Monocular Pose Estimation to Track Biomedical Equipment

A Project report

Submitted by

Ashfakh Rithu

in partial fulfilment of the requirements

for the award of the degree of

BACHELOR OF TECHNOLOGY

DEPARTMENT OF ELECTRICAL ENGINEERING

INDIAN INSTITUTE OF TECHNOLOGY MADRAS

May 2017

THESIS CERTIFICATE

This is to certify that the thesis titled Monocular Pose Estimation to Track Biomedical Equipment, submitted by Ashfakh rithu k, to the Indian Institute of Technology, Madras, for the award of the degree of Bachelor of Technology, is a bona fide record of the research work done by him under our supervision. The contents of this thesis, in full or in parts, have not been submitted to any other Institute or University for the award of any degree or diploma.

Prof. M Manivannan
Project Guide
Touch Lab
Biomedical Engineering Group
Department of Applied Mechanics
IIT-Madras, 600036

Prof. Kaushik Mitra
Project Co-guide
Image Processing and Computer vision Lab
Department of Electrical Engineering
IIT-Madras, 600036

Place: Chennai
Date:

Acknowledgments

I would like to thank my project guide, Dr. M Manivannan, for all his supervision and help during the project work. I would also like to thank Dr. Kaushik Mitra for being the co-guide of our project. I would also like to extend my sincere thanks to the people at Haptics Lab – Joseph Isaac, Ravalli Gourishetty, Chandrasekkhar Burlee and Prabhu Vasulingam for helping me out at different stages of my project work. Lastly, I thank my friend and class mate A.Varunkumar, for being my project partner and a good friend.

TABLE OF CONTENTS

1 ABSTRACT

The project was initially intended to use the RIFT technology of tracking a head mounted display in virtual space used by Oculus rift and use the same principles to track Biomedical Equipment in Virtual reality. The tracking system is then used to track the props (like, needle, stethoscope etc.) in the ACLS project in Haptics Lab. The system we implemented is an accurate and efficient position estimation system based on infrared LEDs used as Markers on the target object. These predefined markers are observed by a single camera of fixed resolution that is equipped with an infrared-pass filter. The LED correspondence is then identified using a combinatorial approach of mapping and is then kept tracked using a constant velocity model. The pose estimation of the target object is then done with a Perspective-3-Point algorithm. The system is optimized to avoid reprojection error. The system is robust to cluttered environments and illumination changes as it is operating in the Infrared spectrum. The whole project is implemented in ROS and the estimated pose can be used for a multitude of purposes.

2 INTRODUCTION

2.1 INITIAL WORK

The Motivation behind this project was to create a proper tracking system for the biomedical equipment as a part of ACLS project in the Haptics lab. We were part of the project which made a 3 DOF tracking system using stereo cameras. But the system was not robust and accurate. So we decided to come up with a robust system based on the tracking technology used by Oculus rift.

Reverse engineering of Oculus Rift was done to find out how the HMD was tracked. There are 41 IR LEDs and an IMU mounted into the Oculus HMD at predefined positions. These LEDs are captured by a synchronized camera to estimate the pose using a PnP solver. Now Oculus lacks a proper Linux SDK from its developers. So we had to work with a few packages we found online which had a lot of bugs. We found that the LEDs in Oculus can be controlled to blink at certain blinking frequencies so as to be captured by the synchronized camera. This is done so as to identify which LED is at the predetermined position. This is important since it's impossible to use a combinatorial approach in this case as there is 41 LEDs and it'll be computationally impossible to do it in runtime.

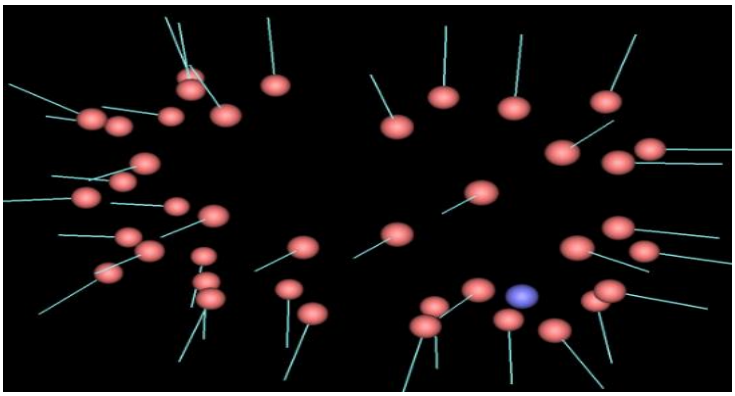


Figure 1A: 3D model of the Oculus Rift DK2 for optical tracking purposes. Red spheres are LEDs, green lines are directions of maximum emissions, and blue sphere is built-in inertial measurement unit.

There is a HID feature report (0x0C) that turns on the LEDs. It has a built-in timeout of 10 seconds, meaning the report must be resent at regular intervals by the driver software or the LEDs turn off again. After sending this feature report, the LEDs will turn on for 10 seconds, and they will show up in the tracking camera image very brightly and flicker at particular blinking frequencies. From this the synchronized camera is able to identify the LED number and then the position is estimated using the PnP solver.

