Monocular Vision based Pose Estimation of Rendezvous and Docking for Autonomous Assembly of a Reconfigurable Space Telescope (AAReST)

Jinseong Lee

Master of Science in Space Engineering
from the
University of Surrey

Department of Electronic Engineering
Faculty of Engineering and Physical Sciences
University of Surrey
Guildford, Surrey, GU2 7XH, UK

August 2016
Supervised by: Professor. Craig Underwood
©Jinseong Lee 2016
DECLARATION OF ORIGINALITY

I confirm that the project dissertation I am submitting is entirely my own work and that any material used from other sources has been clearly identified and properly acknowledged and referenced. In submitting this final version of my report to the JISC anti-plagiarism software resource, I confirm that my work does not contravene the university regulations on plagiarism as described in the Student Handbook. In so doing I also acknowledge that I may be held to account for any particular instances of uncited work detected by the JISC anti-plagiarism software, or as may be found by the project examiner or project organiser. I also understand that if an allegation of plagiarism is upheld via an Academic Misconduct Hearing, then I may forfeit any credit for this module or a more severe penalty may be agreed.

Monocular Vision based Pose Estimation of Rendezvous and Docking for Autonomous Assembly of a Reconfigurable Space Telescope

Jinseong Lee

Date: 30/08/2016

Supervisor’s name: Professor. Craig Underwood
Acknowledgements

I should accept that I could not complete this project without many people around me. First of all, I would like to gratitude to my supervisor Prof. Craig Underwood for his sincere advice and guidance on this project. For Prof. Yang Gao and Dr. Fernando’s important and invaluable comments, I also feel thankful for their feedback on how to improve the result of this research. I cannot help but giving thanks to my wife, Hanbyul who dedicated her all efforts to support our children and me while staying in the UK. She assisted me to have my own time in the library even on Saturday and Sunday. In addition, I am thankful for my three children, Jidam, Yudam and Sodam. They have grown up in good health and played all together although I could not spare much time for them at night and the weekend.

Additionally, I should give my much thanks to the three Asian friends (Alec, Paresh, Humza) who gathered all together and helped each other with our best. I also have much debt to my friends (Sam, Pete, Pete, Toby, Henry, AJ, James, Andy, Dez, Harry, Lois) in MSc Space Engineering for their help and advice. Additionally, our faculty Dr. Brown, Prof. Xiao, Dr. Lofty, Mrs. Karen in our EE department and Dr. Chris Bridges guided me with much advice. I cannot forget giving my special thanks to all my people in local community (Jihoon, Hansung, Peter, Jo, Sue, Jeffrey, Kwangyoon, Esther, Simon, Licardo, Corin, June’s family, prayers in Emmanuel who gave me much supports and comfort in many ways to live and stay in Surrey). For the last, I give all my thanks to our God who always gives much bless around me.
. WORD COUNT

Number of Pages: 73
Number of Words: 18628
ABSTRACT

To assemble a space-based telescope requires autonomous rendezvous and docking to maintain high precision of alignment between free flying structures in orbit. The AAReST (Autonomous Assembly of a Reconfigurable Space Telescope) mission has been proposed to provide the machine vision system’s feasibility on autonomous navigation using nanosatellites with low cost and risk. For rendezvous and docking between two identical 3U mirrorsats, machine vision algorithms are implemented to identify the relative pose estimation between two satellites. In order to demonstrate its functionality, this project demonstrates relative pose estimation using a commercial webcam and presents its robust operation under different distance, and angle view, and light intensity.

For a machine vision system, a Raspberry Pi B board was first chosen for easy accessible processor and a commercial camera module is selected. Later, the development environment was transferred to a PC and could demonstrate detection and estimation using a webcam. From detected image on LED glyph, the monocular vision determines pose estimation and the range to a target satellite. To calibrate a vision sensor, a special LED pattern is designed and rectangular checker board was utilized. The pose estimation algorithm is implemented, utilizing the OpenCV library to give out rotation and translation information. To evaluate its performance, the vision system was tested in various distance and angle views and its performance were checked in different geometry and illumination.

This paper presents how computer vision algorithms using a monocular camera works for autonomous pose estimation and it finally evaluates the accuracy in several configurations. It showed the high accuracy in translation up to a few mm distance, and precise rotation angle with less than a half-degree error. It further highlights future improvement for better detection and algorithms to be applicable to space environment.
TABLE OF CONTENTS

Declaration of originality ........................................................................................................ii
Abstract .................................................................................................................................. iii
Table of Contents ...................................................................................................................... vi
List of Figures .......................................................................................................................... viii

1 Introduction .......................................................................................................................... 1
  1.1 Background and Context ................................................................................................. 1
  1.2 AAReST Mission ............................................................................................................. 1
  1.3 Assumption ..................................................................................................................... 4
  1.4 Objectives ...................................................................................................................... 4
  1.5 Scope .............................................................................................................................. 5
  1.6 Achievement ................................................................................................................ 5
  1.7 Overview of Dissertation .............................................................................................. 6

2 State of the art ....................................................................................................................... 7
  2.1 On Orbit Servicing ........................................................................................................ 7
  2.2 Reconfigurable Space Telescope .................................................................................... 7
  2.3 Autonomous Rendezvous and Docking ......................................................................... 8
  2.4 Optical Sensors ............................................................................................................. 8
  2.5 Pose estimation .............................................................................................................. 9
  2.6 Feature extraction algorithms ....................................................................................... 10
  2.7 Glyph pattern ............................................................................................................... 11

3 Theory .................................................................................................................................. 12
  3.1 LED pattern .................................................................................................................. 12
  3.2 Coordinate transformation ............................................................................................. 12
  3.3 Camera calibration ......................................................................................................... 14
  3.4 Rotation angle ............................................................................................................... 16
  3.5 Thresholding ................................................................................................................ 17
  3.6 Blob detection ............................................................................................................... 14
  3.7 Centroid algorithm ....................................................................................................... 19
  3.8 Template matching ...................................................................................................... 19
  3.9 Pose estimation algorithm ............................................................................................ 20

4 Technical Section ................................................................................................................. 22
  4.1 Block diagram .............................................................................................................. 22
  4.2 Hardware and Software ............................................................................................... 23
    4.2.1 Hardware .............................................................................................................. 23

Jinseong Lee MSc dissertation
LIST OF FIGURES

Figure 1. Reconfigurable satellites of the AAReST Mission .............................................. 3
Figure 2. CoreSat and MirrorSat (Front and Back) ................................................................. 9
Figure 3. ISM structure (left), original ‘L’ shaped marker (middle and right) ....................... 10
Figure 4. The wavelength(left) and intensity(right) of an IR LED ........................................ 12
Figure 5. Coordinate transformation and projection on the image plane .............................. 12
Figure 6. Image mapping on the image plane at focal length and its resemblance .................. 13
Figure 7. Internal and external parameters by camera calibration ....................................... 14
Figure 8. Image of a square with barrel distortion(a), pincushion distortion (b) ................. 15
Figure 9. Image projection (left) and flow chart of the POSIT (right) ................................. 20
Figure 10. Block diagram for pose estimation using Raspberry Pi ..................................... 22
Figure 11. Raspberry Pi 3 and Pi cam connection ................................................................. 23
Figure 12. Cubesat size ........................................................................................................ 24
Figure 13. LED pattern candidates for a satellite ............................................................... 24
Figure 14. Circuit design (left) and Sample design on a PCB board (right) ....................... 26
Figure 15. Sample pattern using five LEDs (left) and Calculation FoV(right) .................... 27
Figure 16. Calculation pixel number at full-size image ..................................................... 27
Figure 17. Calculation pixel number of LED at 1 m away .................................................. 27
Figure 18. Flow chart ........................................................................................................... 27
Figure 19. Image capture test .............................................................................................. 27
Figure 20. Edge detection test using OpenCV library ........................................................ 30
Figure 21. Webcam calibration ........................................................................................... 30
Figure 22. Threshold test .................................................................................................... 31
Figure 23. Edge detection test ........................................................................................... 31
Figure 24. Blob detection test ........................................................................................... 32
Figure 25. Centroid test and position of centers ............................................................... 32
Figure 26. LED pattern matching test .............................................................................. 32
Figure 27. Rotation invariant algorithm to find the index out of 5 LEDs ....................... 34
Figure 28. Point mapping between 3D world and 2D image coordinate ......................... 36
Figure 29. Measuring the ground truth using a ruler and a protractor ......................... 38
Figure 30. Translation measurement along Z-axis ............................................................. 39
Figure 31. More image points using interpolation ............................................................. 39
Figure 32. Translation along Z-axis .................................................................................. 40
Figure 33. Accuracy (left) and error ratio (right) along Z-axis and frame rates .............. 40
Figure 34. Translation along X-axis .................................................................................. 41
Figure 35. Accuracy (left) and error ratio (right) along X-axis .................................................. 41
Figure 36. Translation along Y-axis ............................................................................................... 41
Figure 37. Accuracy along Y-axis, from top to bottom ................................................................. 42
Figure 38. Rotation angle from 0 to 90 degree along Z-axis ....................................................... 42
Figure 39. Rotation accuracy along Z-axis ..................................................................................... 43
Figure 40. Rotation accuracy along X-axis, from left to right ...................................................... 43
Figure 41. Measurement along Y-axis, from top to bottom .......................................................... 44
Figure 42. Measurement in different light intensity ................................................................. 44
Figure 43. Rotation and translation error along Z-axis in different light intensity ....................... 45

Table 1. Threshold Type ........................................................................................................... 17
Table 2. Blob detection parameters ............................................................................................ 18
Table 3. Raspberry Pi 3 specification .......................................................................................... 23
Table 4. Raspberry Pi Camera specification ................................................................................ 24
Table 5. LED pattern power consumption .................................................................................. 26
Table 6. OpenCV library functions ............................................................................................. 29
Table 7. Camera Calibration parameters ..................................................................................... 30
Table 8. Camera distortion parameters ....................................................................................... 30
Table 9. Pose estimation function .............................................................................................. 35
Table 10. Development environment for computation ............................................................... 37
Table 11. Camera parameters ..................................................................................................... 37
Table 12. Blob detection parameters .......................................................................................... 37
1 INTRODUCTION

This section will describe the background of the AAReST missions and deal with the trend of technical elements in relevant fields.

1.1 Background and Context

As the size of space telescope increase, launching large payload from the ground has been demanded. However, it is technically challenging, thus new technology to assemble in space has been required. With small and low cost, and customized performance it can be achievable to build large structure in space, using modular segments for science missions or commercial purposes.

To accomplish this challenging task, autonomous rendezvous and docking becomes an important element to obtain high accuracy in alignment between modular structures and to cope with time delay from ground commands. To experience and avoid risk of building large structure in space, the ARReST mission has been instead proposed to demonstrate the feasibility this technology using nano satellites and show the potential application of micro satellites in autonomous rendezvous and docking at low cost. It is expected to boost confidence in the autonomous assembly for building large space telescope which will make innovative approach in many space applications [2].

A variety of advancement on sensor based navigation has been achieved in many applications such as robotics and automations to guide unmanned systems with autonomy. Among them, machine vision is one of the powerful and a simple approach which provides the information about relative pose estimation and distance. Detecting attitude and range between satellites can be obtained using different types of sensors such as a vision sensor and range finder. A vision system simply uses algorithms on processor boards, and monocular vision is easy access among them to provide visual function. It avoids the complexity of sensor system and also reduces power consumption than other sensor systems[16][17][19][20].

1.2 AAReST Mission

The final goal of the AAReST mission is to assemble telescopes with large aperture more than 20 m in space using modular segments, thus enabling building a large-scale of telescope such as James Webb Space Telescope. Compared to other projects, this mission is equipped with deformable mirror technology and docking system by electro magnets[5][6][7]. After starting in 2009, many aspects of technologies including design of optical parts and autonomous navigation parts have been covered. The bus design of this project is based on the previous mission such as SNAP and STRAND[8], and the bus was mainly from them. It is also based on 3U cubesat structure, and has a capability of full 3-axis attitude control system.
This mission involves two “nano satellite” in 3U cubesats (mirrorsats) size and carry a Deformable Mirror Payload (DMP) and a central “15U” microsatellite (a coresat). Overall size of this package with a stowed volume is 0.4 m by 0.4 m by 0.6 m. Using relocating the position between a coresat and mirrorsats, it carries out maneuvering for redock and undock motion [5]. By means of autonomous optical sensing and electromagnetic force, it forms different configurations of multi-aperture telescope. It is achievable through constructing different array between both mirrorsats and between a coresat and mirrorsats using docking and rendezvous. The overall reconfiguration of a satellites is illustrated in Figure 1.

![Figure 1. Reconfigurable satellites of the AAReST Mission](image)

For Attitude determination system, it consists of 3-axis magnetometer, a Sun Sensor and a GPS receiver and it is controlled by three-axis magnet torque and three reaction wheels. To reconfigure satellite, the main part of this paper is focused on machine vision for optical navigation which enables the satellites to identify the distance and pose of target satellites and docking. Once autonomous pose estimation is achieved, electromagnetic system guide the system for docking within a few meters[5][6][7].

In unknown environment, it is much easier if known features and models such as LED glyph can be used for identify its pose and indeed it alleviates the limitations of pose estimation [48]. By detecting natural features and matching them with model feature, it is needed only to install any artificial pattern on a target object and simply match the pattern based on geometry and structure. There are also ambiguity which leads to false information under illumination, viewing angles and distances. To demonstrate reliable performance, a vision sensor first should be calibrated and multiple images need to be tested in various angles views between satellites. As there are also several factors which influences on light intensities coming from different sources such as Sun, Earth, and satellites in orbit, additional tests on IR LED glyph under sunlight with various light intensities are required to prove its reliable operation.

For this AAReST mission, the LED glyph is attached on a coresat and mirrorsats and its LED pat-
tern on each sat is used for pose estimation in order to make successful rendezvous and docking. For pose estimation, the camera on the mirrorsats detects the LED pattern on a coresat, and determine its relative pose and distance which is needed for docking control.

1.3 Assumption

To demonstrate the performance, the test environment can be simple set by using the LED pattern and a single camera. Measuring the distance along Z-axis in the camera coordinate is regarded to have parallel to the surface of LED face. For bench test environment, the approach on this work is to design LED pattern on a core sat and place a camera which belongs to a chaser satellite. It aims to detect pattern using a single camera to capture image from monocular vision and does not need any other sensors for pose estimation. For this project, it is based on several assumptions as:

1. The distance between two satellites are in close proximity less than 1~2 meter. It is because the attitude control when the angle difference is larger than 20 degree is achieved by attitude control system of the satellite itself. It is required to show the performance in rotation less than 20 degree.

