  Find key points in previous image;
6  Track key points from previous image in to current
     image;
7  From all tracked key points select best 5 using RANSAC;
8  Compute currentTransformation using those key points;
9  finalTransformation = currentTransformation *
     finalTransformation;
10 previous Image = current Image;
```

For key point detection FAST, SURF and SIFT were used in three different implementations. For key point tracking KLT tracker was used. To compute transformation RecoverPose function from OpenCV was used.

Experimentation was done to use this algorithm with corobots, by calculating the difference in depth at feature points. But because of noise and since there was no outlier detection for depth possible the scale values were inconsistent.

B. Approach 2

The second approach was inspired from PCL library. PCL library uses real-world 3D coordinates for points. By using these points triangulation wasn't needed.This approach was implemented by subscribing to point cloud as input from Freenect.

Algorithm 2: Visual odometry algorithm for point clouds

```
1  function track (image);
     Input  : PointXYZRGB point cloud inputCloud
     Output: transformationMatrix
2  Find key points in previous cloud;
3  Find key points in current cloud;
4  Find feature descriptor for previous key points;
5  Find feature descriptor for current key points;
6  Find initial correspondences using kd-tree;
7  Computer final correspondences;
8  Compute currentTransformation using final
     correspondences;
9  finalTransformation = currentTransformation *
     finalTransformation;
10 previous cloud = current cloud
```

To find key points in point cloud SIFT, HarrisKeypoint3D was used. Key point estimation on 3D point cloud was computational heavy and resulted in longer execution time per iteration. For feature descriptor, RIFT(Rotation Invariant Feature Transform) was used. Initial correspondences were found using kd-tree matching which takes $O(nlogn)$ time for n key points. Final correspondences were computed using RANSAC and Levenberg-Marquardt non-linear least squares minimization methods.

C. Approach 3

This approach was inspired by an idea that not all points need triangulation for transformation estimation and key point estimation and tracking is faster, robust and efficient on 2D images. In this approach, RGB and depth data aligned to RGB image was taken as input from Kinect, the key points were estimated and tracked, 3D point triangulation was done only for key points. Inlier detection was done on key points since we could have got the wrong match or depth value was wrong. To detect inliers threshold of 5 cm was applied.

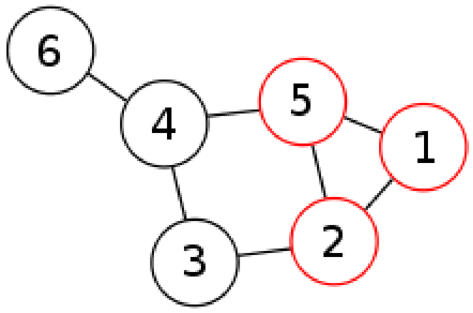
For key point detection FAST and for key point tracking KLT tracker was used. The 3D points triangulation was done using RGB camera's intrinsic parameters and depth value at coordinates of the key point.

Algorithm 3: RGB-D visual odometry algorithm

```
1  function track (image);
     Input  : A image pointer in Bayer format image
     Output: transformationMatrix
2  Convert image to BRG image;
3  Filter BRG image;
4  Convert Filtered image to Gray Scale image;
5  Find key points in previous image;
6  Track key points from previous image in to current
     image;
7  Triangulate 3D point for key points;
8  Perform Inlier detection on 3D points;
9  From all tracked key points select best 5 using RANSAC;
10 Compute currentTransformation using those key points;
11 finalTransformation = currentTransformation *
     finalTransformation;
12 previous Image = current Image;
13 previous depth = current depth;
```

The inlier detection step was based on assumption that the surface in the image is rigid and doesn't change form between frames. That is the actual distance between 2 points stays same between frames. If the distance between 2 points from the previous frame is different from the distance between corresponding points from the current frame, then either 3D triangulation/depth data or matching is wrong. To detect inliers we compute graph with all key points as nodes. Two key points in the graph are connected by an edge only if the distance between them is same in both frames. Now we want to find a subset of the graph such that all points in that subgraph are consistent with each other. This is similar to maximum clique subgraph problem, where a clique is the subset of a graph, that contains nodes that are all connected to each other. This is known NP-hard problem.

Fig. 5. Example maximum clique graph



Finding maximum clique graph was solved using greedy heuristic search.