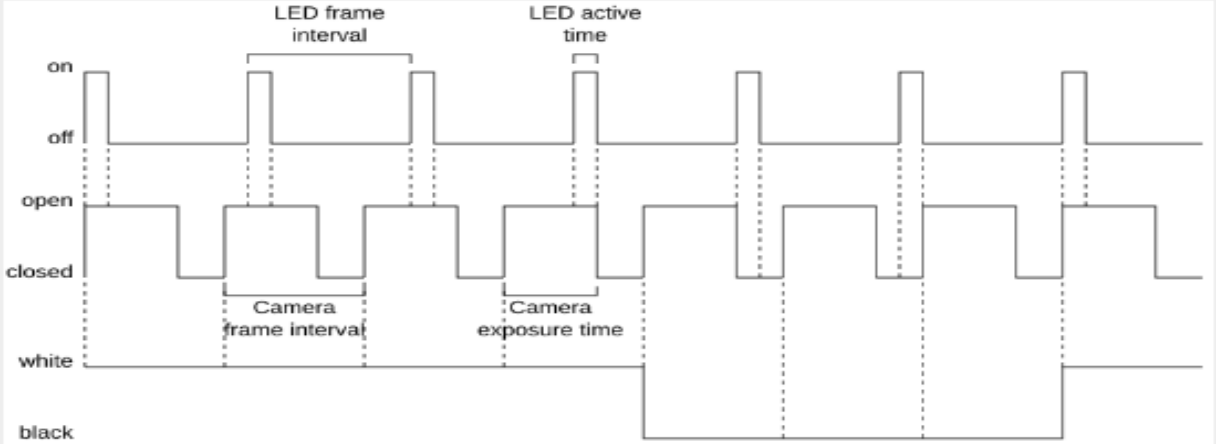


Figure 1B: Showing the active time of a particular LED with respect to the camera.

The lack of a proper Linux SDK made our work very difficult. Even though we were able to dump the estimated pose of the oculus into the terminal, we were unable to expand it so that this can be implemented with another object. More over the synchronized camera had a couple bugs which made it very difficult to use it with normal camera packages and calibration procedures. The hardware used by Oculus was difficult to replicate on a small biomedical object for it to be tracked. So we decided to come up with another tracking system which mimicked the Rift tracking system and was considerably less complex. Our work is based on pose estimation system [1] and [2].

2.2 CHALLENGES IN TRACKING

Object detection and tracking remains an open research problem even after research of several years in this field. A robust, accurate and high performance approach is still a great challenge today. The difficulty level of this problem highly depends on how one defines the object to be detected and tracked. Achieving Sub-millimeter accuracy is especially very difficult, since it is not only the challenge to detect the change in 2D plane, but also to resolve these changes into the 3D virtual world.

Minimizing the Temporal resolution in the tracking system was also a challenge. Since, this is more of a software based approach than the hardware, we couldn't give up on the performance and robustness of the algorithm. This meant that, there could be a latency between, the tracking motion and actual pose detection. There was this trade-off between latency and accuracy which we had to take into consideration while implementing the tracking system.

2.3 MONOCULAR POSE ESTIMATION

This is a pose estimation system consisting of multiple infrared LED markers on the target object and a monocular camera fitted with an infrared pass filter. The LEDs are attached to a 3D printed marker holder of predetermined marker positions. This holder can then be attached to any equipment and can be tracked accurately using the camera. The system operates in IR spectrum and is very easy to be detected from the camera image. Moreover it is not affected by illumination changes and cluttered backgrounds. The camera requires very less exposure time which lets us have higher frame rates and avoid latency. We use four IR LEDs as markers and the holder is also lightweight which makes it almost non-intrusive to the tracking object.

Estimation of the pose from 2D to 3D point correspondence is a famous problem in computer vision termed as the perspective n point problem (PnP) A minimum number of points required for a solution is Three. Three 3D to 2D point correspondence gives us four possible solutions which is then disambiguated by a fourth point. We solve the P3P problem by using the algorithm implemented in [3] which is a very fast and reliable solution compared to its previous implementations. Reference [3] for a detailed overview of PnP algorithms.

The entire tracking system is implemented in ROS [4]. ROS implementation was mentioned in [1] and it helped us in implementing the tracking system in runtime. Moreover ROS has inbuilt camera calibration procedures to obtain the intrinsic camera parameters. Camera parameters are used to improve the accuracy of the tracking system as normal USB cameras are subject to LENS distortion and other Imaging errors. Further, the ROS implementation helps us in integrating the tracking system with multiple projects very easily.

2.4 PROBLEM STATEMENT

The Aim was to create a Monocular Pose Estimation System in runtime, mimicking the RIFT technology to track objects at a sub-millimeter range of Accuracy. The system then has to be implemented to track Biomedical Equipment in VR to be used as a part of ACLS project in Haptics Lab. Most of the Libraries used were Open sourced. Other challenges included, successfully pairing all the libraries, coming up with marker models, and validating our research.

3 ENVIRONMENT SETUP

3.1 ROS

The entire system is implemented in ROS [4]. ROS is a flexible framework for writing robot software. It consists of a collection of tools, libraries and conventions that are very commonly used among developers. We setup a workspace in ROS Indigo. We built our packages in the Catkin workspace. Catkin workspace enables us to build packages as standalone projects. This is similar to cmake projects, but here we have a workspace as well. Multiple projects can be linked together and can be built all at once. This was important since we had to implement multiple packages together.

3.2 OPENCV

OpenCV (Open Source Computer Vision Library) is an open source computer vision and machine learning software library. OpenCV libraries [5] are used for marker detection, by a thresholding function. Gaussian smoothing is also used to identify the center of the blobs to sub-pixel accuracies.

3.3 EIGEN

Eigen linear algebra libraries are used for the computational purposes. They are fast, reliable and versatile. Eigen is used in the P3P problem and the constant velocity model for prediction of the next pose from the current pose.

3.4 TRACKING MARKERS

The tracking marker holder has radially emitting infrared LEDs attached to it at specified unsymmetrical points. The four LEDs attached to the marker holder helps us in determining the 6 DOF pose of the target object with respect to the camera. The

Marker positions are arbitrary but unsymmetrical. All four LEDs shouldn't be in a single plane. This is to reduce the ambiguity in pose estimation. This is also very important to the P3P solver. A 3D marker holder was designed in Solidworks and was printed using a 3D printer. The designed marker holder is lightweight and is designed in such a way that it can be attached to the object easily. Different design requirements such as avoidance of occlusion, and size constraints were also taken into consideration.