2. Prior to doing test along three axis rotation, the experiment on air bearing is to test only one rotation along yaw angle, but is also first tested if the algorithms works in free space.

3. Pose estimation based on vision algorithm is only applied within close proximity due to the limitation on the properties of the image projection.

For this mission, we first utilize a RPi camera based on a board to detect IR LED glyph, It is to demonstrate the capability using an off-the–shelf item and suggests its reliable performance for space environment. However, more complicated algorithm to calculate attitude and distance required faster computation. Although Raspberry pi verified simple image processing of the OpenCV, more complicated pose estimation required higher processing power. Hence, development environment in PC was considered more suitable in the aspects of computing power, saving time for debugging, so PC experiment was preferred for later parts of the experiments [21][22]. To take the measurement under different angle view and distance, and also test under various light intensities to simulate reliable operation under sun light in space environment.

1.4 Objectives

There are tasks such as designing a LED pattern, sensor calibration, algorithms implementation, and tests from different conditions of angle views and light intensities. The purpose of this project
is to obtain implement vision systems for accurately determining 6 DOF parameters in pose estimation and distance between two micro satellites. The key objectives are as follows.

- Design rendezvous vision sensor at 640x480@5 frames/sec images
- Test the accuracy with error less than 6% for rotation and translation
- Test the performance under various light condition

1.5 Scope

There are mainly two tasks. Firstly, it is to acquire the accurate relative pose and distance from different angle view, distance using IR LED pattern. Secondly, it is to verify its performance in different light intensities. The overall detail tasks are described below.

- Pose Estimation
  - Development environment setup
    - Purchase a commercial camera and design LED glyph pattern in circuit
    - Compile OpenCV library in Raspberry Pi and a PC
  - Grab process a cam image and test OpenCV
  - Sensor calibration using checkerboard
  - Test image processing using a sample glyph
  - Implement pose estimation and filter algorithm using C++ or C
  - Test its performance in free 3D space
  - Extract translation and rotation in various test conditions
    - Different distance and angle view
    - Test in different light intensity or interference

1.6 Achievement

Several important tasks have been carried out as below and advanced steps will be conducted based on these.

- Built simple LED pattern to be tested
- Extracted edge of features in the image
- Image and video capture using mono camera
- Image processing(threshold, blob detection, centroid, pattern matching, pose estimation)
- Implemented the algorithm using geometry properties and recognized the pattern
- Apply pose estimation algorithm on Raspberry Pi board and a PC

1.7 Overview of Dissertation

This paper presents a new approach for pose estimation using commercial monocular vision system (RPi camera and a webcam) in order to achieve autonomous reconfigurable of a satellite. It is to demonstrate how they work reliably under different distance, angle views and light intensity to extract accurate pose and distance between satellites. Chapter 2 provides the fundamental background and motivation of this project. Chapter 3 presents the theory behind this experiment and how the principle of image processing is applied in the implementation. Every component of these theories played an important role to make it produce the proper value based on the principle. Chapter 4 addresses the method in the aspects of hardware and software and demonstrate its intermediate outcome which lead to final results. The Chapter 5 describes the experiment procedure and important parameters to set up the environment and other aspect to be considered. Based on those values, final results were obtained. The Chapter 6 provide the conclusion and present the evaluation on outcome and any problems or limitation. To improve the works, additional comments was added for further works which needs to done.
2 STATE OF THE ART

This review will introduce orbit servicing related to the background of the AAReST missions and deal with the trend of technical elements in relevant fields.

2.1 On Orbit Servicing

Nowadays, on-orbit servicing is largely carried out manually or by an astronaut in low earth orbit. However, manned space flight requires high cost, and should not be ignored of human safety. However, it is almost infeasible to carry out those missions in MEO or GEO human cannot reach, in which it is even worse with communication delay and have limitation in bandwidth between ground and spacecraft. Accordingly, a viable solution is to provide autonomy on orbit servicing and it entails the capability of estimation and tracking of pose of the target object [1] [2][4].

As more complicated space missions have been required for scientific observation and earth observation, communication, there has been increasing demand to extend their lifetime of spacecraft and maintain the systems periodically. The importance of on-orbit servicing lies on extension life-time of various space platforms and help space activities for manned or unmanned missions. The key benefits of on orbit servicing are refuelling and repair, and refurbishment capabilities [1]. These benefits increase overall mission’s reliability with economic value. In addition, it provides an extended functionality and utility without the need to launch again. To achieve successful orbit servicing, it requires higher level of technology and still requires much challenge in rendezvous and docking between space systems.

The need to utilize the space systems already in space and to construct large structures for many space ventures, and to provide systems that are reliable and cost-effective is increasing. In addition, abandoned space systems poses more hazards to newer constellation program, and occupy orbital space which can be used for other purposes. In the early 1980s, the NASA also realized the importance of on-orbit servicing to protect many space assets in orbit. The repairing mission such as Hubble Space Telescope and the International Space Station have shown that on orbit servicing can create the new architecture which is essential to challenge the next frontiers in space.

2.2 Reconfigurable Space Telescope

Reconfigurable assembly technology has much potential to provide low cost observation to many applications including government and commercial, and can open a new era to utilize modular, flexible structure and enables to launch meter-class telescopes. Above all, it is highly reasonable to allow on-orbit maintenance, aperture and instrument reconfiguration and re-deployment.

There have been several researches and are more with ongoing projects. The CAST (configurable Aperture Space Telescope) project of NASA is to evaluate an optical and mechanical concept for a
novel implementation of a segmented telescope based on modular interconnected small satellites [2][4]. Additionally, SPOT(Spherical Primary Optical Telescope) is to develop a robust architecture to reduce cost of large-aperture and OpTIIX (Optical Testbed and Integration on ISS eXperiment) uses light weight mirrors for the purpose of on-orbit assembly[4][5]. These technologies are attractive to many commercial space sectors such as Google, Skybox, planet labs etc.

2.3 Autonomous Rendezvous and Docking

Autonomous Rendezvous and Docking (ARD) technology is indispensable to assemble large space structures using lightweight, modular segments. The NASA has recently considered ARD as a key technology for future space missions. There have been several demonstration flight such as ETS, XSS, PRISMA. Not all of them were successful like the Demonstration of Autonomous Rendezvous Technology (DART) and Orbital Mission left significant advances for ARD. Those missions address that high-level technical difficulties still remain to be challenged [9]. In addition, last March in 2016, the satellite, Hitomi satellite from JAXA, lost its control in space and has been still tumbling. However, it is unknown how its motion is like, so it would be helpful to identify tumbling satellite’s motion by a servicing satellite in order to recover its unstable motion from anomaly state. To do that, optical sensor in proximity will be able to give an aid for autonomous rendezvous and pose estimation, not just remotely check its status through telemetry on the ground station[10][38]

2.4 Optical Sensors

Various types of optical sensors were used for satellites using active and passive sensors. For active sensors such as RF radar, LIDAR, it provides more accuracy in measurements while having more complexity, size, and power consumption [10][11][12]. On the other hand, vision sensor as passive sensors are preferred for its simplicity and low power which uses multiple-view imaging and feature mappings[14][15].

There have been a number of autonomous rendezvous and docking on uncooperative satellites in a way of single or multiple sensors. Depending on the distance, it is necessary to choose different methods to deal with different information. Combining LIDAR and vision camera more range of measurement from a longer distance and also higher accuracy, however, it contained more complexity to deal with different data sources from each sensor and leas to high cost[12][14][15]. Similarly, LIDAR and Model based vision has been attempted because it offers accurate measurements, but still has a problem in power consumption compared to passive camera sensors [16]. Accordingly, monocular vision system is relatively simple in the architecture, however, more sophisticated algorithms should be supported compared to stereo vision [16][17][19][20][57].
As illustrated in Figure 2, a coresat is located at the center, and mirrorsats are moving around and dock or detach from it. When doing rendezvous, it used IR LED pattern to recognize its pose and the pattern should be drawn considering the size of face as many numbers of the faces as there may orient [7]. To design glyph, LED patterns are designed in different shapes for each face of a satellite to identify which face of a coresat and mirrorsats are looking at [5][6]. By simply looking at the pattern, monocular vision system can detect the image, and calculate its relative pose estimation[57].

2.5 Pose Estimation

There are a large number of different algorithms already exist to find pose by image processing from stereo or mono camera. These vision algorithms are commonly used for many applications in robotics and augmented reality. Several algorithms also work for both vision and ranging sensors to match their data and determine final pose for autonomous rendezvous. The vision algorithms can be processed and implemented using integrated development environment and compiled in a microprocessor. For pose estimation, there are several approach to find the 3D pose such as finding features, perspective projection, orthographic projection, weak projection. The Perspective-n-Point(PnP) is the most popular algorithm, however, P3P which is the most commonly used generates 4 possible solutions, so it is not enough. For having an unique solution, there must be 4 points like P4P and also the Coplanar POSIT algorithms is widely used as it utilizes optimization, weak perspective projection. In addition, this does not require initial pose estimate, nor expensive in its iteration loop[63][57].

The methods for pose estimation are categorized to three methods; model-free, model-based and non-model based techniques[23]. Model-Free requires no priori model for estimation. Non –model-based techniques do not assume a priori information of their target objects but can be obtained by
measurement process. This method can be used to solve for camera motion provided image velocity or feature correspondence measurements under motion constraints. One drawback is it often fails in tracking for target feature points due to perspective projection. On the other hand, model-based techniques have a priori knowledge of the object’s pose and motion to be estimated and it contains shape, structure, transmittance, reflectance or visual attributes. It is less sensitive to the partial occlusions and loss of data. For pose estimation relying on a 3D model, there are several algorithms such as Direct Linear Transform (DLT) method, POSIT, homography, non-linear estimation. All these uses the characteristics of the linear system. The DLT requires at least 6 non-coplanar points, however, the others works with at least 4 coplanar points.

This method can be divided into a few stages to process. First, a priori model is computed to obtain initial pose. After then, a coarse initial pose is estimated [13]. Here, we are to adopt the most common algorithm, POSIT, as a model-based technique, and it uses the given information of the geometry between LED glyph to estimate 3D pose.

There are also previous heritage from for reference [60] which uses ‘L’ shape marker on a target of the Intelligent Self-powered Module (ISM). By the projection, the rotation angle of roll, pitch, yaw can be calculated using the length between the line of AC or BC depending on the angle view. However, this method has disadvantage in detection because its sensor does not emit the light, making it difficult to capture its image when in darkness such as eclipse. Even though it is under sunlight, it is still very vulnerable to solar lighting condition.

Another examples offers possible approach even to the coplanar points which is also applicable markers on the surface of satellites [39][60][61]. This method adopts the optimization algorithms of Levenberg-Marquardt which uses the steepest descent method. For this paper, it's picks up the coplanar points and also uses this optimization in a similar way.
2.6 Feature extraction algorithms

Among vision system, stereo vision is most commonly used, however, it is not reliable when the distance is in far range[12]. There are other algorithms to process features for a monocular, stereo or sensor fusion. The ICP (Iterative Closes Point) algorithms uses the closest points to align a target model with the model in the expected pose and then rejects outliers until criteria is met[12][13][41]. Alternatively, Bounded Hough Transform (BHT) searches for a six dimensional solution around the last pose and estimate the pose in sparse data although noise and outliers are present [27]. Structure from motion (SFM) is utilized in both stereo and mono camera system, and it produces very reliable correspondences in an image. It is possible to estimate the camera motion when it is already known for camera calibration, however, its process becomes unstable when there is little translation of the features between views. There are an algorithm which is of non-deterministic such as RANSAC(Random Sample Consensus ) and it uses a stochastic approach when some erroneous stereo and temporal feature is associated in tracking and features[11][21][31][47]. It selects random items and iterates until the data items fit the model (inlier) within a given tolerance, thus produces a reasonable result within a certain probability [48]. If outliers are too much ratio more than 60 %, it is hard to estimate a robust camera pose. The Point Cloud techniques build a point cloud of the target using multiple images and then match the point cloud model to its shape. This method is easily accessible by utilizing online library and the pose can then be accurately identified using three dimensional feature correspondences.

2.7 Glyph pattern

The intensity of light changes drastically between ellipse and sunlight In space environment. To increase the reliability of detecting the satellite pose, it is desirable to utilize a target marker as another solution, other than the LED itself. Typically, glyphs are described by arranging several square grids and each cell can be black or white color. Its pattern can be random, however, in most cases, the first and last column contains black border. When the LED detection fails, it is also possible to detect white and black pattern instead of LED, so it makes more robust algorithm for pose estimation. For this project, I placed the glyph as asymmetric pattern to distinguish its pose in any orientation, which is made of white color as a background of LED and the remaining in black. It is then detectable using threshold or blobs and calculate the center of each cells for pose estimation.
3 THEORY

3.1 LED pattern

When rendezvous mission is to be done during night or within eclipse area, LED glyph on a satellite should be seen easily and detectable. To detect the markers of a satellite, an infrared LED is used to detect regardless of light intensity. The central wavelength of the chosen LED is at 870nm which is above red light, so it is not visible when it is turned on[52]. To test the brightness of a LED and use its light for pose estimation, the angle of half-intensity was considered to be more than 30 degree. In Figure 4, the angles detected ranges up to 90 degree, theoretically illuminating on half of sphere.

![Figure 4. The wavelength(left) and intensity(right) of an IR LED](image)

3.2 Coordinate transformation

The transformation between world coordinate and camera coordinate involves rotation and translation and its relationship is expressed as a combination of two matrices as below. Here, \( \nu_i \) are the target points and R is rotation, and \( t \) is translation vector from the camera coordinate to the object respectively.

