Real-time Vision-based 6-DoF Pose Detector for Robotic Assembly Systems

Test Plan

Group 45 – Team Four Explore

List of your Names:

Name	Position	Email	Phone
Daniel Leong	Team Leader	102589132@student.swin.edu.au	0431691326
Parth Patel	Coordinator	101604584@student.swin.edu.au	
Preenit Kshirsagar	Support Manager	102486657@student.swin.edu.au	0404655250
Dylan Were	Monitor	102102160@student.swin.edu.au	

EAT40003, FINAL YEAR RESEARCH & DEVELOPMENT PROJECT 1, SEMESTER 1 2022

Table of Contents

1 Resources Required

- PC/Laptop
- Built-in/External Camera
- IDE to initially run code
- Users to perform actions
- Measuring device for pose
- Deployment Device (Microcontroller)

2 Features to be Tested

- Detection of Marker/s
- Precision of Marker/s
- Efficiency of software

3 Functional Test Cases

Taken from Software Requirements Specifications.

Test Case/s:	Steps	Expected Results
Detect objects using camera		
1) User display marker/s to camera connected to device.	1. User turns on device 2. System automatically tries to establish a connection with the connected device 3. User will display marker/s to the camera's field of view.	Device will display current pose information to the users via a user interface.
Capture numerical data from object and Perform algorithms		
1) User wants to know the current pose of the displayed marker/s	1. User will show marker/s to the camera. 2. User will move marker/s around. 3. System will display pose data.	Device will display current pose information to the users via a user interface.
Execute on microcontroller based on internal calculations		
1) User operates on device that is deployed on microcontroller	1. User will turn on the microcontroller. 2. User will display marker/s to the camera's field of view. 3. Microcontroller will respond in a programmed manner to the pose of the markers.	Software will operate the same regardless of device used. Ideally, the mechanism should mimic the movement of the maker.
View and edit live functional parameters of system		
1) User modifies the parameters of the system	1. User modifies parameters using the user interface. These	Software will adapt to modified parameters and operate according to the new

	are camera selection and calibration parameters, selection of marker dictionary and specification of marker/checkerboard dimensions. 2. User will display marker/s to the camera's field of view.	parameters for greater pose estimation precision. Improving overall performance and flexibility.

4 Non-Functional Test Cases

Taken from Software Requirements Specifications.

4.1 Functional Suitability:

Requirements	Test Case/s	Result
The accuracy of the 6-DoF data should be within the allowed error range of +- 1 millimeter for all x, y, z coordinates and the pitch, yaw, and roll.	User places a marker at different fixed positions and orientations.	Placed markers will produce near identical results when placed in the same location multiple times.
System should provide 100% capacity to obtain and display the relevant 6-DoF data and send parameters to the system for both eye-in-hand and eye-to-hand variations.	User places a marker at all different positions and orientations possible.	As Requirement states, data should be available for all times in any orientation.

4.2 Performance Efficiency:

Requirements	Test Case/s	Result
Detection and response are almost instantaneous to the human eye, at computational level processing delay should be less than 800 milliseconds.	User moves a marker systematically within the camera field of view.	Pose value updates dynamically with little to no delay as observed by the human eye.
Under normal operation, no more than 50% of a system's full memory should be utilized.	User moves a marker systematically within the camera field of view.	Observed memory should be less than 50% of capacity.

4.3 Compatibility:

Requirements	Test Case/s	Result
System must be able to successfully transfer required information to relevant software and hardware components working in the	User swaps the camera to a different one. Application is transferred to different systems with	System will successfully operate the exact same regardless of the varied hardware components.

same environment with 100% compatibility and efficiency to avoid data loss.	required runtime environment installed Various microcontrollers can be controlled by minimal adjustment needed.	
More than 50% of the system's software/hardware components should be able to function independently.	Running the system without the integration of a microcontroller.	As Requirement states.