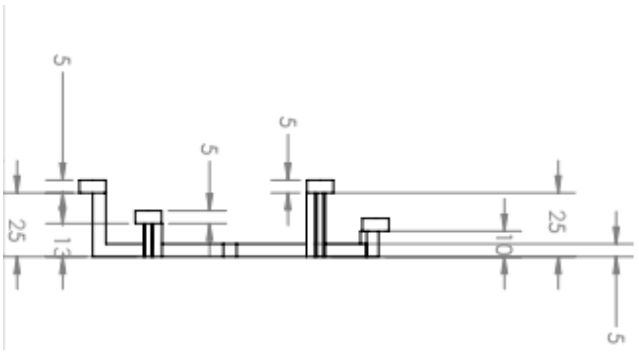


Figure 2A: Side view of the Marker Holder

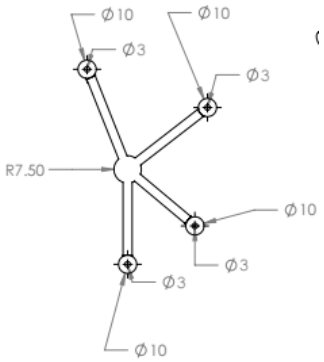


Figure 2B: Top view of the Marker Holder

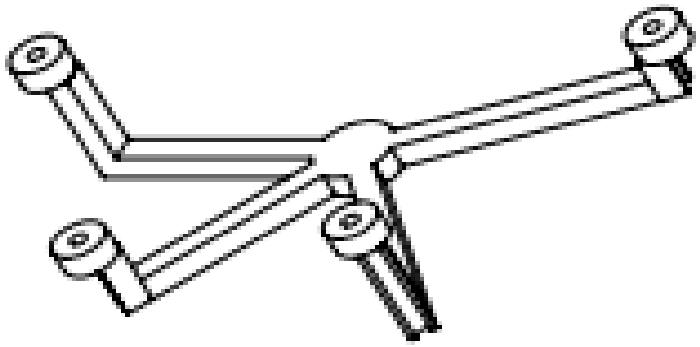
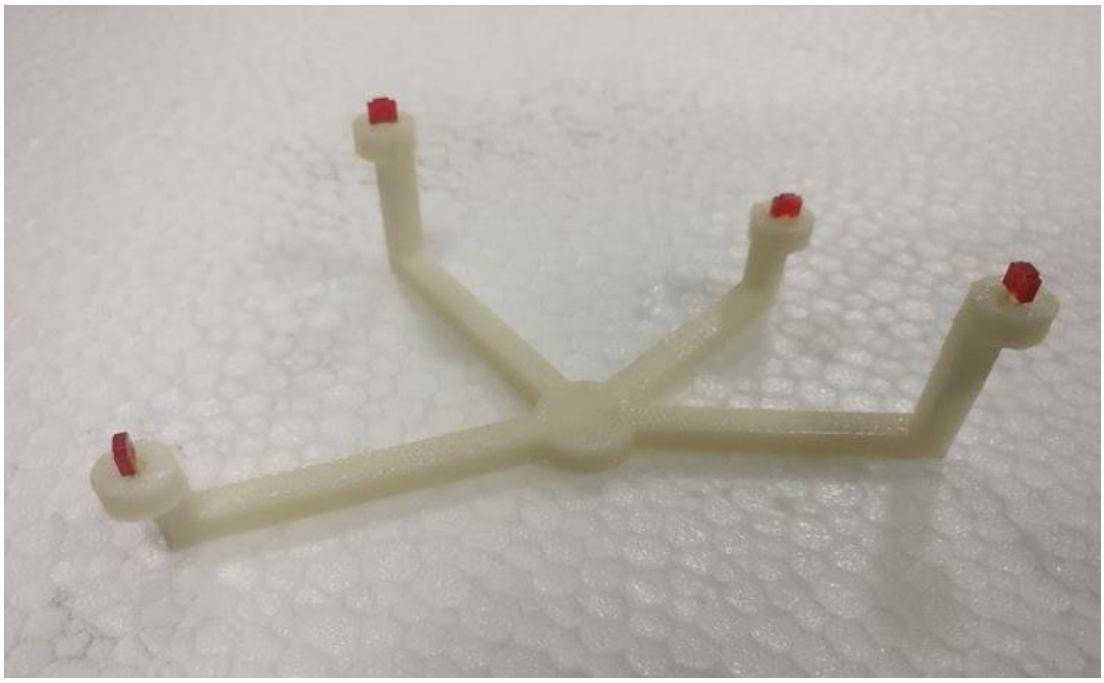


Figure 2C: 3D model



Printed 3D model with IR LEDs attached to it

The Marker positions were specified and 3D coordinates in the trackable's frame of reference is given as a YAML file called 'marker_positions.yaml'. We can track multiple objects using multiple YAML files. This file is given as an argument in the terminal.

LED Number	X	Y	Z
1	0.044	0.034	0.025
2	-0.022	0.055	0.010
3	0.000	-0.052	0.025
4	0.037	-0.031	0.013

Table 1: Table specifying the coordinates of marker positions with respect to the trackable's frame of reference. (Measurements given in meters)

3.5 CAMERA CALIBRATION

Camera calibration is done to obtain the intrinsic parameters of a camera. This is important so as to avoid errors due to distortion, rectification and projection. ROS has an inbuilt library called '*cameracalibrator.py*' which uses the OpenCV libraries to calibrate the camera. The camera calibration is done using 8x6 checkerboard. The service '*camera_calibration*' then listens to the nodes, camera/image_raw and camera/camera_info to calibrate the camera. We used a normal Logitech USB camera which has 640x480 resolution