![Figure 5. Coordinate transformation](image)
The relationship is simply written as

\[ V_i \propto R P_i + t \]

and its expanded form is

\[
\begin{bmatrix}
X' \\
Y' \\
Z'
\end{bmatrix} =
\begin{bmatrix}
X \\
Y \\
Z
\end{bmatrix} +
\begin{bmatrix}
t_x \\
t_y \\
t_z
\end{bmatrix}
\]

Combined matrix form of 3 by 3 rotation and 3 by 1 translation is 4 by 4 matrix and it can be again expressed as

\[
\begin{bmatrix}
X' \\
Y' \\
Z'
\end{bmatrix} =
\begin{bmatrix}
r_{11} & r_{12} & r_{13} & t_x \\
r_{21} & r_{22} & r_{23} & t_y \\
r_{31} & r_{32} & r_{33} & t_z \\
0 & 0 & 0 & 1
\end{bmatrix}
\begin{bmatrix}
X \\
Y \\
Z
\end{bmatrix}
\]

Euler’s rotation theorem defines that any combination of rotations can be expressed as a rotation w.r.t Euler axis with specific rotation. Combining this principle and coordinate transformation, OpenCV provides the function calculates the rotation and translation vector using images. The point, \( P_c \) in the camera coordinate and the point, \( P_w \) in the world coordinate has a relationship as

\[
P_c = R P_w + T
\]

\[
P_w = R^{-1}(P_c - T)
\]

\[
P_w = R^{-1}(0 - T) = -R^{-1}T
\]

To translate the point into world coordinate, \( P_w = R^{-1}(P_c - T) \) and if the center point on the image plane is set to zero, origin point along optical axis is calculated as To define the coordinate, it is conventionally accepted that X-axis is to right, Y-axis is downward, and Z-axis is facing outward toward target. Assuming a camera is a pin hole camera, its relationship between target object and the image plane is explained with a simple concept as Figure 6

Figure 6. Image mapping on the image plane at focal length and its resemblance
Therefore, the object point \( P(X, Y, Z) \) corresponds to the point on the normalized image plane \( P' \left( \frac{X}{Z}, \frac{Y}{Z}, 1 \right) \) at the distance 1. However, the physical camera makes the image projected on the image plane spaced apart by the focal length \( f \), which is \( P'' \left( f \frac{X}{Z}, f \frac{Y}{Z}, f \right) \). Its relationship is easily calculated by resemblance triangle. However, camera has different focal length \( f_x, f_y \) in along X and Y axis, and it is mapped to physical distance at \( \left( f_x \frac{X}{Z}, f_y \frac{Y}{Z} \right) \). Considering the coordinate of the center point \( C(C_x, C_y) \) as origin, the point in the world coordinate is mapped to the pixels on the image plane at \( P'' \left( x = f_x \frac{X}{Z} + C_x, \ y = f_y \frac{Y}{Z} + C_y \right) \).

### 3.3 Camera calibration

A camera has its own intrinsic parameters and it is needed to calibrate for accurate computer vision.

![Camera calibration diagram](image)

**Figure 7. Internal and external parameters by camera calibration**

There are two types of the parameters; intrinsic and extrinsic [48]. The intrinsic parameters correspond to the relationship between image plane and camera coordinate, and include internal values such as skew, focal length, an image format, and principal point. Whereas, extrinsic parameters are commonly referred to as the camera pose and which relates 3D point between a target object and a camera frame and has the information about rotation and translation. These values are changeable and are also dependent on the orientation [40]. To find the values for those parameters, a checker board already known its exact scale is used.

Intrinsic parameters affect the distortion on the image. There are two terms in distortion; Radial distortion as a result of lens shape, and tangential distortion from the assembly process of the camera [59].
If the point \((x_{n,u},y_{n,u})\) in the world coordinate is project to the image plane without any distortion,

\[
\begin{bmatrix}
x_{n,u} \\
y_{n,u}
\end{bmatrix} = \begin{bmatrix} X_c/Z_c \\ Y_c/Z_c \end{bmatrix}
\]

However, nonlinearity on the lens project the point into different 2D point \((x_{n,d},y_{n,d})\) on the normalize image plane and it is expressed by radial distortion and tangential distortion terms. Here, the coefficient \(k_1, k_2, k_3\) of the first term on the right side affect radial distortion, and the other \(p_1, p_2\) in the second term makes tangential distortion. Considering all these parameters, distortion parameters is presented with 5 columns

\[
\text{Distortion coeff} = [k_1, k_2, k_3, p_1, p_2]
\]

Its distorted point can be described as

\[
\begin{bmatrix}
x_{n,d} \\
y_{n,d}
\end{bmatrix} = (1 + k_1 r_u^2 + k_2 r_u^4 + k_3 r_u^6) \begin{bmatrix} X_c/Z_c \\ Y_c/Z_c \end{bmatrix} + \begin{bmatrix} 2 p_1 x_{n,u} y_{n,u} + p_2 (r_u^2 + 2 x_{n,u}^2) \\ p_1 (r_u^2 + 2 y_{n,u}^2) + 2 p_2 (x_{n,u} y_{n,u}) \end{bmatrix}.
\]

It is usually omitted to set the value of \(k_3, p_1, p_2\) as zero. Considering skew between X and Y axis and focal length, the point in the world coordinate is then projected to the normalized image plane, it is described as

\[
\begin{bmatrix}
x_{p,d} \\
y_{p,d}
\end{bmatrix} = \begin{bmatrix} f_x & c_x & x_{n,d} \\ 0 & f_x & y_{n,d} \\ 0 & 0 & 1
\end{bmatrix}
\]

If the point in the world coordinate is transformed to the 2D image plane, it is then combined with the rotation and translation matrix,\(3\) by \(4\) as previously discussed. In the end, its overall relationship can be described as \(K, R, t\) and here \(K\) is camera intrinsic matrix to transform from normalize image to the pixel on the image, \(R\) is rotation and \(T\) is translation matrix. By using trigonometry and linear relationship between a target object in 3D coordinate and an 2D image plane, its inner and outer parameters such as “\(K, R\) and \(t\)” are illustrated as
By decomposing the intrinsic matrix, it is then divided by the matrix, including skew matrix and focal length respectively.

\[
\begin{bmatrix}
X \\
Y \\
Z \\
1
\end{bmatrix} = K[R|t] = K[R] \begin{bmatrix}
x \\
y \\
x
\end{bmatrix}
\]

The result of camera calibration is affected by many factors such as auto focusing, however, it is sometime still different even for the same camera. The reasons are

- Calibration is based on a pinhole camera, but a real camera may not be the one
- Lens distortion model is approximated from high order polynomial function
- Calibration looks for the best solution after iterative process with optimization, which may lead to different solution at every time.

### 3.4 Rotation angle

Any arbitrary rotation can be described by 3 by 3 matrix and Rodrigues can express this rotation using 4 parameters such as 3 is for rotation vector and the other one is rotation angle. For example, if the point P is rotated about vector v with

The angle can also be obtained the relationship which is called Rodrigues rotation which uses only 3 component of vector and rotation angle. Rodrigues theorem defines the Euler vector \( v = [V_x \ V_y \ V_z] \) and Euler angle, theta for any rotation matrix as a form of

\[
R = \cos \theta \ I_{3 \times 3} + \sin \theta \begin{bmatrix}
0 & -V_z & V_y \\
V_z & 0 & -V_x \\
-V_y & V_x & 0
\end{bmatrix} + (1 - \cos \theta) v v^T
\]
Its vector and angle can be described as

$$\theta = \arccos \theta \left( \frac{\text{trace}(R) - 1}{2} \right), v = \frac{1}{2 \sin \theta} \begin{bmatrix} r_{32} - r_{23} \\ r_{13} - r_{31} \\ r_{21} - r_{12} \end{bmatrix}.$$  

To express the rotation vector into rotation angle such as roll, pitch, yaw can be obtained using 3 by 3 rotation matrix (Direction Cosine Matrix, DCM). It is easily translated to the angle using the function Rodrigues() provided by OpenCV. From the rotation matrix, euler angle can be extracted by using formula, however, there are several solutions. Therefore, it is needed to consider which sign is appropriate for the orientation and more importantly singularity should also be considered. If a roll angle is defined as the rotation angle in the counter clockwise w.r.t Z-axis, that is sign of + . It is also possible to transform the rotation matrix to the quaternion to calculate the calculation faster and avoid the singularity. To compare the angle more intuitively, conversion from the DCM to Euler angle is preferable to this experiment. For pan and tilt, each component of the rotation matrix is used to calculate the angle

$$\theta_{\text{pan}} = a \tan 2(Z_y, Z_x) - \frac{\pi}{2},$$

$$\theta_{\text{tilt}} = a \tan 2(Z_x, \sqrt{Z_x^2 + Z_y^2})$$

For roll,

$$X_w = R^{-1} X_c = [X_x, X_y, X_z]^T$$

$$X_{\text{pan}} = [\cos(\theta_{\text{pan}}), \sin(\theta_{\text{pan}}), 0]^T$$

$$\theta_{\text{roll}} = \text{sign}(X_z) \cos^{-1}(\frac{X_w \cdot X_{\text{pan}}}{\|X_w\| \|X_{\text{pan}}\|})$$

### 3.5 Thresholding

Once the computer vision transform the RGB image into grayscale, it can be divide its gray level of the object to distinguish from its surrounding. To do that, thresholding processing is necessary. There are different kinds of methods to do thresholding for image pixels. Thresholding divide the image value into several categories based on the threshold level. For example, there are binary, binary_inverse, truncation, threshold_to_zero, threshold_to_zero_to_inverse. The types are described as below.

<table>
<thead>
<tr>
<th>Description</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Value and Threshold Level</td>
<td></td>
</tr>
</tbody>
</table>
Table 1. Threshold Type

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Threshold Binary</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Threshold Binary, Inverted</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Truncate</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Threshold to Zero, Inverted</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Threshold to Zero</strong></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

3.6 Blob detection

The blobs are assumed to be circular, and the size stored as the size of the keypoints in the output vector.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Color</strong></td>
<td>0 extract dark blobs, 255 extract light blobs</td>
<td><img src="image" alt="Color Example" /></td>
</tr>
<tr>
<td><strong>Area</strong></td>
<td>between minArea &lt; &lt; maxArea</td>
<td><img src="image" alt="Area Example" /></td>
</tr>
<tr>
<td><strong>Circularity</strong></td>
<td>(4<em>pi</em>Area/(perimeter*perimeter)) between min and max</td>
<td><img src="image" alt="Circularity Example" /></td>
</tr>
<tr>
<td><strong>Inertia ratio</strong></td>
<td>Circle : 1, Ellipse 0 &lt; &lt; 1 Line: 0</td>
<td><img src="image" alt="Inertia Ratio Example" /></td>
</tr>
<tr>
<td><strong>Convexity</strong></td>
<td>Convex, Concave</td>
<td><img src="image" alt="Convexity Example" /></td>
</tr>
</tbody>
</table>

Table 2. Blob detection parameters

The OpenCV offers the function of SimpleBlobDetector which gives out the blob detection in a given image. To find blobs, this algorithm extract connected components from every binary image by find Contours. Within this contour, it finds the pixel group which is closer than the minDistBetweenBlobs parameter. There are several parameters to characterize the blobs as below. Each sub stages are described as below. Once the blobs are found, it is possible to find the radius of the size, and its contour information can be utilized for centroid algorithm.
3.7 Centroid algorithm

In order to calculate the centroid of Region of Interest or any polygon, each intensity value of every pixel within the ROI will be summed, and its error, mean and moment are then calculated to determine the position of a center point. The properties of the contour and area of the image can be used to calculate the center point of the polygon. To derive the formula to calculate the center point, it is necessary to be familiar with the moment of the gray value-function \( f(x,y) \) and its spatial moment is

\[
m_{p,q} = \iiint x^p y^q f(x,y) \, dx \, dy
\]

Here the \( p \) and \( q \) is the order of the moments and zero-moment is when \( p=q=0 \), and either of \( p \) or \( q \) is 1 for the 1\textsuperscript{st} order moments, it is given as

\[
m_{0,0} = \iiint dx \, dy \, b(x,y)
\]

\[
m_{1,0} = \iiint dx \, dy \, x \, f(x,y)
\]

\[
m_{0,1} = \iiint dx \, dy \, y \, f(x,y)
\]

For binary image after thresholding, \( f(x,y) \) is 1 for object and 0 for background. As the 0\textsuperscript{th} order of the contour moment is perimeter and 0\textsuperscript{th} order of the area moment is area, the 1\textsuperscript{st} order moment is then given as

\[
x_c = \frac{m_{1,0}}{A} = \frac{m_{1,0}}{m_{0,0}}
\]

\[
y_c = \frac{m_{0,1}}{A} = \frac{m_{0,1}}{m_{0,0}}
\]

To calculate the central moments is similarity derived from the spatial moments. The center of gravity is simply given as the division of the 1\textsuperscript{st} order spatial moment by 0\textsuperscript{th} order moment. Using this formula, the center point of the blobs can be calculated, and the OpenCV library provides the function for centroid algorithm with several different options.

3.8 Pattern matching

Pattern matching is the process to explore the position of the reference image which matches the template image. Generally, matching for the moving object in translation is relatively easy, however, it requires higher level of techniques to find the object in rotation or scaling. For template matching, it utilized its spatial information into the edge, corner, frequency transform space for template matching, and also require normalization to be less sensitive to the light intensity of the image. The
general method is to slide the template image to scan using rectangle window from the top left corner to right and to down. At each location, it compares the its resemblance OpenCV also provide this algorithms with several different options.

3.9 Pose estimation algorithm

The Pose of the 3D object is a combination of the orientation (rotation) and its position (translation). Rotation matrix® is 3 by 3 matrix and translation is 3 by 1 vector. So, the pose(P) which as 3 by 4 matrix can be described as P=[R|T]. If the at least 4 non-coplanar points of a 3D object are given, also provided with the corresponding 2D points on the image plane, and the focal length of the camera, it is possible to estimate the pose.

The POSIT algorithm uses POS (Pose from Orthography and Scaling with iteration) assumes that all the points of the object have same depth. Basically, this algorithm works for at least 4 non-coplanar points. It provides two best solutions to estimate perspective projection within 4 to 5 iterations [39] as explained in Figure 9

![Figure 9. Image projection (left) and flow chart of the POSIT (right)](image)

To implement the algorithm which calculates coplanar features algorithm, focal length and coplanar points are set, and then rotation and translational vector for the 3D model can be obtained using the relationship between the camera reference and model coordinate. It does not require an initial pose, but is very fast, easy and robust in the presence of camera calibration errors. It has advantage in that it does not need a starting pose [49][50]. However, it has limitation which doesn’t work if the objects are flat or plane. To overcome this problem, there are advanced algorithms which is Coplanar POSIT[39]. This algorithm is already implemented in AForge.NET framework. The difference between POSIT and Coplanar POSIT is that the latter provides two solutions, so it needs
to try both and select the one which best fits to the model.