4.4 Usability:

Requirements	Test Case/s	Result
Use of universal codes, symbols, characters and adhere to general language/cognitive standards so that users can recognize them in less than 5 seconds.	N/A	As Requirement states.
Users should be able to learn to operate the system in less than 1 day.	Users unfamiliar to the software, operate the software.	Users are able to briefly describe the operations of the system.
Allow users to make less than one error per 10 operations	Users operates with the system.	As Requirement states.
User interface is practically laid out and easy to navigate.	Users should be able to navigate the system easily.	As Requirement states.
Can be operated by colour-blind people as well as deaf people.	User wearing noise-cancelling headphones, operates the system	Users can operate the system proficiently.

4.5 Reliability:

Requirements	Test Case/s	Result
System must be available 100% or minimum 99% of the time when in operation.	System is left on for a period, the user will sparingly display a marker.	As Requirement states.
Error rate < 1%.	N/A	As Requirement states.
Expected time between failures (downtime) is limited to 5-7 minutes.	Leave the system on for a long period of time.	As Requirement states.

4.6 Maintainability:

Requirements	Test Case/s	Result
Testing satisfies >90% of the test criteria.	N/A	As Requirement states.
No more than 18 days needed to implement, test, and deploy the modification.	Transfer to another system and attach to a different microcontroller.	As Requirement states.

More than 50% of the system's software/hardware components should be able to function independently.	N/A	As Requirement states.

5 Strategy

The Project Plan briefly iterated over the critical success factors which put an indirect focus on the testing strategy that will be taken forward. To ensure the success of most software systems, multi-billion dollar companies have adopted the four-level testing strategy, thus, we have decided too, as follows:

The hierarchy is default in the order which testing should be carried out:

5.1.1 Unit Testing:

Due to the vast and multi-disciplined nature of the system, it is crucial to test incremental development of particular units of functionality independently to allow detection of errors/bugs in the vulnerable initial stages of the development process. To implement this practice, unit tests will be designed to test small fragments of code/functionality.

5.1.2 Integration Testing:

The logical next step following Unit testing, is integration testing. This is a point in the testing process where after ensuring individual components function as intended, the software components should be collaborative, thus should be tested as a cluster. This will be accomplished by testing various components of the systems by designing effective test modules.

5.1.3 System Testing:

As the name suggests, system testing is the testing of the nearly complete integrated product, in other words, to ensure that the product satisfies majority/all of the requirements. To reiterate, requirements consist of functional and non-functional requirements, thus, while the functional requirements might have been met in previous stages, it is essential to ensure test cases are designed to test non-functional test cases.

5.1.4 Acceptance Testing:

This testing stage is mainly to ensure that the business requirements have been catered for by the software product. This will involve client collaboration with the finalized product to evaluate client/stakeholder satisfaction.

5.2 Roles and Responsibilities

Below are the stated roles and responsibilities in-line with the testing strategies stated above:

5.2.1 Unit testing:

As unit testing deals with small fragments of the components/codes, it is a good practice that the team members/developers working on the fragments actively test their creation as an ongoing process by designing relevant test cases deemed suitable by themselves throughout the process.

5.2.2 Integration testing:

As integration testing is concerned with the unifying functionality of all software modules, this step can be executed with the input and presence of more than one developer and/or by the managing body and/or testing team (if applicable).

5.2.3 System testing:

It is preferred that this type of testing be carried with personnel specially appointed for testing (users) in the presence of the team leader/manager. The suitability of the testing group will depend on their amount of experience with software systems.

5.2.4 Acceptance testing:

At last, this type of testing should be conducted with the presence of the leader/manager and the testing personnel mentioned in the previous step. However, this time, the whole process will be under the eyes of the client which is necessary for them to evaluate the system with them possibly having no experience in developing software systems.