1. Find a node with maximum edges and put it in the clique.
2. From remaining nodes, find nodes which are connected to all nodes and put it in set S.
3. From S select node with maximum edges.

Repeat this till we have potential nodes which are not in the clique but are connected to all nodes in the clique. This inlier improves RANSAC estimation time.

IV. Experimentation

The first approach was tested on KITTI data set. Since image had a higher resolution around 2000-3000 features were detected per frame using FAST. KLT tracker tracked most of the key points in all 3 methods. SIFT and SURF detected a similar number of features with higher computation time. The accuracy was very similar using all keypoint detectors.

Fig. 6. Example input image

Fig. 7. Detected Key points using FAST

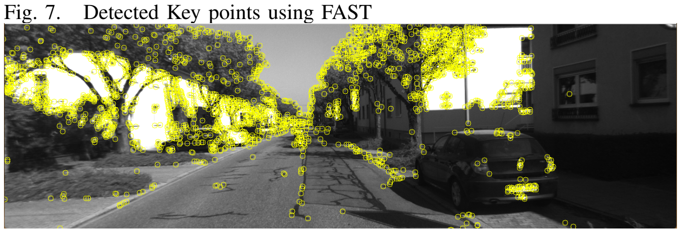
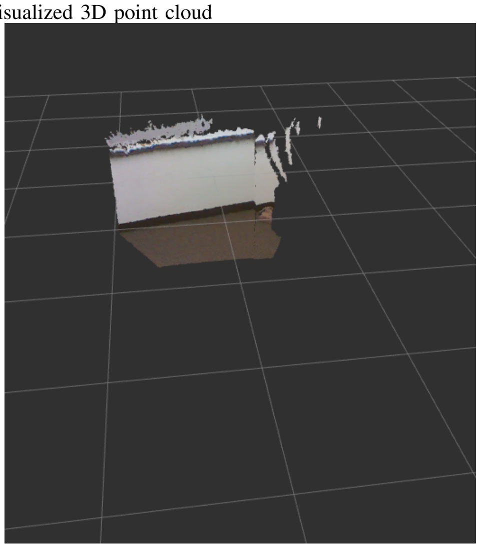


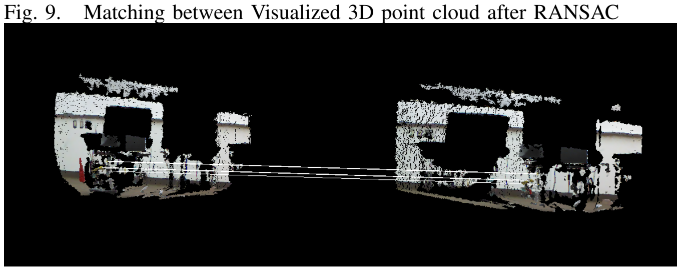
Fig. 8. Visualized 3D point cloud



In the second approach, Harris 3D key point detector was slowest while SIFT was fastest. Matching key points based on features leads to n^2 matches. This approach doesn't consider optical flow, so matching was inefficient in terms of computation and contained many mismatches. In a case of surfaces where a design pattern is repeated multiple time, this approach will fail. Using tree-based matching computational time was reduced to $O(nlogn)$ compared to $O(n^2)$, but it was still very high. Levenberg-Marquardt non-linear least squares minimization method was the most computational heavy process and caused an increase in processing time to 8-10 seconds per frame.

The second approach was tested in controlled environment in the lab with ambient and consistent light and enough features by moving the robot forward straight, at right angles, at $45°$ and a combination of these steps.