To find the closest solution, this algorithm uses iterative method to find the minimum error between the measured vector and projected vector of the point on the image, using the focal length proportion. When the point is $v_i$, a target point, $R$ and $t$ is rotation and translation vector, its error is given as below. It is repeated iteratively until it reaches the tolerance with the least error to give out the optimum solution of the pose estimation in terms of camera coordinate.

$$E_{os}(\tilde{R}, \tilde{t}) = \sum_{i=1}^{n} \left\| \left( I - \frac{\tilde{v}_i \tilde{v}_i^T}{\tilde{v}_i^T \tilde{v}_i} \right) (\tilde{R}p_i + \tilde{t}) \right\|^2$$

Similarly, SovePnP() function provided by OpenCV offers this algorithm based on the 2D image points of the target object. It expresses the target position in the camera coordinate using inverse transformation to see it more friendly. This algorithm supposed that all the points lie on the same plane because its depth is mostly same, compared to the distance from the camera. Once the initial pose is obtained, the rotational and translation vector is solved by linear system and approximates its solution using scale orthographic projections.
4 TECHNICAL SECTION

4.1 Block diagram for Pose estimation system

The LED image on the face of a target satellite is captured from a Pi camera processed on a Raspberry Pi board and OpenCV supports various packages to process necessary image processing procedures and algorithms such as calibration and feature extraction. Using a POSIT algorithm, it obtains the relative pose estimation in terms of rotation and translation matrix. To test the performance, a LED pattern was designed on a PCB and the captured image is delivered to onboard processor. The images are tested from different perspectives and distances. The final outcome for pose estimation can also be transferred to the PC via serial or USB communication to display the information of pose estimation and distance, however, we limit this function by only checking its values within a RPi board.

![Block diagram for pose estimation using Raspberry Pi](image)

Figure 10. Block diagram for pose estimation using Raspberry Pi

The accuracy of rotating angle and distance can be determined by the comparison between measured value by scale and the output of pose estimation algorithms. The overall performance is checked by applying different angle views and distance between satellites and also evaluate its reliability under variable light intensity to simulate variable sunlight in space environment. By combining two separate patterns of IR LED and black and white background of each LED, it is also tested whether pose estimation is still achieved by both.

4.2 Hardware and Software

This section describes how hardware and software tasked have been done to achieve pose estimation using Raspberry Pi and algorithms based on OpenCV. The Pi camera is easily compatible with Raspberry Pi board as illustrated in Figure 11, and its interface is supported by Raspbian.
OS[54][55]. Any type of image and video can be defined by various parameters to set up the properties of images to be captured. After then, the data is processed by algorithms on the multi core threads and gives out the results for pose estimation. During the experiments, there was delay and frequent bugs in compiling and running files which significantly affects the experiment, so the platform was changed to the PC environments where OpenCV runs more stable.

![Image](image.jpg)

**Figure 11. Raspberry Pi 3 and Pi cam connection**

### 4.2.1 Hardware

#### 4.2.1.1 Raspberry Pi Board and Pi Camera

A Raspberry Pi3 is a suitable item to process many images. The most recently released RPi 3 offers an upgraded performance than previous models. It provides high computing power and reliability with the aid of light operating system. Upgraded Core clock, GPU, and memory is a reasonable option for processing video streaming in motion. The Python or C language can also be compiled based on Raspbian OS and various peripheral are supported as Table 1.

<table>
<thead>
<tr>
<th>Chip</th>
<th>Raspberry Pi 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>Core architecture</td>
<td>Broadcom BCM2837 64 bit processor 1.2GHz,</td>
</tr>
<tr>
<td></td>
<td>Quad-Core ARM Cortex-A53</td>
</tr>
<tr>
<td>GPU</td>
<td>Dual Core VideoCore IV® Multimedia Co-Processor,</td>
</tr>
<tr>
<td></td>
<td>hardware-accelerated OpenVG, and 1080p30 H.264 high-profile decode</td>
</tr>
<tr>
<td>Memory</td>
<td>1GB LPDDR2</td>
</tr>
<tr>
<td>Operating System</td>
<td>Boots from Micro SD card, running a version of the</td>
</tr>
<tr>
<td></td>
<td>Linux operating system or Windows 10 IoT</td>
</tr>
<tr>
<td>RAM</td>
<td>1GB LPDDR2</td>
</tr>
<tr>
<td>Dimension</td>
<td>85 x 56 x 17mm</td>
</tr>
<tr>
<td>Power</td>
<td>Micro USB socket 5V1, 2.5A</td>
</tr>
<tr>
<td>Connection</td>
<td>10/100 BaseT Ethernet socket, 15-pin MIPI Camera</td>
</tr>
<tr>
<td></td>
<td>Serial Interface (CSI-2), 40 pin GPIO, 4 x USB2,</td>
</tr>
<tr>
<td></td>
<td>HDMI output</td>
</tr>
</tbody>
</table>

Table 2. Raspberry Pi 3 specification
There are two commercially available Raspberry Pi cameras, and it is appropriate to use IR sensor camera to use at night and day, and to test in different sunlight. This camera does not have IR filter on a lens, so it can detect both visible and infrared sunlight. Its specification is described in the Table 2. This high resolution of a CMOS image sensor has a focal length of a camera as 3.6 mm, and the FoV is 53.50 H x 41.40 V degree. A NOIR camera is removed its IR filter, so the infrared camera is seen in the image. This has a slightly different color than the image with visual light camera.

<table>
<thead>
<tr>
<th>Criteria</th>
<th>Raspberry Pi- IR Camera Specification</th>
</tr>
</thead>
<tbody>
<tr>
<td>Resolution</td>
<td>5 Mega Pixel ((2592 * 1944 pixel)</td>
</tr>
<tr>
<td>Frame rate</td>
<td>1080p: 30 fps, 720p: 60 fps</td>
</tr>
<tr>
<td>Size</td>
<td>20 * 25 * 10 mm, 1 Pixel (1.4 um x 1.4 um)</td>
</tr>
<tr>
<td>Weight</td>
<td>3 g</td>
</tr>
<tr>
<td>Function</td>
<td>Automatic for Exposure control, white balance, band filter, luminance detection, black level calibration</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 3. Raspberry Pi Camera specification

4.2.1.2 LED Glyph pattern

As standard cube sat size is 100 by 100 mm, the size of the LED glyphs on satellite faces can be determined [5][6][7]. The satellite has vertical frames on each face, and the actual size of PCB is 80 by 80 mm smaller than 100 by 100 mm in Figure 12. As it is unknown from which direction rendezvous and docking is to be approached, a LED pattern size on a docking face should be determined according to the size of the face on a satellite.

Figure 12. Cubesat size
To design LED patterns, there are at least both eight faces for larger areas and four for small area, each 80 by 80, and 80 by 15 mm, there should be different patterns as Figure 13. To make individual patterns for 8 patterns, at least 4 LED should be used, however, many LEDs may affects the extent of accurate detection. Considering failing in detection one of LEDs, it was considered to use five LEDs rather than four. They are mirror symmetrical pattern, and thus has unique pattern even though it is rotates vertically or horizontally. To be short, the number of LEDs was set to five for 80 by 80 mm faces, and four LEDs were used for 80 by 15 mm faces.

As two mirrorsats and a coresat have to dock from different orientation, LED pattern has to be different patterns on each face and also its size has two types of pattern according to the size of a face.

For a test, the first glyph pattern illustrated in above figure was designed with its asymmetry, five LED were mounted on a PCB board. To mount LEDs, the distance between LEDs is 80/3=26.66mm and each LED has 4 lattices. To make LEDs more distinguishable from far distance and avoid of detecting LEDs, its background on LED has white color and the black one on no LED. Such pattern makes it detect the pattern easier during day and night even when the LED light gives weak intensity, and a black and white background supports LED patterns to be easily detected.

The power is supplied via USB cable from a PC. This power connection makes it an easy test in any environment. The forward drop voltage of a LED is 1.35 V, and each LED is connected in parallel as Figure 14. 5V power is supplied from a USB port and the total current is around 50 mA, It consumes the power consumption at around 5V * 50 mA, 250 mW, and is enough to emit bright-
To assume the LED brightness is not enough to be detected, the background of the LED will be white and other unoccupied by LED is surrounded by black color. This makes it easier to be detected even the LEDs is not visible properly due to external disturbances or errors in internal imaging processing.

To make it easier to do experiment, LED PCB board was designed to supply its power from the USB port and it is convenient to light the LED from the PC.

<table>
<thead>
<tr>
<th>Voltage</th>
<th>Forward drop (Parallel)</th>
<th>Voltage on load</th>
</tr>
</thead>
<tbody>
<tr>
<td>5V</td>
<td>1.33</td>
<td>3.6</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Resistance</th>
<th>Total Current</th>
<th>Current on each LED</th>
</tr>
</thead>
<tbody>
<tr>
<td>33R</td>
<td>3.6 V /33R=110 mA</td>
<td>110 mA / 5 leds=22 mA</td>
</tr>
</tbody>
</table>

Table 4. Reconfigurable satellites of the AAReST Mission

4.2.1.3 Field of View

The captured LED pattern has different size on an image plane according to the distance. The geometry can be calculated according the distance between pixels. When the target is closer until it is full of image, its corresponding FoV is 53.9 degree and its pixel number is 2592 pixels with respect to a horizontal axis on an image plane.
In the same way, the distance regarding a vertical axis can be calculated as below. Its relationship is described as below and drawn in Figure 15.

\[
\text{Distance : half of PCB size} = \text{focal length : half of image plane (total pixel number/2 * pixel size)}
\]

\[
\begin{align*}
\text{Horizontal} & : \text{Dis}_\text{hor} = 26.6 = 3.66 : 1.814 \\
& \Rightarrow \text{Dis}_\text{hor} = \frac{26.6 \times 3.66}{1.814} = 53.669 \\
\text{Vertical} & : \text{Dis}_\text{ver} = 26.6 = 3.66 : 1.36 \\
& \Rightarrow \text{Dis}_\text{hor} = \frac{26.6 \times 3.66}{1.36} = 71.585
\end{align*}
\]

**Figure 16. Calculation pixel number at full-size image**

Here, the distance for vertical direction is shorter than that of horizontal. This means that if the distance between two satellites is closer than 71.585mm, the image is full of either of axis on an image plane. For the most cases, the experiments will be carried out farther than this distance. Once the distance increases, the number of pixels occupied by the LEDs will also be smaller, however, it is possible to calculate how many pixels the LED occupies in the image plane. If the distance is set to 10 m, the pixel number is calculated as below.

\[
\begin{align*}
\text{Dis}_\text{hor/ver} & = 1000 \times \tan(26.75^\circ) = 504mm \\
\text{pix}_\text{LED}_\text{hor} & = \frac{4.5}{504} \times 2592 = 23\text{pix} \\
\text{pix}_\text{LED}_\text{ver} & = \frac{4.5}{504} \times 1944 = 17.3\text{pix}
\end{align*}
\]

**Figure 17. Calculation pixel number of LED at 1 m away**

It still takes many pixels, so it requires centroid algorithms to determine the exact center position of LEDs on the image plane.
4.2.2 Software

There are several stages to detect the target image and extract features to use them for pose estimation. After knowing the exact center point of each led, pose estimation algorithms is processed using the number of keypoints. Its overall procedures are described as a flow chart and its flow chart and its brief explanation is followed. Once the image is captured from the camera, it goes through thresholding process in which the gray level of pixel is translated to zero to one according the threshold level.

4.2.2.1 OpenCV library

Image processing requires high level of processing power and advanced knowledge to deal with images. OpenCV (Open source computer vision library) is easily accessible on the web for the purpose of computer vision[54]. It is compatible with many languages such as Python, Java, C or C++, and also runs in many different operating systems. For this project, various sources from the OpenCV were adopted to be utilized for running on a Raspberry Pi.

The image processing for pose estimation is actively utilized by users on the web and its processing steps are explained in the flow chart Figure 19. To check camera’s function and IR LEDs operation, sample images and videos were captured at default setting on RPi board. It gave results without any problem. The LED lights are captured from a Pi camera although it is invisible to human eyes. It is noticeable to know that Pi camera is working properly.

For this project, it is recommended to know the general function of OpenCV library, and it makes it easy to call it whenever needed. Here the short summary of the OpenCV is listed.
<table>
<thead>
<tr>
<th>Library</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>cv.h</td>
<td>Default function for computer vision, image processing algorithms</td>
</tr>
<tr>
<td></td>
<td>Ex) cv.h, cvXXX.lib, cvXXX.dll (*.Lib: library file, *.dll: dynamic library</td>
</tr>
<tr>
<td></td>
<td>when running</td>
</tr>
<tr>
<td>cvaux.h</td>
<td>Special functions for vision algorithm</td>
</tr>
<tr>
<td></td>
<td>Ex) cvaux.h, cvauxXXX.lib, cvauxXXX.dll</td>
</tr>
<tr>
<td>cvcore.h</td>
<td>Support Matrix data and data structure</td>
</tr>
<tr>
<td></td>
<td>Ex) cvcore.h, cvcoreXXX.lib, cvcoreXXX.dll</td>
</tr>
<tr>
<td>highgui.h</td>
<td>Related to GUI(reading, window, mouse/keyboard, camera/video handling)</td>
</tr>
<tr>
<td></td>
<td>Ex) highgui.h, highgui.lib, highgui.dll</td>
</tr>
<tr>
<td>ml.h</td>
<td>Functions for machine learning</td>
</tr>
<tr>
<td></td>
<td>Ex) ml.h, ml.lib, ml.dll</td>
</tr>
<tr>
<td>Cvcam.h</td>
<td>Camera input/output</td>
</tr>
</tbody>
</table>

Table 5. OpenCV library functions

4.2.2.2 Raspberry Pi Camera Operation

After installing Raspbian OS, its default function was checked and tested for image capturing and sample code compiling. The IR LED image was successfully captured by RPi camera.

![Image capture test](image)

Figure 19. Image capture test

4.2.2.3 OpenCV Function Test

As OpenCV supports algorithms such as pose estimation and feature extraction, it is easily adoptable to apply those algorithms to test edge detection as the first step. A sample image was processed to check if the edge of features is detected. The edge of an object in raw image file is detected and successfully shows that the Canny edge was detected from the image.
4.2.2.4 Camera Calibration

The camera calibration was done for RPi cam, but also for PC webcam. To test much complex code, it sometimes needed to verify its function on a faster processor first. By using the chessboard, the code takes several images and gives out the intrinsic parameter. As previously explained, its measured parameter value are described in the chart.