5.3 Test Deliverables

- Test Plan
- Test Cases
- Test Reports
- Usability Test Guide

5.4 Schedule

System Features	Estimated Testing Period
Object detection	Semester 2 (early)
Pose detection	Semester 2 (early)
3D location with rotation output	Semester 2 (mid)
Operation in mobile setting (eye-in-hand, eye-to-hand)	Semester 2 (mid-end)
User Interface	Semester 2 (end)

5.5 Risk and Contingency with Testing

Risk	Contingency
Team members leaving/unable to make substantial contributions	Alteration of scope if required/distribution of tasks to other available team members
Scope variations	Updating and recording changes from client. Harnessing agile development methodologies.
Inadequate risk management	Perform a risk analysis and keep an updated backlog.

Incompatible devices (camera, microcontroller)	Research various devices with compatible devices. Find a universal device that can handle different connections
Client/stakeholder engagement	Frequent user acceptance testing along with other tests, such as unit testing
Time management with development and testing	Run Tests alongside software development to reduce excess time.

5.6 Testing Tasks

- Effective test cases will be derived from real life testing scenarios in the intended operational environment of the product.
- According to testing scenarios, unique test cases will be designed to test particular functions/features of the system.
- Resource analysis for testing will be conducted.
- Test results along with errors/bugs observers will be properly documented along with changes made to fix them.

6 Pass/Fail Criteria

6.1 Product Level

6.1.1 Initial Stage of Product Development

Testing Criteria	Pass Criteria	Fail Criteria
Object Detection: **The system should be able to detect a moving object in its' field of vision**	90% Objects Detected with minimal interference	Unable to Detect simplest of shapes or moving tags
Pose Detection: **The system should be able to detect the pose of the tags and illustrate axis upon it**	95% Tags should have an axis on them in field of camera and detect any changes in pose of the tags	Unable to determine the changing pose of the tags
3D location with rotation output: **The system is able to provide an output of the pose of the tag in (x, y, z, pitch, yaw, roll)**	System is able to provide the requested output in relative quick cycles (10 - 100 ms)	It can provide some output but not all. It is providing all the outputs but there is lot of latency making the system inefficient

6.1.2 Middle Stage of Product Development

Testing Criteria	Pass Criteria	Fail Criteria
Operation in Mobile Setting: **The system is deployed on a microcontroller and functions normally**	System is functional within the microcontroller.	System does not work within the microcontroller.
Eye-in-hand: **The system should be able to be mounted on a robot and work properly**	This system needs to be able to calibrated on the fly to be able to match the movement skill.	This system is unable to be used for mounted operation

Eye-to-hand: The system should be able to use normally at a fixed position		

6.1.3 Final Stage of Product Development

Testing Criteria	Pass Criteria	Fail Criteria
Speed of the system: The system is able to run fast and efficiently	The system is able to run a relative cycle at 10 to 40 ms	The system is unable to run at the required speed
Accuracy of the system: The system is able to be accurate	The system is able to be accurate to 1 to 4 mm	The system is not accurate to the standards
Interface for operations: The interface is easy to use and integrate with other systems	The interface is user-friendly and is able to integrate with client specified robotic arm	The system does not have good interface

6.2 Suspension Criteria and Resumption requirements

6.2.1 Suspension Criteria:

- The system is found to have a critical operational error.
- The client requirement has changed for the proposed module
- The testing has found irregular issues in operational and results.
- The system is not suitable for the final product

6.2.2 Resumption Requirements

- All critical issues have been resolved.
- The system matches client's vision for the system
- The proposed system is suitable for the final product.
- The client has verified the use of the selected design and system.

7 Test Results

7.1 Hardware Used

	Member Conducting	Hardware Used
1	Daniel Leong	Raspberry Pi 4 Model B, 8GB Raspberry Pi Camera Module 2, Logitech StreamCam
2	Daniel Leong	Lenovo ThinkPad 13 Type 20GK Logitech StreamCam
3	Preenit Kshirsagar	Raspberry Pi 4 Model B, 8GB Raspberry Pi Camera Module 2
4	Preenit Kshirsagar	Dell Inspiron 15 5000