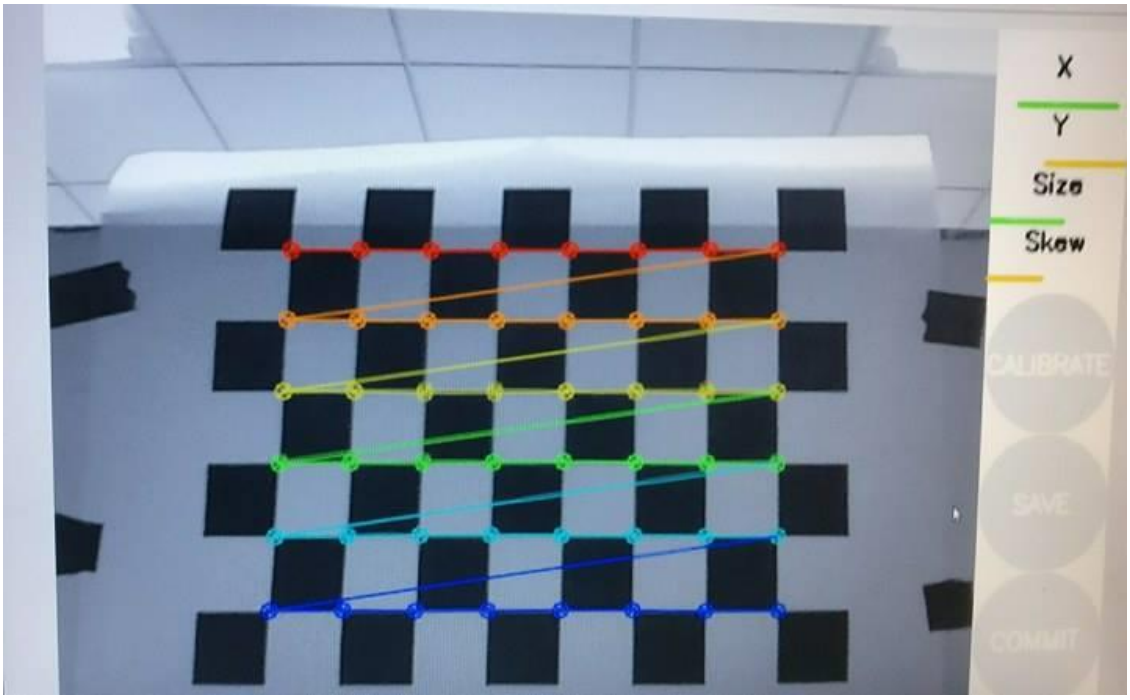


Figure 3A: Calibration of the USB camera

Once calibration is completed, we obtain the camera matrix, distortion parameters, rectification parameters and projection parameters, the calibration data is then written into a YAML file and is committed by ROS to be used as default calibration data for the particular camera node.

```
header:
  seq: 3625
  stamp:
    secs: 1245298570
    nsecs: 594038704
  frame_id: /usb_cam
height: 480
width: 640
distortion_model: plumb_bob
D: [0.063241 -0.436403 0.004366 -0.000807 0.000000]
K: [709.499013 0.000000 308.494160
    0.000000 710.795077 238.423288
    0.000000 0.000000 1.000000]
R: [1.0, 0.0, 0.0,
    0.0, 1.0, 0.0,
    0.0, 0.0, 1.0]
P: [704.308716 0.000000 307.059576 0.000000
    0.000000 710.402039 238.979036 0.000000
    0.000000 0.000000 1.000000 0.000000]
binning_x: 0
binning_y: 0
roi:
  x_offset: 0
  y_offset: 0
  height: 0
  width: 0
  do_rectify: False
```

Camera parameters saved as YAML file

Image transformation using this obtained parameters are done as a preprocessing step by ROS before running the pose estimator.

4 ALGORITHM

The algorithm is pretty straight forward and is shown in the flowchart in Figure 4. The current frame, Marker positions and previously estimated poses are given to the system as input. The first and foremost step is to detect the LEDs in the image. Once this is done, we determine the correspondence using a constant velocity prediction or an exhaustive brute force search. The optimized pose is then found out minimizing the reprojection error from all the LEDs. The covariance of the pose is also estimated.

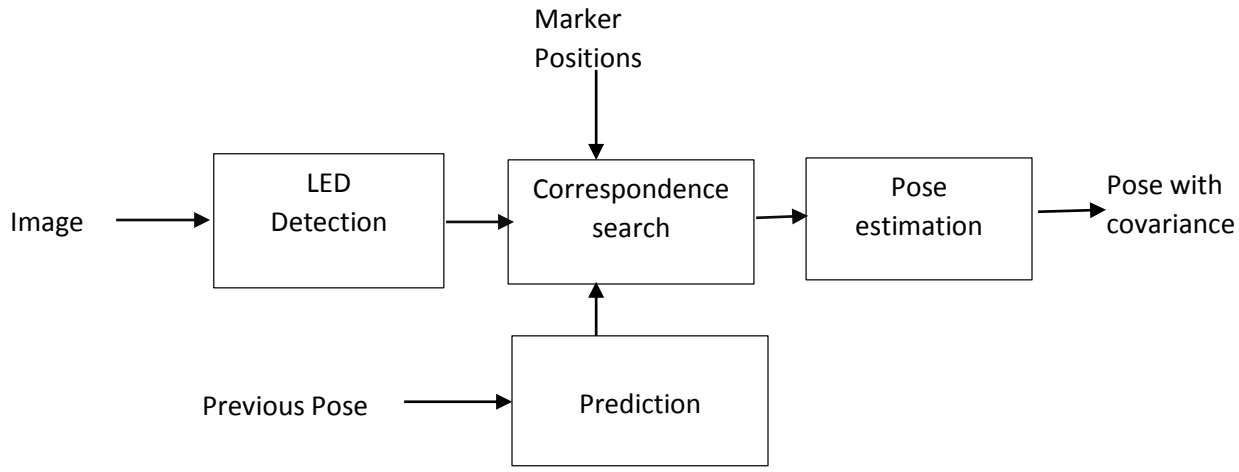


Figure 4A: Flowchart for the Algorithm

4.1 LED DETECTION

We are using Infrared LEDs whose wave length is similar to the IR pass filter we're fitting on the camera. So the IR LEDs appear as bright white blobs in the image compared to the environment which will be mostly dark. So a thresholding function can be used to identify all the LEDs.