<table>
<thead>
<tr>
<th>Focal length(Fx)</th>
<th>Focal length(Fx)</th>
<th>Center Point(Cx)</th>
<th>Center Point(Cx)</th>
</tr>
</thead>
<tbody>
<tr>
<td>785.76802</td>
<td>786.5981</td>
<td>333.756</td>
<td>242</td>
</tr>
</tbody>
</table>

Table 6. Camera Calibration parameters

Distortion parameters are also measured as below.

<table>
<thead>
<tr>
<th>K1</th>
<th>K1</th>
<th>P1</th>
<th>P2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.15</td>
<td>-0.91</td>
<td>0.009</td>
<td>0.003</td>
</tr>
</tbody>
</table>

Table 7. Camera distortion parameters
4.2.2.5 Thresholding

To detect the light of LED, it is needed to find the bright point in the image. To do so, the pixels which is included in the light can be converted to the grayscale level from 0 to 255, and grouped together depending on the gray level of the pixel compared to threshold level. Using “Threshold binary” type among threshold types, the pixel is regarded to zero (black) or 1 (white) when its gray level is lower than threshold. Using a slide bar to adjust threshold value, the value is adjusted to adapt to the experiment environment and the threshold value was set to 200 to eliminate the other background brightness.

![Threshold test](image)

Figure 22. Threshold test

4.2.2.6 Edge detection

As capturing the LED light is a beginning step for image processing, it was tested to detect the edge of five LEDs. With a Pi camera, it is clearly shown in Figure 23 that five LEDs are lighting and its edge was processed using OpenCV. Later, the center of each led should be calculated and the position and distance between each LED needs to be processed using pose estimation algorithm and determines rotation and distance for pose estimation.

![Edge detection test](image)

Figure 23. Edge detection test
4.2.2.7 Blob detection

To detect blob, it requires several parameters to define the properties of blobs. By utilizing SimpleBlobDetector which is provided by OpenCV, blobs were detected after setting parameters. After several trials, the optimal parameters such as min and max area and filtering parameters such as circularity, convexity and inertia which represents shape properties were fixed. Once 5 blobs of the IR LED are detected, it generates the contour of the blobs in red circle to calculate the center point of the blobs.

![Figure 24. Blob detection test](image)

4.2.2.8 Centroid algorithm

Once the image is captured and processed to detect the edge or features, it is required to obtain the center point of the pattern in order to it for the input value of pose estimation algorithm. Using the 0\textsuperscript{th} and 1\textsuperscript{st} order moment, the center point of the blobs of the five LEDs were calculated and its position is displayed.

![Figure 25. Centroid test and position of centers](image)

4.2.2.9 Pattern matching

There are various algorithms to find features and matches the image to the template such as FAST,
BRISK (Binary Robust Invariant Scalable Keypoints), Generalized Hough Transform and Normalized Grayscale Correlation. Simple algorithm utilizing classical matching and tracking was tested, however, it didn’t detect the pattern in rotation properly and nor easy to match its features if the object’s rotation is more than 1 DOF. To detect the led pattern robustly,

![Image](image_url)

**Figure 26. LED pattern matching test**

Instead of classical pattern matching, it was developed manually to recognize this 5 led pattern. As blob detection by OpenCV assigns the keypoint number randomly to each LEDs, it was necessary to develop the algorithm. The sequential index was allocated to 5 LED according to the geometry which forms the two lines and intersects at the index 3 LED. The increasing order was from the top right to top left and then to bottom right. The algorithm is described as below, and it is rotation and scale invariant.

To match the LED patter, the algorithm was developed to find the reference LED and allocate the index to each LED. There are 5 LED which considers the planar pose estimation requires at least 4 and the middle point has same distance to the LEDs on its side. To easily make indexing, the first index was fixed to the end of the corner. Using the different distance between the LEDs, the two middle pont(index 1,3) were found, and the remaining led was assigned by comparing the distance to the Led1 and the angle of the led 0 with 90 was a key factor to determine which LED is index 0. By developing this algorithm, it is rotation invariant when the LED is rotated to any direction along yaw axis and robustly recognized its pattern with the proper index. Its detail algorithm is explained as below.
**Algorithm: To assign index to 5 LED**

1. To find the LED on the middle of the 1st line
   For loop
   - Choose any LED out of 5 LEDs
   - Choose any LED out of remaining 4 LEDs
   - Choose any LED out of remaining 3 LEDs
   - Compare if one LED is in the center of the other 2 LED
   - Save the index of the 3 LED when detected.
   - Save the distance0 from the center to the side

2. To find the another LED on the middle of the another line
   For loop
   - Choose any LED out of 4 LEDs
   - Choose any LED out of remaining 3 LEDs
   - Choose any LED out of remaining 2 LEDs
   - Compare if one LED is in the center of the other 2 LED
   - Save the index of the 3 LED when detected
   - Save the distance1 and 1 from the center to the side

3. To Assign index 1 to the LED among two detected LEDs.
   - Compare the dist0 and dist1
   - assign index 1 for the led with shorter distance.
   - assign index 3 for the led with longer distance.

4. To Assign index 0 to the LED in the corner
   For loop
   - Choose any LED out of the remaining 3 LEDs excluding the LED
   - If the angle between two line from the led to other two led is about 90 deg(∠315)
     - Assign the index 0 to the led

5. To Assign remaining index 2 and 4 to the remaining LEDs
   - Save the dist3 to the index0 to the one LED
   - Save the dist4 to the index0 to the other LED
   - Compare dist3 and dist4
   - Assign index 2 to the led with short distance to the index0
   - Assign index 4 to the last led

**Figure 27. Rotation invariant algorithm to find the index out of 5 LEDs**

**4.2.2.10 Pose Estimation**

Basically, pose estimation algorithm uses the 2D image points to estimate rotation and translation vectors of camera from at least 4 corresponding points. From the output of the blob detection and
centroid algorithm, it is possible to provide the information of at least 4 points of LEDs as input to this algorithm. It takes camera intrinsic parameter which consists of the focal length, skew and projection center. As these parameters are not consistent, it is desirable to choose the close value after repeating calibration. Distortion coefficient is assumed to have small values. Once these input are provided, this algorithm embedded as a OpenCV library calculate the rotation and translation of a camera coordinate w.r.t world coordinate. To find the nearest output, it goes through Levenberg-Marquardt which minimized the square of errors for optimization. Here summarize a brief information for input and output.

<table>
<thead>
<tr>
<th>Input</th>
<th>2D image point of the LED, camera matrix/distortion coefficient</th>
</tr>
</thead>
<tbody>
<tr>
<td>Output</td>
<td>rotation vector, translation vector</td>
</tr>
</tbody>
</table>

Table 8. Pose estimation function

To do this, it needs steps as below.

1. Identify at least 4 points in the 3D world coordinate to the input of 2D image points
2. Project those points to the image coordinate as 3D points.
3. Use pose estimation algorithm to generate relative pose of the object to the camera

Here, the 3D point of five IR LED is spaced at constant distance vertically and horizontally, which is fixed to 26mm. Using this value, it is mapped into the 3D points of the each LED. For example, when the index 3 led is regarded to be located near the center of the image at the initial pose, its 3D point is given as (0,0,0). At the same time, the index 0 LED at the top right corner has the position of (+26,-26,0). Its 3rd component is all zero, because it is lies on the image plane. By taking the picture of the target object, and once the five LEDs are given its unique positions on the 2, it is possible to recover the 3D point in the world coordinate from the 2D point on the image coordinate using the focal length and translating by center coordinate of the image plane. As there relationship is simply geometrical resemblance, its processing computing power is not demanding. To transform the 3D point in the world coordinate to the 2D image plane, it can be easily expressed by this logical flow. Once we put the keypoint of the LED on the normalized image at the distance 1(unitless), and it projects its imagePoint[i] at the focal length distance. Its detail procedure is described as below.

Input parameter of the pose estimation requires the image points which corresponds to the object in 3D coordinate. Assuming that more point may lead to better accuracy, it was tested if it may give higher accuracy. In order to so, the distance between LED with index 0 to 4 have been used to create 3 by 4 pattern of LED array which add virtual 7 led on the image. Those all 12 points were
added to the input of the pose estimation algorithm. To define the position of the virtual LEDs, the vector properties and the distance between LEDs were used and interpolated.

\[
3D_{World\ Points}[i] = 3D_{World\ Points}[i] - World\_Origin; //\text{distance from world center}
\]

\[
3D_{World\ Points}[i] = \text{Rotate}(3D_{World\ Points}[i], r); //\text{rotation}
\]

\[
\text{keyPoints}[i] = 3D_{World\ Points}[i] + \text{translation}; //\text{translation}
\]

\[
\text{imagePoint}[i].x = \text{focalLength} \times \text{keyPoints}[i].x / \text{keyPoints}[i].z //\text{X\_point}
\]

\[
\text{imagePoint}[i].y = \text{focalLength} \times \text{keyPoints}[i].y / \text{keyPoints}[i].z //\text{Y\_point}
\]

Figure 28. Point mapping between 3D world and 2D image coordinate

To recognize its orientation, the camera coordinate frame was drawn at the center of the LED pattern which is at the LED with index 3.

5 EXPERIMENT

This section describes the several experiment to measure the accuracy of the pose estimation in short range, different angle view and light intensity.

5.1 Setup and Procedure

5.1.1 Setup

To measure the accuracy in short range such as translational, rotational and processing time, and light intensity, measuring distance from the camera to LED pattern was set to different value ranging from 20 cm to 100 cm. To represent the majority patterns of the LED glyph among the patterns, 5 LED glyph was selected to measure its accuracy for this test.

To verify the accuracy of the results, it is necessary to compare those values to the ground truth. To measure real value, ruler was used along X, Y, and Z axis with respect to camera coordinate. For translation, measurement of the depth along Z-axis requires high accuracy, so the ruler was always kept fixed to ensure its measurement upto +/-1 mm. In addition, to ensure the orthogonality of the translation along X and Y axis onto Z-axis, their directions were measured by several square objects and were checked to provide their measurements upto +/-1 mm. For rotation, protractor was used to check its 90 degree between each axis and its measurement was assured to be upto +/-1 deg.

The final test for the algorithm was implemented on the PC environment and its image was captured by a webcam which has a resolution 640 by 480. Its computing power was
5.1.2 Parameters

As the measurement is highly dependent on the performance of the camera sensor, it is required to analyse the characteristics of the sensor in advance. Foremost, image pixel number and intrinsic parameters were checked because it strongly influences on the calculation of the image properties. The default and maximum image pixel is 640 by 480, however its size can be adjustable to test the accuracy along image quality. Higher image resolution leads to more pixel numbers taken by LED, making its range to be more discernible. However, this requires higher processing time by handling more pixels and reduces the frame rate of the images. In contrast, if the resolution is too low, it deteriorates the accuracy, however its processing times goes faster and more image frames can be handled. The distance between each led is 2.66 and the position of the index 0 to 4 LEDs were used using at least four points. Their position was extracted by the centroid algorithm and were given to the input of the pose estimation algorithm.

To further tests, the performance of processing is also influenced by the number of LEDs, however, it is fixed to 5 for this experiment, assuming the camera detect all LEDs without failure. If the number of led is increased, it may require longer processing time. For camera intrinsic parameters, the most suitable parameters were chosen after several repeating calibration.

<table>
<thead>
<tr>
<th>LEDs #</th>
<th>Camera resolution</th>
<th>Focal length</th>
<th>Distortion (Center point)</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>640 x 480</td>
<td>760</td>
<td>(K1 K2 P1 P2 P3)</td>
</tr>
</tbody>
</table>

Table 10. Camera parameters

For image processing, there are different types of parameters for code. Here it illustrates each stages of image processing such as thresholding and blob detection. These value are chosen after many trials to find the optimum value to detect the LEDs. The threshold value was set to 200 out of 255, which strongly rejects other white features, only remain highly bright object. For blob detection,

<table>
<thead>
<tr>
<th>min Area</th>
<th>max Area</th>
<th>minCircularity</th>
<th>minConvexity</th>
<th>minInertiaRatio</th>
</tr>
</thead>
<tbody>
<tr>
<td>50</td>
<td>300</td>
<td>0.4</td>
<td>0.4</td>
<td>0.4</td>
</tr>
</tbody>
</table>

Table 11. Blob detection parameters
Initial 5 point of the IR LED was given as the position in the 3D world coordinate as a form of P(-26.6mm,-26.6mm,0) for the LED2. Its coordinate is illustrated as below.

5.1.3 Procedures

Before measuring the pose estimation in different distance, angle view were attempted, detecting 5 LEDs was checked by software and confirmed if the algorithm successfully processes the image and produces the pose estimation. First, the led indexing algorithm was tested if it assign the the first index to the reference LEDs in the top right corner. When the camera detects led glyphs, it was checked if the 1st index is given to the reference LEDs. Once the indexing is successful.

To detect 5 LED from any orientation and distance, the pattern recognition algorithm was first examined and checked if they assign the index 0 to the reference LED and increase its index along adjacent LEDs. X-axis is to the right and Y-axis is downward.

To check how much error the estimated result has compared to the ground truth, every distance and angle were measured using ruler and protractor. If the LED is too close more than 7 Cm, it is full of the image plane which is larger than the FOV, as the distance was started from at least 10 Cm. To observe its tendency in the amount of interval, several steps of distance and angle view were test. However, due to the instability when it is more than 1 meter and 30 degree, LED detection was not successful and the test distance is limited to within 1 meter and angle view was also constraint to upto 30 degree.

For translation and rotation, led pattern rather than sensor was positioned at different distance and angles because it is much easier to move. By considering relative motion w.r.t. the camera coordinate, the distance and angle was carefully checked along the optical axis(Z) and also done for perpendicular two axis. To prevent misreading of the measurement, same steps for measurement were tested three times to obtain average value. Once the test on the Z-axis is successful, the pa-
rameter was fixed to the tests on X and Y axis. The frame rates was checked to more than 5 fps.

![Frame rates measurement](image)

**Figure 30. Translation measurement along Z-axis and frame rates**

To test if the accuracy is increased when using many more points than five, the vector composition between each LED was analyzed and interpolated to determine the virtual LED points. The figures shows more LED were created as virtual points by interpolation. However, over all accuracy was similar to the case of not using the interpolation.