7.2 Functional Test Cases

	Detect Objects using Camera	Capture data from object and Perform algorithms	Execute on microcontroller based on internal calculations	View and edit live functional parameters of system
1	True	True	True	True (not live)
2	True	True	True	True (not live)
3	True	True	True	True (not live)
4	True	True	True	True (not live)

7.2.1 Camera Selection

Video Reference – https://youtu.be/8f4Gk_oZHdY, https://youtu.be/1HWdiYyCuaU

Following from the core selling properties of the application, it was designed to accommodate various hardware components, the most important being cameras. Some users of the application may need to attach cameras with custom specifications and/or may have multiple cameras attached and want to choose a specific source camera. This provides successful evidence that the application is able to detect all cameras attached to it and provide users with an option which one to choose. Once the camera is selected, its specifications will be used for further functionality (frame width, frame height, fps etc). Below figure shows evidence of the application running the camera select feature:

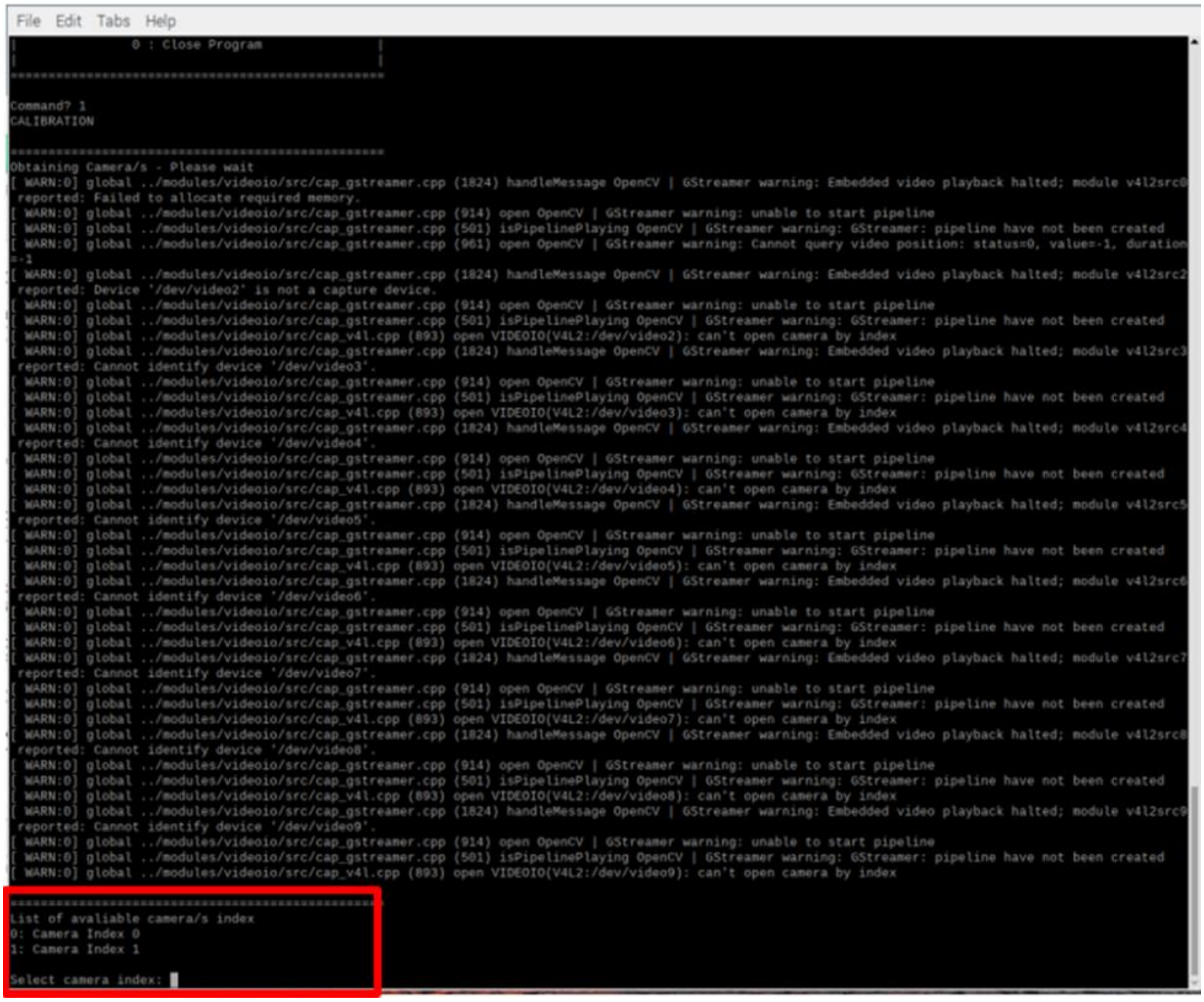

Figure 1 - Checking Available Cameras (2 connected in this case)

Referring to figure 14, the test was conducted (on raspberry pi computer) by connecting two different cameras to the system:

1) Raspberry Pi Camera Module 2
2) Logitech USB camera

When the camera selection function is executed, the program scans for all the available cameras and correctly lists the number of cameras available for selection (see red box).