$$\mathbf{I}'(u,v) = \begin{cases} \mathbf{I}(u,v), & \textbf{IF } \mathbf{I}(u,v) > \textbf{Threshold value} \\ 0, & \textbf{Otherwise} \end{cases}$$

The Threshold value depends on a number of parameters like shutter speed, exposure, and illumination. But with a few trial and error, we were able to find the value that works for us. The value of around 100-150 was detecting almost all the LEDs in most cases. Gaussian smoothing and group neighboring pixels to blobs to determine the center of these blobs accurately. The center is calculated using the first Image moments that are defined as:

$$M_{pq} = \sum_u u^p \sum_v v^q I'(u,v).$$

This then gives us the weighted centers of the blobs in the image. We assume, our camera is of pinhole model. So we correct for radial and tangential distortion using OpenCV libraries [5].

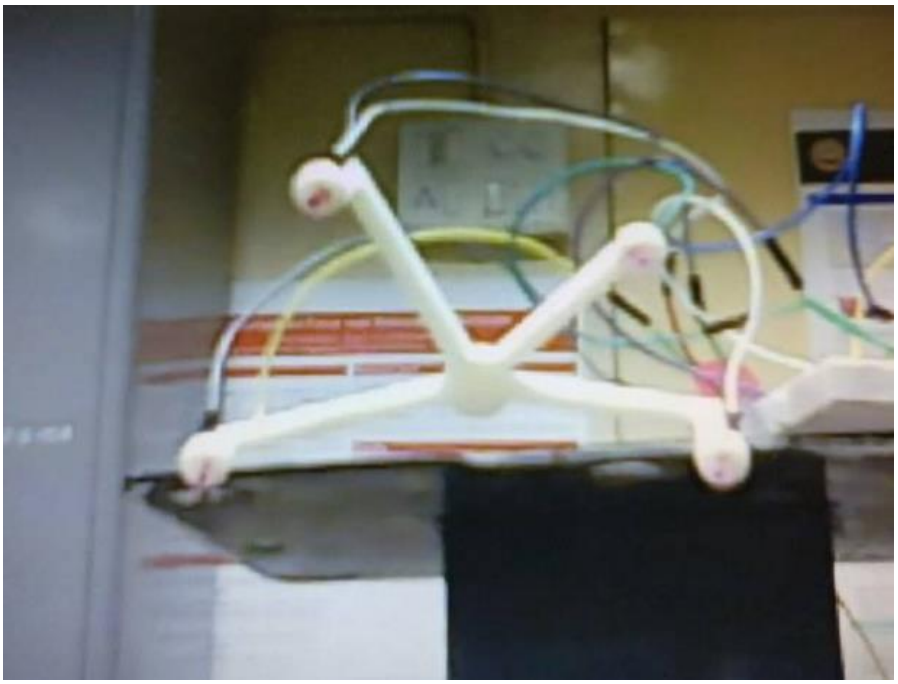


Figure 5A: Image_raw from camera without IR filter

Figure 5B: Image_raw from camera with IR filter

Once the LED detections are done the detected 2D positions is passed on for correspondence search.

4.2 POSE EVALUATION USING P3P

Given Four 2D/3D correspondence (points in 3D that are defined in the 3D model coordinate system, and points in 2D that are defined in the image coordinate system) (*A↔ u, B↔ v, C↔ w, D↔ z)*. We use three points (A, B, C) to solve the P3P equations system and thus determine up to four possible sets of distances ‖PA‖, ‖PB‖ and ‖PC‖ with P the camera optical center. The four sets of distances are converted into four pose configurations, in the end, the fourth point (D) is then used to select the best pose configuration against the "up to four" proposed.

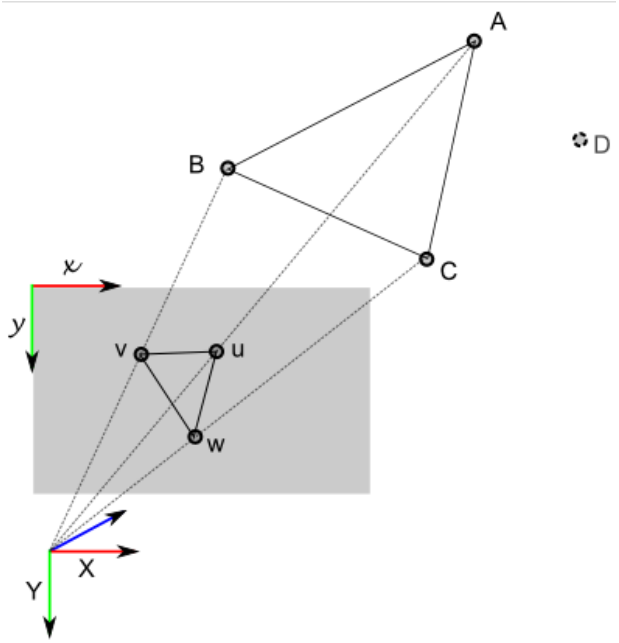


Figure 6A: 3D/2D point correspondence for pose estimation

A detailed explanation of P3P solver is beyond the scope of this Thesis, We implemented a library which uses OpenCV libraries for the P3P solver which directly takes in the 2D/3D correspondence and returns the pose estimation along with the covariance matrix. This function was then called at various instances to obtain the required pose.