![More image points](image)

**Figure 31. More image points using interpolation**

### 5.2 Result

This section present how much the estimated result is close to the ground truth value and how error ratio is distributed over distance along each axis and different angle view. After measuring estimated value, its error was calculated comparing with the value obtained from a ruler or a protractor. The graphs show the absolute error and relative error.

#### 5.2.1 Translation

The measurement along the 3 axis showed that its result is accurate with less than 6%. The accuracy showed the best result within the suitable distance from 30 Cm to 80 Cm, and around the center point of an image.
Analysis: The figure above shows that it has high accuracy along Z-axis. It has an error ratio less than 2%. It showed the error is larger than 10% when the pattern is very close to the camera, because it leads to too big blobs which are difficult to be detected properly with its size and geometry. However, once the pattern is placed within the range of the min and max blob size when farther than 40 cm, it gave very accurate result of the average error ratio up to 0.6%.

As the distance becomes larger, the error also increased slightly. The error at less than 80 cm, its error ratio stayed around 1% and it is attributed to its suitable blob size and occupied pixels to calculate the algorithms such as blob detection and centroid, and further to pose estimation algorithm. However, at the distance more than 80 Cm, the error ratio is steadily increased due to not much difference in occupied pixel numbers and the difficulty in detecting small blobs.
Figure 34. Measurement along X-axis

Figure 35. Measurement (left) and error ratio (right) along X-axis

Analysis: The measurement along X axis was tested at the 60 cm distance because the measurement Z-axis showed the most accurate result. The translation error along X-axis stayed around 10% and it had the most accuracy around the center of an image. It is because the pixel difference at the center of an image brings the highest sensitivity and it gives the big difference in angle and distance in movement along the X-axis and it thus gives the highest sensitivity to the input of the pose estimation algorithm. When it is around the center, the accuracy was minimized to the 4.8 %, however, its error is increased as the camera is moving toward the side of the image which means it has less variance in pixel image, so the error is increased.

Figure 36. Measurement along Y-axis
Analysis: In a same manner, the measurement along X axis was tested at the 60 Cm distance because the measurement Z-axis showed the most accurate result. The translation accuracy was highest when the distance is around the center point an image. Around the center point, its accuracy reached upto 1.8 %, however, it is increased as the distance from the center is growing to the side of right or left. Because field of view, As the error includes the wrong measurement of center point of the LED and misalignment of the LED arrangement and measurement of the scaler, its accuracy in ideal cases is regarded to be higher.

5.2.2 Rotation

The measurement in rotation was tested at the 60 cm distance because the measurement Z-axis showed the most accurate result. It was then rotated slow while measuring its rotating angle with a protractor.
Analysis: It is measured along the optical axis, its error was relatively small compared to the rotation error in X and Y axis. As this rotation does not have any projection, its rotation is directly linked to the rotation matrix itself without any ambiguity of projection. As seen from the figure above, the error decreased as the rotation angle becomes away from the reference point. Its average was less than 2%. There were not much difference in error ratio as it rotates. It is because the angle error is not sensitive to the direction of the rotation because the normal direction of the rotation plane is normal to the camera direction.

Analysis: For X-axis translation, the camera first pointed its normal direction, its accuracy was high as an error ratio of almost zero like the rotation along Z-axis has similar value. However, as the angle is increased, its accuracy becomes worse and stable measurable up to 20 deg. As the its algorithm is sequentially rely on the LED detection, the difference in distance when projected makes significant difference on the image. Once the angle is over 20 degree, its blob detection was not stable, which led to the inability to measurement of the pitch angle. Within 20 degree, its accuracy was obtained by overall less than 10%.

As the angle is larger, it gets more difficult to detect blobs and also the calculate of the center
point due to its elliptical shape. When the angle is larger, the projection increases, it gives the distorted image and make the estimation inaccurate. The highest accuracy was obtained around the zero degree when it is normal to the object, and it was measurable up to 20 deg. Once the angle is more than that value, its projected distance was not well suitable to the detection algorithm for LED pattern. Its overall error was 6.2 degree when they have projection.

![Image](image1.png)

**Figure 41. Measurement along Y-axis, from top to bottom**

Generally, the rotation is only measurable when the angle is within 20 degree, however, its angle can be extended. If the angle is detectable by 45 degree, it can increase its accuracy by placing different pattern on each face. Once the rotation is over 45 degree, and another face is becoming dominant to be seen, two different pattern can be utilized simultaneously for more accurate results.

### 5.2.3 Light intensity

The measurement in different light intensity was tested at the 60 Cm distance because the translation and rotation showed the most accurate result. Different light intensity has significant effect on the detection of LED when the light is projected from behind the LED. However, once it is observed with the light in front of the image its background noise greatly worsened the detection of the blobs. However, blob detection is relatively stable as it is detected depending on its size. When the light of camera is on, its light is shown to be much larger area, which is subject to suppressed.

![Image](image2.png)

**Figure 42. Measurement in different light intensity**
Figure 43. Translation(left) and Rotation(Right) error along Z-axis in different lights

**Analysis:** When it is dark, it has less error than where there is light. It is because it is easier to be detected its IR LED in darkness when there is even little light. The error was larger where the light intensity is bigger because the light from the circumstance deters the proper detection on Led pattern and sequentially affects on the algorithms. For translation, the highest accuracy was obtained at around 40~60 distance, which is best detectable with blob detection. Its translation error showed 7% and the rotation had accuracy upto 1.8 degree and had the least accuracy upto less than 20%.

### 6 CONCLUSION

#### 6.1 Summary

This paper aims to design and verify the rendezvous sensor system for the AAResST mission using monocular vision and infrared LEDs. In addition to reviewing previous works and works in Surrey Space Center, simple marker hardware using IR LED was designed and the software for pose estimation was also tested to measure its accuracy compared to the ground truth. Overall accuracy was within 6% in translation and rotation, and its best accuracy was obtained upto 0.6%. It was more sensitive when the distance is farther from the center, and the best result was obtained when is near 50 cm along Z-axis and around center of the image plane. Additionally, rotation measurement showed that higher angle led to increased error ratio. It demonstrated its strong measurement near distance upto 1 meter and normal direction to the camera. The goals of this experiment were ac-
complished in terms of pose estimation algorithm, but its accuracy level needs to be improved.

- **Design optical vision system using RPi board with 7 fps, more than 5 fps**
  
  The IR LED marker was placed in asymmetry to calculate its different shape and distance between each LED, and successfully captured the image with more than 5 frame/sec.

- **Obtained the accuracy of up to 0.6 degree in rotation**
  
  The measurement for Z-axis rotation showed it is highly reliable with up to 0.6 degree along Z-axis. However, for the X and Y axis rotation, it becomes unstable when the viewing angle is larger than 20 degree. It is due to the inconsistency of detecting blobs, giving more estimation errors.

- **Obtained the accuracy of up to +/- 6% distance in short (< 1m) range**
  
  Within 1 meter’s near range, its accuracy error ratio was less than 10 %, and was the best in near 50 cm when it is regarded to be the most clearly projected in the image. For further distance, the blob detection was not successful due to the other blobs in surrounding.

- **Test under different light intensities.**
  
  The LED detection is highly influenced by its surrounding light intensity, therefore, it was not robust to recognize the LED blob. Even the grid pattern on the background was difficult to be analysed enough.

Several fundamental development stages were conducted to utilize pose estimation including hardware and software. After installing the necessary package for RPi board and the PC with OpenCV, several tests were carried out on both platform and checked along the procedures as below. The algorithm was tested on the PC for final stage.

- OpenCV function test using RPi board and PC webcam
- Calibration to get camera intrinsic parameters
- Image processing for extracting the features (Thresholding, blob detection, centroid)
- LED marker tracking using vector property in rotation and scale invariant
- Experiments in different angle view, distance and light intensities.

To summarize, the pose estimation demonstrated that fundamental functions properly and presented accuracy in near range around 1 meter and small angle view under 20 degree.
6.2 Evaluation

The development environment for machine vision has been set up first with Raspberry Pi and later on PC. Several priori stages for pose estimation has been accomplished such as thresholding, blob detection, centroiding, pose estimation, pattern recognition and pose estimation. As the keypoint is changing, robust LED pattern detection was necessary to implement, and it was developed to assign the index to each LED in increasing order. When the LED is rotated or was seen in different shape, its index was used to calculate the orientation of the target object.

For experiments, several tests with different angle view and distance will be carried out to measure accuracy of pose estimation. To verify reliable performance, additional experiment under different light intensity is also to be check and final capability of this vision system is evaluated through the real-time motion in 2D air bearing. The result of each step of the experiments showed it is better than the previous works.

Several problems were discovered during the experiment. The issue and the approach is described here.

- **Problem 1:** Blob detection on multiple objects
  - **reason:** Different light intensity, irregularities in surrounding object.
  - **troubleshooting:** extract the keypoints of the LEDs based on distance and angle

- **Problem 2:** camera resolution
  - **reason:** webcam with low resolution to distinguish the object in farther distance
  - **troubleshooting:** tested its accuracy upto 1 m

- **Problem 3:** Low computation
  - **reason:** RPi has much overload to process image
  - **troubleshooting:** image processing and pose estimation requires iterative thread to find the optimal solution, hence the processing power of the RPi board was low and led to much delay for producing the result. It is considered to prove the performance of the algorithm on the PC in order to import it once its function is reliable.

- **Problem 4:** LED recognition and index allocation
  - **reason:** The index of the LED as a keypoint was changed when rotating and translation
  - **troubleshooting:** Created an algorithm to recognize the LED marker for rotation invariant

There are still limitations and it is summarized as below.
• **Narrow FOV**: The narrow field of view in the camera makes it difficult to obtain the measurement in farther distance toward X and Y axis, and to keep the target always visible within small area.

• **Infrared wavelength**: The wavelength of the LED is sometimes not detected properly in various light condition even with the same distance and angle perspective. There was significant difference even day and night inside a building.

• **Blob detection by background pattern**: The black and white pattern behind the led marker sometimes deters blob detection because of its reflectivity, so it needed to get rid of the pattern and covered with uniformly dark color behind LEDs.

• **Resolution**: the camera resolution of 640 by 480 should be higher to measure more accurately, which also affects the result of pose estimation. For faster computing process, and easier debugging environment, it was tested under PC environment. If there is a requirement to reduce delay due to resolution and enhance computing power, it would be also recommended to conduct the experiment onboard platform.

• **IR filter**: Non IR filter on a camera is desirable. As the webcam absorbs only the visual light without filter, its image may be weaker than its original intensity, which is most concentrated its intensity at infrared wavelength range.

• **Environment**: the experiment environment is important to capture marker only. Similar object determined to be blobs may distort the calculation and also brought wrong result.

### 6.3 Future Work

There remain several tasks to be accomplished to demonstrate the full functions of rendezvous and docking. As Noise has an influence on the image data in uncertainty, filtering algorithm is necessary to remove undesired data in the middle of processing and calculating the data. To do so, Kalman filter (or Extended) is recommended because it is one of the most widely used algorithm for robust filters in space navigation systems, but also majority of various unmanned systems. Owing to its robustness in variance noise sources, accurate prediction based on previous and current states is obtained using error covariance and adaptable gain[34]. Other than the prediction, detec-
tion can also be improved by using color LEDs. The RGB information combined with graylevel can provide better features to detect a target at night, combining IR LED at the same. If the marker has different RGB, it is much easier to detect the led in its unique position, and make the pattern recognition algorithm much easier without having to implement to recognize the pattern in rotation.

To summarize the further works:

- different color LEDs, different patterns, template matching algorithm
- sensor fusion: use LIDAR to increase accuracy
- Kalman Filter on measurement
- Communication with a PC
- Test 2D air bearing with different angle view, distance and light intensities.
- Close loop control for docking

This IR LED glyph based pose estimation algorithms will boost new approach on autonomous navigation for satellites and also contribute to designing low cost and simple vision system. Once it is achieved with high reliability, it is expected to be highly useful for autonomy in proximity control and navigation in many types of unmanned systems.
6. REFERENCES


[8] C. P. Bridges1, S. Kenyon2, STRaND: Surrey Training Research and Nanosatellite Demonstrator; 1st IAA Conference on University Satellite Mission and CubeSat Workshop January 24-29, 2011 Roma, Italy,


[23] Daniel Grest, *Marker-Free Human Motion Capture in Dynamic Cluttered Environments from a Single View-Point*


[36] NW Omer, Tracking and Pose Estimation of Non Tracking and Pose Estimation of Non-Cooperative Satellite for On-Orbit Servicing, isairas, 2012


[48] Tobias Nöll1, Alain Pagani1 , *Markerless Camera Pose Estimation – An Overview*, IRTG
1131 Workshop, 2011


[53] Pi Camera, https://www.raspberry.org/documentation/hardware/camera.md


[56] Andrew Kirillov, 3D Pose Estimation http://www.aforgenet.com/articles/posit/

[57] A Monocular Pose Estimation System based on Infrared LEDs.pdf

[58] BV RISK: Binary Robust Invariant Scalable Keypoints,

[59] Book, Learning OpenCV, Gary Bradski, O'Reilly


http://home.in.tum.de/~grembowi/ar2004_05/3dPoseEstimation_presentation.pdf

[64] G Schweighofer, Robust Pose Estimation from a Planar Target, 2006

[65] Patrick Maletz, Craig Underwood, AAR EST MISSION: Rendezvous Sensor, 2015
## APPENDIX 1 - WORK PLAN

<table>
<thead>
<tr>
<th>Task</th>
<th>Feb</th>
<th>Mar</th>
<th>Apr</th>
<th>May</th>
<th>Jun</th>
<th>Jul</th>
<th>Aug</th>
<th>Sep</th>
</tr>
</thead>
<tbody>
<tr>
<td>Literature review</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Raspberry Pi Set up</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>OpenCV Compile Setup</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>LED pattern/hardware design</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Vision Algorithm implement</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Camera Calibration</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Image processing with OpenCV</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Interim Report</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>PC development setup</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Pattern detection algorithm</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Code integration</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Translation Test</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Rotation Test</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Test in different lights</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Writing dissertation</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### Deliverables

- Theoretical analysis on image processing blob detection/Centroid algorithms
- Experimental results of pose estimation of LED pattern
- Software for pose estimation algorithms