Fig. 9. Matching between Visualized 3D point cloud after RANSAC

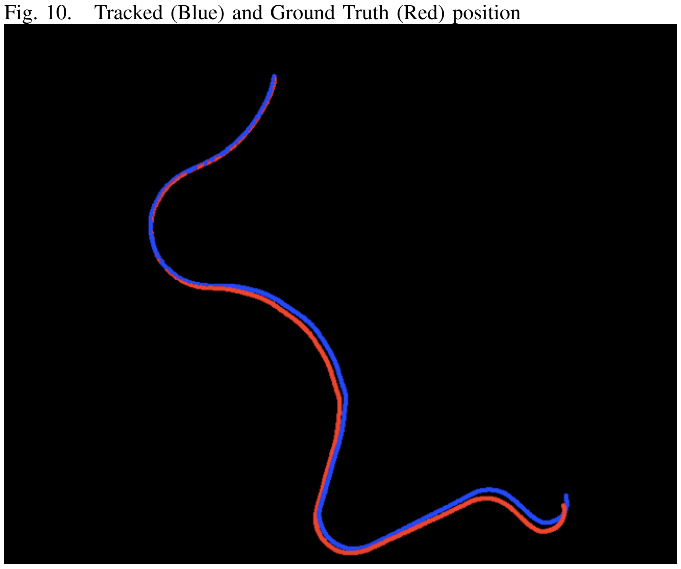


The third approach performed well, though computation time was high it was still much lower compared to the second approach. The accuracy was pretty good when tested in the lab by moving robot over smaller distances with many key points available. The accuracy was dependent on key points in image, availability, and accuracy of depth values. When taken into hallway, it could not find enough key points and sometimes computed the wrong transformation with limited key points.

V. Results

We tested approach one on KITTI data set, Fig 10 shows the result for the data set 2. The red color is ground truth and the blue color is estimated position.

Fig. 10. Tracked (Blue) and Ground Truth (Red) position



We measured the time taken by each algorithm for a single iteration of computing transformation.

Algorithm	Time(Seconds)
Monocular using SIFT	0.5
Monocular using SURF	0.5
Monocular using FAST	0.2
Using PCL using SIFT and RANSAC	5
Using PCL with Harris and RANSAC	8
Using PCL with SIFT and Levenberg-Marquardt	8
Using PCL with Harris and Levenberg-Marquardt	13
Using our approach	1

When robot was tested in Lab by moving over small distance (30-100 cm) in straight lines to make shape of square, right angles, turns of 45 degree and combination of these moves, the difference between final position and estimated position was less than 5% of total distance travelled from starting point for approach 2 and 3.

When detected features > 100

Algorithm	Accuracy
Using PCL	+-5%
Our approach	+-5%

For the last state, the robot was tested in lab and hallway by moving over major distance, distance accuracy wasn't measured in this case, only estimated key points and inliers were measured. The algorithm failed in the hallway when it couldn't find enough features for RANSAC or enough inliers while running RANSAC.

Place	No. of Key points	Inliers
Lab	40-250	30-150
Hallway	5-30	0-5

VI. Future Work

Use of better camera will result in less motion blur, better camera image can produce more features to track. Depth data produced by Kinect had limitations like only working within a range of 50 cm to 5 m, use of better depth sensing technique or use of stereo cameras might result in lesser outliers. Faster processor or use of GPU will result in processing of more frames per second. Places where none to very few key points are found, instead of finding feature, sampling can be done on the whole cloud and transformation can be estimated. This approach will require very high computation power, possible only by use of Discrete GPU. A lot of other small tasks in image processing can be done using GPU in parallel in the shorter amount of time as well. Complete RGB-D SLAM is possible by storing all point in point cloud after applying a standard transformation with calculated transformation.

VII. Conclusion

Detecting and tracking key points in a 2D image is faster, accurate and requires lesser computation power. The combination of FAST and KLT tracker works best in terms of speed and while FAST has no effect on accuracy KLT improves accuracy in this case. Inlier detector on calculated 3D key point cloud improves accuracy causing RANSAC to fails less often. Visual odometry works better than encoders in good lightning when ambient features to track and accurate depth values are available. But fails in cases where light is inconsistent, few features to track or depth values are lacking, places like hallways, dark room, tunnel plain wall.

Acknowledgment

I would like to thank Dr. Zack Butler for his inputs, guidance and immense support during my Masters and this project and Prof. Joe Geigel for his valuable inputs, guidance for project and presentations. I would also like to thank Rochester Institute of technology for providing the infrastructure required for this project.

References

[1] Robot Operating System,*h*ttp://www.ros.org/

[2] Point cloud Library, *h*ttp://www.pointclouds.org/

[3] Edward Rosten and Tom Drummond. Fusing points and lines for high performance tracking. October 2015.

[4] Bruce D. Lucas and Takeo Kanade. ADetection and Tracking of Feature Points 1991.

[5] Dirk Holz, Alexandru E. Ichim, Federico Tombari, Radu B. Rusu and Sven Behnke. Registration with the Point Cloud Library PCL: *A* Modular Framework for Aligning 3D Point Clouds. September 2015.