4.3 CORRESPONDENCE SEARCH

The different LEDs detected in the image is not identified with their corresponding positions in the marker file, so we need to run a correspondence search in order to identify which LED is which one. Since we've 4 LEDs as markers and only 3 LEDs are required for P3P pose estimation, we can find four possible combination of any three LEDs and every permutation of LEDs in the marker holder. Now we use the fourth LED in each case and compute its pose estimated by the position matrix and re project it into the input image_raw. If the closest neighbor of the re projected LED is less than a given threshold value, then that particular pose is used. We used the re projection limit to be around 4 pixels. A histogram of each and every possible combination and corresponding re projected value is stored, and then it is used to find the one which is closest to the original image.

If n_D is the number of detections and n_L is the number of possible configurations, then, the number of total possible pose estimations are

$$N = 4 * \binom{n_D}{3} * \frac{n_L!}{(n_L-3)!}$$

This grows very quickly for increase in n_L and n_D . Since we use only 4 LEDs, this shouldn't be a problem. But in the case of Oculus rift, which uses nearly 41 LEDs, it is very difficult to determine the correct pose using an exhaustive pose estimation search. For 4 LEDs, a total of 384 pose estimation candidates are present. The search through this is done pretty quickly so as to obtain the correct corresponding LED-Marker candidates. Once they are determined, they are kept tracked using a constant velocity tracker, so that the brute force search is not necessary all the time.

4.4 PREDICTION

Once a pose is estimated using the brute-force approach, it is then not necessary to go through an exhaustive search all the time. The current pose is then calculated from previous pose using constant velocity model assuming that, the frame rate is high enough so that the markers doesn't make any jumps. If we parameterize, the pose by twist coordinates ξ, then the next set of parameters are

$$\xi_{k+1} = \xi_k + \Delta T(\xi_k - \xi_{k-1})$$

$$\Delta T = \begin{cases} 0, & if\ n_p = 1 \\ (T_{k+1} - T_k) \Big/ (T_k - T_{k-1}), & if\ n_p \geq 2 \end{cases}$$

Where T_k is the time at step *k* and n_p is the total number of estimated poses till T_k.

Now using the predicted pose, The LEDs are projected into the image, and is matched with the closest detection. We use a different threshold value as the previous step to determine whether or not this is the correct pose. We then check if the predicted correspondence are correct or not. This is done by computing the four pose candidates from four LEDs as inputs to the P3P algorithm. Then we check the projection and to know whether or not they correspond to the predicted projection. It it's correct, we save the current pose and reiterate through the same steps to find the next pose. Now in case, the predicted pose is wrong, then we initialize the tracking system and use the previously mentioned brute force approach to obtain the initial pose.

4.5 VISUALIZATION

Run time tracking of the Tracking object was one of the biggest obstacle we faced. Although the package was working very well on a prerecorded video, getting it to work on a video stream was tad difficult. But it was solved in the end by running two packages simultaneously, using a single ROS launch file and passing the usb_cam/image_raw node as the input to the video stream. The launch file specified that the video stream is from \dev/video1 and other parameters like resolution and stream type. The same launch file, after launching the camera node, launched the pose estimator. The system takes a couple of seconds to initialize and identify the correspondence between LEDs and markers. Once it figures out that part, it goes on to track the markers using the constant velocity model, obtaining the pose at each and every frame. This is then super imposed to the video to visualize the tracking in real time. The blobs are marked by red circles. A blue rectangle marks the region of interest and a RGB trivector is placed at the origin (in trackable's frame of reference) corresponding to the XYZ coordinate axis. Also the corresponding 2D coordinates of the blobs are printed to a sub-pixel accuracy. The visualized video stream is then passed to another node, image_with_detections to be viewed. This can be viewed using rqt_image_view of ROS or a simple GStreamer pipeline.

Different parameters like Threshold values, Minimum blob area, maximum blob area, sigma value used in Gaussian smoothing etc. are given at launch time. ROS lets us change the launch file parameters at runtime using rqt_reconfigure. We can use this runtime parameter change to better model the system for a particular environment according to its characteristics.

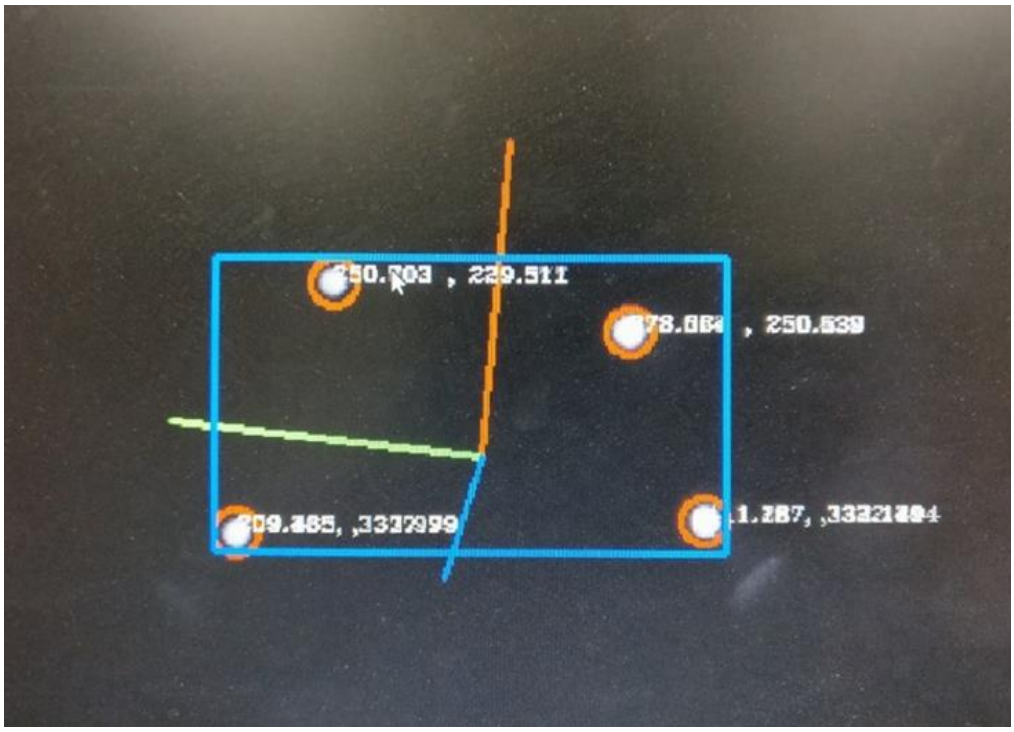


Figure 7A: Visualized image with LED blobs (Red circles), Region of Interest (blue rectangle) and the trivector. (Refer Figure 5B for camera input before the tracking)