- 54 -
APPENDIX 2 – COMPUTER VISION CODE

#include "stdafx.h"
#include <stdio.h>
#include <time.h>
#include <iostream>
#include <string>
#include <iomanip>
#include <math.h>
#include "opencv2/opencv.hpp"
#include "opencv/cv.hpp"
#include "opencv/highgui.h"
#include "opencv/cvaux.h"
#include "opencv/cxcore.h"
#include <opencv2/core/core.hpp>
#include <opencv2/calib3d/calib3d.hpp>

using namespace cv;
using namespace std;

//variables for threshold
int threshold_value = 220;
int threshold_type = 0;
int const max_value = 255;
int const max_type = 0;
int const max_BINARY_value = 255;

//variables for blob
Mat src, src_gray, im, dst, frame, src_centroid, im_with_keypoints, testMat;
std::vector<KeyPoint> keypoints;

cv::Point2f a, b, d, e, f, g, h, n, temp, middle0, middle1;

// LED interpolation for 3 by 4
int chessboardHeight = 3;
int chessboardWidth = 4;
Size cbSize = Size(chessboardHeight, chessboardWidth);

double* p;

// camera parameters
string filename = "out_camera_data.yml";

bool start = true;
const int FRAME_WIDTH = 640;
const int FRAME_HEIGHT = 480;

//variables to display texts on screen
//rotation
char string_rvec[50];
char string_rvecX[30];
char string_rvecY[30];
char string_rvecZ[30];

//translation
char string_tvec[50];
char string_tvecX[30];
char string_tvecY[30];
char string_tvecZ[30];

// Euler angles
char string_euler0[30];
char string_euler1[30];
char string_euler2[30];

char string_euler[50];
float rvecX = 0;
float rvecY = 0;
float rvecZ = 0;
float tvecX, tvecY, tvecZ;

//LED vector description
float square_dist0, dist0;
float square_dist1, dist1;
float square_dist2, dist2;
float square_dist3, dist3;
float square_dist4, dist4;
float square_dist5, dist5;
float dist_gn, dist_hn;
float deg_A;
int idx_a, idx_b, idx_c, idx_d, idx_e, idx_f, idx_g, idx_h, idx_n;
int k = 0;

cv::Vec3f euler;
float Rad2Deg = 180 / CV_PI;
#define PI 3.14

/// Function header
void Threshold(int, void*);
void thresh_callback(int, void*);
void RTVectors(Mat rvec, Mat tvec);
void assign_LED_index(std::vector<KeyPoint>);
void rot2euler(const cv::Mat &rotationMatrix);
void pose_dementhon(const std::vector<cv::Point3d> &wX, const std::vector<cv::Point2d> &x, cv::Mat &ctw, cv::Mat &chw);
Vec3f rotationMatrixToEulerAngles(Mat &R);
bool isRotationMatrix(Mat &R);

int main(int argc, char** argv)
{
    FileStorage fs;
    fs.open(filename, FileStorage::READ);
    Mat intrinsics, distortion;
    fs["Camera_Matrix"] >> intrinsics;
    fs["Distortion_Coefficients"] >> distortion;
    fs.release();

    Mat webcamImage, gray, one;
    Mat rvec = Mat(Size(3, 1), CV_64F); //rotation vectors
    Mat tvec = Mat(Size(3, 1), CV_64F); //translation vectors
    vector<Point2d> imagePoints, chess_imagePoints, imageFramePoints, imageOrigin;
    vector<Point3d> boardPoints, framePoints;
    double L = 26.6; //dimension: mm, distance between LED
    boardPoints.push_back(Point3d(L, -L, 0.0)); //0
    boardPoints.push_back(Point3d(0, -L, 0.0)); //1
    boardPoints.push_back(Point3d(-L, -L, 0.0)); //2
    boardPoints.push_back(Point3d(0, 0, 0)); //3
    framePoints.push_back(Point3d(0.0, 0.0, 0.0));
    framePoints.push_back(Point3d(3.0, 0.0, 0.0));
    framePoints.push_back(Point3d(0.0, 3.0, 0.0));
    framePoints.push_back(Point3d(0.0, 0.0, 3.0));

    //part of code below
    IplImage* imgScribble = NULL;
    CvMoments *moments = (CvMoments*)malloc(sizeof(CvMoments));
    VideoCapture cap;
    if (!cap.open(0))
        return 0;
    for (;;)
    {
        t_a = getTickCount(); // Mat frame;
        cap >> frame;
        if (frame.empty()) break; // end of video stream
        cvNamedWindow("WebCam", CV_WINDOW_FREERATIO); //CV_WINDOW_NORMAL,CV_WINDOW_FREERATIO
        imshow("WebCam", frame);
        waitKey(1); // too fast update one by 0.1 sec
        cvtColor(frame, src_gray, CV_RGB2GRAY);
        cvNamedWindow(window_name, CV_WINDOW_NORMAL, CV_WINDOW_FREERATIO);
        imshow(window_name, src_gray);
        if (trackbar_value > 100)
            trackbar_value = 100;
    }
}
window_name, &threshold_value, max_value, Threshold);

/// Call the function to initialize Threshold(0, 0);

// Setup SimpleBlobDetector parameters.
SimpleBlobDetector::Params params;

// Filter by Color
params.filterByColor = 1; // Set blobColor = 0 to select darker blobs, and
blobColor = 255
params.minThreshold = 190;

// Filter by Area.
params.filterByArea = true; //params.minArea = 1500;
params.minArea = 10; //only 1 point detected
params.maxArea = 300; //add

// Filter by Circularity
params.filterByCircularity = true;
params.minCircularity = 0.4;

// Filter by Convexity
params.filterByConvexity = true;
params.minConvexity = 0.4;

// Filter by Inertia
params.filterByInertia = true;
params.minInertiaRatio = 0.4;

// Storage for blobs
std::vector<KeyPoint> keypoints;

#if CV_MAJOR_VERSION < 3 // If you are using OpenCV 2
    // Set up detector with params
    SimpleBlobDetector detector(params);
    // Detect blobs
    detector.detect(im, keypoints);
#else
    // Set up detector with params
    Ptr<SimpleBlobDetector> detector = SimpleBlobDetector::create(params);
#endif

// Show blobs
detector->detect(im, keypoints);
drawKeypoints(im, keypoints, im_with_keypoints, Scalar(0, 0, 255), DrawMatchesFlags::DEFAULT);
namedWindow("blob", CV_WINDOW_FREERATIO);
imshow("blob", im_with_keypoints);
waitKey(1);

if (keypoints.size()<5)
    printf(" keypoints is %d, less than 5 \n", keypoints.size()); //to restore
else if (keypoints.size() > 5)
    printf(" keypoints is %d, more than 5 \n", keypoints.size()); //to restore
else
    printf("\n"); //to restore

int X[5], Y[5];
std::vector<KeyPoint> Newkeypoints;
int NewX[5], NewY[5];
namedWindow("Pose_Estimation", CV_WINDOW_FREERATIO);
putText(im_with_keypoints, "Pose_Estimation", cvPoint(30, 35), FONT_HERSHEY_COMPLEX_SMALL, 1,
cvScalar(0, 200, 200), 1, CV_AA); //O.K,
imshow("Pose_Estimation", im_with_keypoints);
waitKey(1);

char string_keypoint[30];
namedWindow("Pose_Estimation", CV_WINDOW_FREERATIO);
sprintf_s(string_keypoint, "Keypoint # is %d", keypoints.size());
putText(im_with_keypoints, string_keypoint, cvPoint(30, 55), FONT_HERSHEY_COMPLEX_SMALL, 1,
cvScalar(0, 0, 255), 1, CV_AA); //O.K,
imshow("Pose_Estimation", im_with_keypoints);
waitKey(1);
imshow("Pose_Estimation", im_with_keypoints);
waitKey(1);

Point Xaxis, Yaxis, Zaxis;
Xaxis.x = 10; Xaxis.y = 10;
Yaxis.x = 60; Yaxis.y = 10;
\[ Zaxis.x = 10; \ Zaxis.y = 60; \]

```cpp
line(im_with_keypoints, Xaxis, Yaxis, CV_RGB(255, 255, 0)); // -> x dir, red, to the right
line(im_with_keypoints, Xaxis, Zaxis, CV_RGB(0, 255, 255)); // | y-dir, green, to down
```

```cpp
putText(im_with_keypoints, "X", cvPoint(70, 14), FONT_HERSHEY_TRIPLEX, 0.3, Scalar(0, 255, 0), 1);
// green
putText(im_with_keypoints, "Y", cvPoint(8, 70), FONT_HERSHEY_TRIPLEX, 0.5, Scalar(0, 255, 0), 1);
// green
imshow("Pose_Estimation", im_with_keypoints);
waitKey(1);
```

```cpp
char string0[10];
char string1[10];
char string2[10];
char string3[10];
char string4[10];
char string5[10];
char string6[10];
char string7[10];
char string8[10];
char string9[10];
char string10[10];
char string11[10];
```

```cpp
sprintf_s(string0, "%d", 0);
sprintf_s(string1, "%d", 1);
sprintf_s(string2, "%d", 2);
sprintf_s(string3, "%d", 3);
sprintf_s(string4, "%d", 4);
sprintf_s(string5, "%d", 5);
sprintf_s(string6, "%d", 6);
sprintf_s(string7, "%d", 7);
sprintf_s(string8, "%d", 8);
sprintf_s(string9, "%d", 9);
sprintf_s(string10, "%d", 10);
sprintf_s(string11, "%d", 11);
```

```cpp
// Assign five point
if (keypoints.size() == 5) {
    cnt = 0;
    // find middle point
    for (int i = 0; i < keypoints.size(); i++)
    {
        a = keypoints[i].pt; // first pickup point among keypoint.size()-2
        idx_a = i;

        for (int j = 0; j < keypoints.size(); j++)
        {
            if (j != i)
            {
                b = keypoints[j].pt; // 2nd pickup point
                idx_b = j;

                // for (int k = 0; k < keypoints.size(); k++)
                for (k = 0; k < keypoints.size(); k++)
                {
                    if ((k != i) && (k != j))
                    {
                        c = keypoints[k].pt; // 3rd pickup point
                        idx_c = k;

                        // for
                        if ((a.x > ((b.x + c.x) / 2 - 10)) &&
                        (a.y < ((b.x + c.x) / 2 + 10)))
                        {
                            middle0 = a;
                            square_dist0 = (a.x - b.x) * (a.x - b.x) + (a.y - b.y) * (a.y - b.y);
                            dist0 = sqrt(square_dist0);
                        }
                    }
                }
            }
        }
    }
}
```

- 58 -
if (cnt == 1)
    break;
} else if (cnt == 1)
    break;
}

// 2. find another middle point among 4 points
for (int i = 0; i < keypoints.size(); i++)
{
    if ((i != idx_a) &&
        ((cnt == 1) && (k != idx_d)))
    {
        d = keypoints[i].pt; // 4th pickup point
        for (int j = 0; j < keypoints.size(); j++)
        {
            if ((j != idx_a) && (j != idx_d))
            {
                e = keypoints[j].pt;
                for (int k = 0; k < keypoints.size(); k++)
                {
                    if ((k != idx_a) && (k != idx_d) &&
                        (k != idx_e))
                    {
                        f = keypoints[k].pt; // 4th pickup point
                        idex_d = j;
                        // for
                        if ((d.x > (e.x + f.x) / 2 -
                             10) && (d.x < ((e.x + f.x) / 2 + 10))
                            && (d.y < ((e.y + f.y) / 2 - 10))
                            && (d.y < ((e.y + f.y) / 2 + 10)))
                    {
                        middle1 =
                            square_dist1 = (d.x - e.x)*(d.x - e.x) +
                              (d.y - e.y)*(d.y - e.y);
                        dist1 = sqrt(square_dist1);
                        cnt++;
                        break;
                    }
                }
            }
        }
    }
    if (cnt == 2)
        break;
}

// 3. find index1 point among two middle points
for (int i = 0; i < keypoints.size(); i++)
{
    if ((i != idx_a) &&
        ((cnt == 2) && (k != idx_d)))
    {
        g = keypoints[i].pt; // 4th pickup point
        idex_d = i;
        for (int j = 0; j < keypoints.size(); j++)
        {
            if ((j != idx_a) && (j != idx_d) && (j != idx_g))
            {
                h = keypoints[j].pt;
                idex_d = j;
                square_dist2 = (a.x - g.x)*(a.x - g.x) +
                              (a.y - g.y)*(a.y - g.y);
                square_dist3 = (a.x - h.x)*(a.x - h.x) +
                              (a.y - h.y)*(a.y - h.y);
                dist2 = sqrt(square_dist2); // between led1 and g
                dist3 = sqrt(square_dist3); // between led1 and h
                if (((abs(dist0 - dist2)) < 5) && (abs(dist0 -
                        dist3) < 5)) // for short line
                {
                    // need to assign later
                    if (dist0 < dist1) // middle point on a
                    {
                        led1 = keypoints[idx_a].pt;
                    }
\[ led3 = keypoints[\text{id}_d].pt; \]
\[ \text{else} \]
\[ \{ \]
\[ led3 = keypoints[\text{id}_a].pt; \]
\[ \}
\]
\[ \text{square_dist4} = (\text{led1.x} - \text{g.x})^2 + (\text{led1.y} - \text{g.y})^2; \]
\[ \text{dist4} = \sqrt{\text{square_dist4}}; \]
\[ \text{square_dist5} = (\text{led1.x} - \text{h.x})^2 + (\text{led1.y} - \text{h.y})^2; \]
\[ \text{dist5} = \sqrt{\text{square_dist5}}; \]
\[ \text{if} (\text{abs(dist4 - dist5)} < 10) \]
\[ \{ \]
\[ \text{for (int p = 0; p < keypoints.size(); p++)} \]
\[ \{ \]
\[ (p != \text{id}_d) && (p != \text{id}_g) && (p != \text{id}_h) \} // \text{don't know yet which is led 0 or led 2 between g,h} \]
\[ n = \text{keypoints}[\text{p}].pt; \]
\[ \text{dist_gn} = (\text{g.x} - n.x)^2 + (\text{g.y} - n.y)^2; \]
\[ \text{dist_hn} = (\text{h.x} - n.x)^2 + (\text{h.y} - n.y)^2; \]
\[ \text{if} (\text{dist_gn} > \text{dist_hn}) \] // compare g and h to led 0 and led 2
\[ \text{if g is led2} \]
\[ \text{Point2f h_left} = \text{h} - \text{led1}; // \text{to the left line} \]
\[ \text{Point2f h_down} = \text{h} - n; // \text{to the downward line} \]
\[ \text{float num} = (\text{h_left.x}^2 h_down.x + \text{h_left.y}^2 h_down.y); // \text{dot product, numerator} \]
\[ \text{float denum_h_left} = \text{abs}((\text{h_left.x}^2 h_left.x + \text{h_left.y}^2 h_left.y)); // \text{distance, x^2+y^2} \]
\[ \text{float denum_h_down} = \text{abs}((\text{h_down.x}^2 h_down.x + \text{h_down.y}^2 h_down.y)); // \text{distance, x^2+y^2} \]
\[ \text{if} (\text{num} / (\text{denum_h_left} * \text{denum_h_down}) < 1); // \text{dot product,} \]
\[ \{ \]
\[ \text{float rad_A} = \text{acos(num} / (\text{denum_h_left} * \text{denum_h_down})); // \text{angle between LED0-LED2, LED0 to LED4, should be 90 degree} \]
\[ \text{deg_A} = \text{rad_A} * 180 / \text{PI}; \]
\[ \text{if} (\text{deg_A} > 60) // \text{If tilted much, it is still considered larger than 60,} \]
\[ \{ \]
\[ \text{led2} = \text{keypoints[\text{id}_g}.pt;} \]
\[ \text{led0} = \text{keypoints[\text{id}_h}.pt;} \]
\[ \text{led4} = \text{keypoints[\text{id}_n}.pt;} \]
\[ \} \]
\[ \text{imagePoints.push_back(led0); // looks ok} \]
\[ \text{imagePoints.push_back(led1); // looks ok} \]
\[ \text{imagePoints.push_back(led2); // looks ok} \]
\[ \text{imagePoints.push_back(led3); // looks ok} \]
\[ \text{imagePoints.push_back(led4); // looks ok} \]
\[ \text{chess_imagePoints.push_back(led0); // looks ok} \]
chess_imagePoints.push_back(led1); // looks ok
chess_imagePoints.push_back(led2); // looks ok
chess_imagePoints.push_back(led3); // looks ok
chess_imagePoints.push_back(led4); // looks ok