The Visualized output stream is further sent into a different node from where the pose, the visualization and other outputs can be obtained to be used by a different program. We implemented a turtle sim at this node in-order to evaluate the performance of our system. Multiple image nodes can be created if we are doing segmentation, one for each trackable. But the input *image_raw* node remains the same, letting us track multiple object, using a single camera.

5 EVALUATION

The Evaluation scheme was to find out the accuracy and precision of the tracking system subject to various variable changes. Also different parameters given as input and their corresponding values were also found out for optimum results in a trial and error basis.

The optimal values for different parameters are given below and may subject to change based on External conditions (Illumination changes, cluttered environments etc.)

- Threshold parameter needs to be within the range of 80-180
- Gaussian sigma used in smoothing needs to be at or below 1.7. We wanted the range within which it will not false-detect LEDs where they are not present.
- Minimum blob area was found to have an upper limit of 37 (on a scale to 0-100)
- Maximum blob area was found to have a lower limit of 53 (on a scale of 0-1000)
- Max width height distortion was found to have a lower limit of .48 (on scale of 0-1.0)
- Max circular distortion was found to have a lower limit of 0.48 (on scale of 0-1.0)
- Back projection pixel tolerance was found to have a lower limit of 5 (on scale of 0-10)
- Nearest neighbor pixel tolerance was not making much of a difference. We used the value of 7 (scale of 0-10)
- ROI border thickness was given as 10 (on a scale of 10-200)

5.1 TURTLE SIM

Turtle sim is a Robot simulator used in ROS to test packages. It creates a *turtlesim_node* which listens to geometric messages that can control the simulated bot. It subscribes to geometric messages (*turtle1/cmd_vel*) which gives the linear and angular velocity commands and publishes the pose (x, y, theta, linear and angular velocity). The

simulator runs in a 2D plane with z axis taken as zero. We create a ROS node named *turtlepos* which subscribes to the pose messages from *pose_estimator* which outputs the pose data as a covariance vector. We then converted the covariance vector to a twist vector which is then subscribed to the *turtlesim_node* . We then used the position data in the twist vector to control the turtle bot.

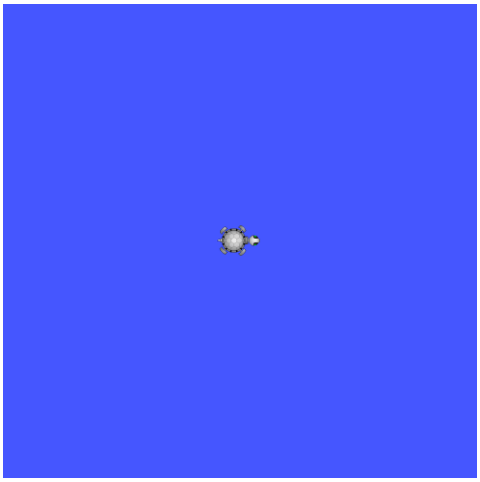


Figure 9A: Turtle Sim

5.2 TRACKING ACCURACY AND PRECISION

The evaluation measure was considered to be the distance the trackable had to move in order for the turtle sim to move two units in virtual space. Measurements were taken at different Z values.

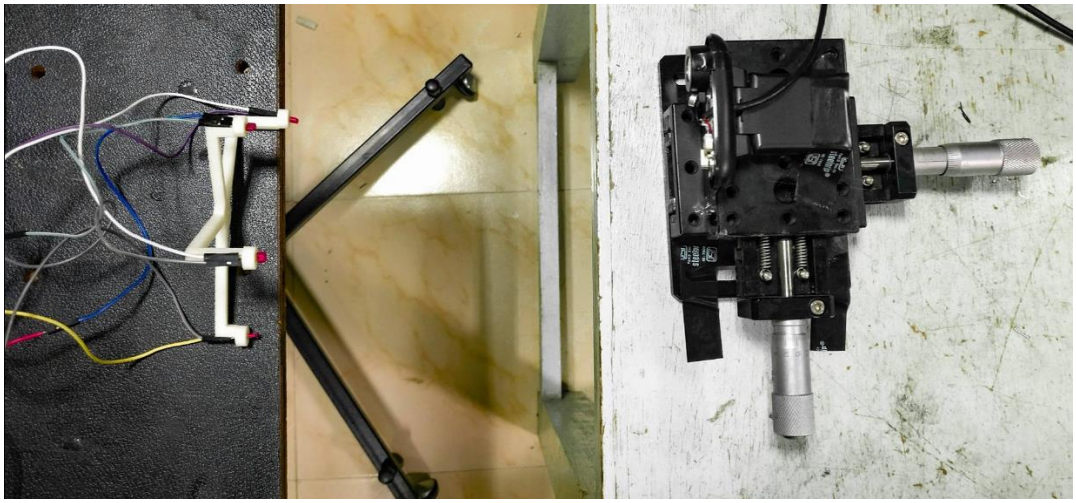


Figure 10A: Experimental setup to measure accuracy over sub-millimeter range

z-axis(cm)	Horizontal (X-axis) (mm)	Vertical (Y-axis) (mm)
20	0.4	0.43
25	0.52	0.56
30	0.7	0.72
35	0.93	0.89
40	1.3	1.4
45	1.7	1.67
50	2.3	2.34
55	3.1	3.1
60	3.9	3.8
65	4.8	4.7
70	5.8	5.9
75	6.9	6.8

Table 2: Measurements in different planes from the camera

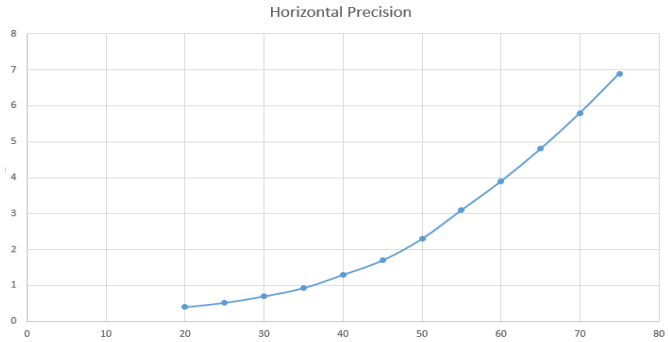


Figure 10B: Graph showing precision in x-axis

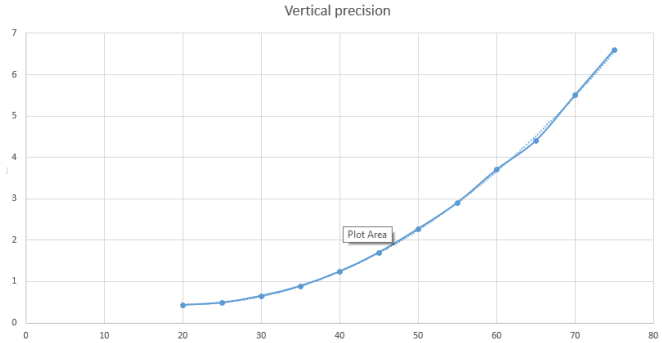


Figure 10C: Graph showing precision in y-axis

5.3 VALIDATION

The Experiment was to determine the tracking accuracy and precision in horizontal and vertical plane and the result of the experiments were as given above. In the experiment, the trackable was mounted on a movable turn table and was kept stationary for the first part. The camera was then mounted on a stage which can be moved in x, y directions using a screw gauge of .02mm least count. This stage is kept stationary and the camera is moved until we detect two units change in the turtle sim. Measurements were carried at different Z values from 20cm to 75cm (+/- .1 cm) in steps of 5 cm. An interesting observation was, we thought the change in precision would be linear with time, but it was more of polynomial in nature. We found out that this is due to the fixed resolution of the camera which is unable to register the changes at further distances in the image. And since this is an image based tracking system, the sweet spot would be between 20cm-60cm from the camera.

The second part of the experiment consisted of finding out how much angle the trackable can be rotated to still be tracked. In this experiment, we kept the camera stationary and rotated the turn table carrying the trackable, The results were expected as the camera was able to detect the trackable in a 270^{o} field of view as the emitting angle of the LED was about 135^{o}. Occlusion was preventing from detecting the LEDs at further angles.

Achieving sub-millimeter accuracy was the challenge and we solved it at a distance of 20-40cms from the camera. A higher resolution camera will give better results since this is completely an image based tracking system. But our system still gives better performance than systems like April Tags and other image based tracking systems.

6 CONCLUSION

We present a robust, accurate tracking system for Biomedical Equipment in VR. Since it is implemented in ROS, it can be easily integrated with other projects. The system can also be implemented across a network so that the pose data can be read and manipulated by a different system furthering the integration into Virtual reality. The Algorithm is fast and reliable and is very easy to implement. There is a need for a lot of general purpose libraries and repositories for virtual reality development.

There are a few **Limitations** to the solution we implemented. The first one is regarding spatial resolution. Even though the tracking system gave really good accuracy (sub millimeter) in the near field (20cm-40cm) it was giving only resolutions of the level cm in the far field (above 1m). A higher resolution camera might fix the problem but may increase the Temporal resolution since there'll be much more processing involved. The Temporal resolution for the current system is good, with the system taking nearly 1-2 seconds for the correspondence search and then as little as 10-20ms for keeping the object tracked. But the constant velocity model is not able to track the objects in swift and continuous motion as, one swift movement would break the constant velocity model and the system will have to do the correspondence search again. One possible solution is to use a camera of high frame rate. Our system can run cameras of frame rate up to 60 fps in real time, anymore would increase the temporal resolution further.

Future work is to implement segmentation to track multiple objects at the same time. A solution based on blinking frequency (implemented by Oculus) is proposed and we hope to further the research into the same.

The implementation will also require us to integrate our tracking system with the ACLS project that is happening at the Haptics lab. We hope to implement this system to some of the generally used Biomedical equipment and test their performances.

7 REFERENCES

[1] Matthias Faessler, Elias Mueggler, Karl Schwabe and Davide Scaramuzza "Monocular Pose estimation System"

[2] A. Breitenmoser, L. Kneip, and R. Siegwart, "A Monocular Visionbased System for 6D Relative Robot Localization"

[3] L. Kneip, D. Scaramuzza, and R. Siegwart, "A Novel Parametrization of the Perspective-Three-Point Problem for a Direct Compuation of Absolute Camera Position and Orientation"

[4] M. Quigley, K. Conley, B. P. Gerkey, J. Faust, T. Foote, J. Leibs, R. Wheeler, and A. Y. Ng, "ROS: an open-source Robot Operating System,"

[5] G. Bradski, "The OpenCV Library,"

[6] A. Censi, J. Strubel, C. Brandli, T. Delbruck, and D. Scaramuzza, "Low-latency localization by Active LED Markers tracking using a Dynamic Vision Sensor,"

[7] M. A. Fischler and R. C. Bolles, "Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography,"

[8] T. Pintaric and H. Kaufmann, "A Rigid-Body Target Design Methodology for Optical Pose-Tracking Systems,"

[9] R. Szeliski, "Computer Vision: Algorithms and Applications"