namedWindow("Pose_Estimation", CV_WINDOW_FREERATIO);
circle(im_with_keypoints, (Point)led0, 10, CV_RGB(0, 255, 0)); // green
circle(im_with_keypoints, (Point)led1, 10, CV_RGB(0, 0, 255)); // blue
circle(im_with_keypoints, (Point)led2, 10, CV_RGB(255, 0, 255)); // pink
circle(im_with_keypoints, (Point)led3, 10, CV_RGB(0, 255, 255)); // cyan
circle(im_with_keypoints, (Point)led4, 10, CV_RGB(255, 255, 0)); // yellow

putText(im_with_keypoints, string0, cvPoint(led0.x, led0.y - 10), FONT_HERSHEY_TRIPLEX, 1, Scalar(0, 255, 0), 1); // green
putText(im_with_keypoints, string1, cvPoint(led1.x, led1.y - 10), FONT_HERSHEY_TRIPLEX, 1, Scalar(255, 0, 0), 1); // blue
putText(im_with_keypoints, string2, cvPoint(led2.x, led2.y - 10), FONT_HERSHEY_TRIPLEX, 1, Scalar(0, 0, 255), 1); // red
putText(im_with_keypoints, string3, cvPoint(led3.x, led3.y - 10), FONT_HERSHEY_TRIPLEX, 1, Scalar(255, 255, 0), 1); // cyan
putText(im_with_keypoints, string4, cvPoint(led4.x, led4.y - 10), FONT_HERSHEY_TRIPLEX, 1, Scalar(0, 255, 255), 1); // yellow

cnt++;
solvePnP(Mat(boardPoints), Mat(chess_imagePoints), intrinsics, distortion, rvec, tvec, false);
projectPoints(framePoints, rvec, tvec, intrinsics, distortion, imageFramePoints); // origin

line(im_with_keypoints, chess_imagePoints[0], chess_imagePoints[2], CV_RGB(0, 255, 255), 2); // cyan
line(im_with_keypoints, chess_imagePoints[0], chess_imagePoints[4], CV_RGB(0, 255, 255), 2); // green

imshow("Pose_Estimation", im_with_keypoints);
waitKey(1);

line(im_with_keypoints, imageFramePoints[0], imageFramePoints[2], CV_RGB(0, 255, 0), 2); // green
line(im_with_keypoints, imageFramePoints[0], imageFramePoints[3], CV_RGB(0, 255, 0), 2); // blue

line(im_with_keypoints, imageFramePoints[0], 20 * (imageFramePoints[1] - imageFramePoints[0]) + imageFramePoints[0], CV_RGB(255, 0, 0), 2); // red
line(im_with_keypoints, imageFramePoints[0], 20 * (imageFramePoints[2] - imageFramePoints[0]) + imageFramePoints[0], CV_RGB(0, 255, 0), 2); // cyan
line(im_with_keypoints, imageFramePoints[0], 20 * (imageFramePoints[3] - imageFramePoints[0]) + imageFramePoints[0], CV_RGB(0, 0, 255), 2); // blue

rvecX = rvec.at<double>(0, 0);
rvecY = rvec.at<double>(1, 0);
rvecZ = rvec.at<double>(2, 0);
tvecX = tvec.at<double>(0, 0);
tvecY = tvec.at<double>(1, 0);
tvecZ = tvec.at<double>(2, 0);

sprintf_s(string_rvecX, "%3.2f ", rvecX);
sprintf_s(string_rvecY, "%3.2f ", rvecY);
sprintf_s(string_rvecZ, "%3.2f ", rvecZ);
sprintf_s(string_tvecX, "%3.2f ", tvecX);
sprintf_s(string_tvecY, "%3.2f ", tvecY);
sprintf_s(string_tvecZ, "%3.2f ", tvecZ);

namedWindow("Pose_Estimation", CV_WINDOW_FREERATIO);
putText(im_with_keypoints, "        X        Y       Z", cvPoint(50, 75), FONT_HERSHEY_COMPLEX_SMALL, 0.8, cvScalar(0, 0, 255), 1, CV_AA);//O.K,

sprintf_s(string_rvec, "   rot:   %3.2f     %3.2f     %3.2f", rvecX, rvecY, rvecZ);
sprintf_s(string_tvec, "trans:  %3.2f     %3.2f     %3.2f", tvecX, tvecY, tvecZ);
putText(im_with_keypoints, string_rvec, cvPoint(50, 90), FONT_HERSHEY_TRIPLEX, 0.5, cvScalar(0, 255, 0), 1);//green
putText(im_with_keypoints, string_tvec, cvPoint(50, 110), FONT_HERSHEY_TRIPLEX, 0.5, cvScalar(0, 255, 0), 1);//green

RTVectors(rvec, tvec);

sprintf_s(string_euler0, "%3.2f ", euler[0]);
sprintf_s(string_euler1, "%3.2f ", euler[1]);
sprintf_s(string_euler2, "%3.2f ", euler[2]);

sprintf_s(string_euler, "        %3.2f      %3.2f      %3.2f", euler[0], euler[1], euler[2]);
putText(im_with_keypoints, "----------------------------", cvPoint(50, 120), FONT_HERSHEY_TRIPLEX, 0.5, Scalar(255, 255, 0), 1);//green
putText(im_with_keypoints, "       Roll     Pitch     Yaw", cvPoint(50, 130), FONT_HERSHEY_COMPLEX_SMALL, 0.8, cvScalar(0, 255, 0), 1, CV_AA);//O.K,

imshow("Pose_Estimation", im_with_keypoints);
waitKey(1);
imagePoints.clear(); // empty vector
chess_imagePoints.clear(); // empty vector

break;

} }}

if (cnt == 3)
break;

} if (cnt == 3)
break;

} // end of if, keypoint_size() >4
t_b = getTickCount(); //add
t = t_b - t_a; //time difference
\[
frame_t = (1 / \text{getTickFrequency()}) \times t; \quad // \text{time}
\]
\[
cout \text{.precision(3);}\]
\[
cout \ll " \text{fps}" \ll 1 / frame_t \ll \text{endl};\]
\[
\text{namedWindow("centroid", CV_WINDOW_FREERATIO);}
\]
\[
createTrackbar(" \text{Canny thresh:}", \text{source_window}, \&\text{thresh, max_threshold, thresh_callback);}
\]
\[
\text{int key = cvWaitKey(1);}\]
\[
\text{if (key > 0)}\{
 \text{break;}\}
\]
\[
}// \text{end of for(;;)}
\]
\[
}\// \text{end of main}
\]
\[
\text{void Threshold(int, void*)}\{
\text{threshold(src_gray, dst, threshold_value, max_BINARY_value, threshold_type);}\]
\[
\text{imshow(window_name, dst);}\]
\[
\text{im = dst;}\}
\]
\[
\text{void thresh_callback(int, void*)}\{
\text{Mat canny_output;}\]
\[
\text{vector<vector<Point>> contours;}\]
\[
\text{vector<Vec4i> hierarchy;}\]
\[
\text{// Detect edges using canny}\]
\[
\text{Canny(im_with_keypoints, canny_output, thres - thr + thr * 2, 3);}\]
\[
\text{// Find contours}\]
\[
\text{findContours(canny_output, contours, hierarchy, RETR_TREE, CHAIN_APPROX_SIMPLE, Point(0, 0));}\]
\[
\text{// Get the moments}\]
\[
\text{vector<Moments> mu(contours.size());}\]
\[
\text{for (size_t i = 0; i < contours.size(); i++)}\{
 \text{mu[i] = moments(contours[i], false);}\}
\]
\[
\text{// Get the mass centers:}\]
\[
\text{vector<Point2f> mc(contours.size());}\]
\[
\text{for (size_t i = 0; i < contours.size(); i++)}\{
 \text{mc[i] = Point2f(static_cast<float>(mu[i].m10 / mu[i].m00), static_cast<float>(mu[i].m01 / mu[i].m00));}\}
\]
\[
\text{// Draw contours}\]
\[
\text{Mat drawing = Mat::zeros(canny_output.size(), CV_8UC3);}\]
\[
\text{for (size_t i = 0; i < contours.size(); i++)}\{
 \text{Scalar color = Scalar(rng.uniform(0, 255), rng.uniform(0, 255), rng.uniform(0, 255));}\]
\[
\text{drawContours(drawing, contours, (int)i, color, 2, 8, hierarchy, 0, Point());}\]
\[
\text{circle(drawing, mc[i], 4, color, -1, 8, 0);}\]
\]
\[
\text{// Show in a window}\]
\[
\text{imshow("centroid", drawing);}\]
\[
\text{int c = cvWaitKey(1);} //\text{add}\]
\[
\text{for (size_t i = 0; i < contours.size(); i++)}\{
 \text{Scalar color = Scalar(rng.uniform(0, 255), rng.uniform(0, 255), rng.uniform(0, 255));}\]
\[
\text{drawContours(drawing, contours, (int)i, color, 2, 8, hierarchy, 0, Point());}\]
\[
\text{circle(drawing, mc[i], 4, color, -1, 8, 0);}\]
\]
\]
\[
\text{void RTVectors(Mat rvec, Mat tvec)}\{
\text{const float pi = 3.14;}\]
\[
\text{double y_rot, x_rot, z_rot;}\]
\[
\text{double y_rot_angle, x_rot_angle, z_rot_angle;}\]
\[
\text{Mat R_mat;}\]
\[
\text{cv::Mat_<float> quat(4, 1);}\]
\[
\text{rvec.convertTo(rvec, CV_32F);}\]
\[
\text{tvec.convertTo(tvec, CV_32F);}\]
\[
\text{Rodrigues(rvec, R_mat);}\]
\[
\text{Mat R_inv = R_mat.inv();}\]
Mat P = -R_inv*tvec;//camera origin(cener) position from world coordinate

float* p = (float*)P.data;
float w;
w = sqrt(w);
quat(0, 0) = (R_mat.at<float>(2, 1) - R_mat.at<float>(1, 2)) / (w * 2);
quat(1, 0) = (R_mat.at<float>(0, 2) - R_mat.at<float>(2, 0)) / (w * 2);
quat(2, 0) = (R_mat.at<float>(1, 0) - R_mat.at<float>(0, 1)) / (w * 2);
quat(3, 0) = w / 2;
euler = rotationMatrixToEulerAngles(R_mat);//get Euler angle
cout << fixed << setprecision(2) << "Euler- Roll:  " << euler[0] * Rad2Deg << "  , Pitch:" << euler[1] * Rad2Deg << "  , Yaw: " << euler[2] * Rad2Deg << endl;//camera position
cout << "R=\[" << rvec.at<float>(0, 0) << "," << rvec.at<float>(1, 0) << "," << rvec.at<float>(2, 0) << "]," << "T=\[" << tvec.at<float>(0, 0) << "," << tvec.at<float>(1, 0) << "," << tvec.at<float>(2, 0) << "]" << endl;

// Checks if a matrix is a valid rotation matrix.
bool isRotationMatrix(Mat &R)
{
  Mat Rt;
  transpose(R, Rt);
  Mat shouldBeIdentity = Rt * R;
  Mat I = Mat::eye(3, 3, shouldBeIdentity.type());
  return norm(I, shouldBeIdentity) < 1e-6;
}

// Calculates rotation matrix to euler angles
// The result is the same as MATLAB except the order
// of the euler angles (x and z are swapped).
Vec3f rotationMatrixToEulerAngles(Mat &R)
{
  R.convertTo(R, CV_64F);//works
  assert(isRotationMatrix(R));
}
float sy = sqrt(R.at<double>(0, 0) * R.at<double>(0, 0) + R.at<double>(1, 0) * R.at<double>(1, 0));
bool singular = sy < 1e-6; // to check if it is in singularity
float x, y, z;
if (!singular)
{
    x = atan2(R.at<double>(2, 1), R.at<double>(2, 2))*Rad2Deg; // roll
    y = atan2(-R.at<double>(2, 0), sy)*Rad2Deg; // pitch
    z = atan2(R.at<double>(1, 0), R.at<double>(0, 0))*Rad2Deg; // yaw
}
else
{
    x = atan2(-R.at<double>(1, 2), R.at<double>(1, 1))*Rad2Deg;
    y = atan2(-R.at<double>(2, 0), sy)*Rad2Deg;
    z = 0 * Rad2Deg;
}
return Vec3f(x, y, z);