The outcome of the test was successful. Refer to the video for full test coverage.

7.2.2 Calibration

Video Reference – https://youtu.be/0TkNknL9HQ8, https://youtu.be/Arq2QwTZxfw

To be able to detect the pose of an object through a camera, it is essential to consider the issues with using hardware, in particular the camera lens, this has been discussed in the section architecture design. The purpose of the functional test is to verify that we are able to calibrate different cameras based on selection however many times we like repeatedly. As discussed, the camera has intrinsic and extrinsic properties which affect the image in terms of stretching, radial/tangential distortion etc which incorrectly modifies the image in the camera and thus we receive a poor image which negatively impacts the pose detection algorithm. The purpose of calibration is to capture these properties of the camera (camera matrix and distortion matrix) and use them to reverse the effects of the camera on the image, so we get a perfect image. The following figure 15 shows evidence of producing the required matrices after doing the calibration using a raspberry pi camera module 2, note that different cameras will have different matrices:

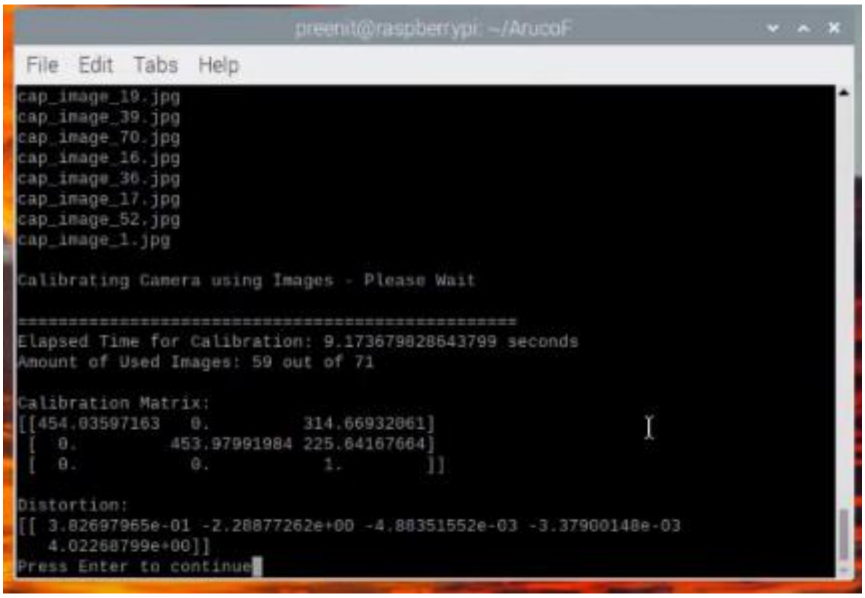

Figure 2 - Creating of Relevant Matrices from Calibration

As can be seen, the test was successful. Refer to the video for full test coverage.

7.2.3 Generate Tags

Video Reference – https://youtu.be/H31N1WGUrUA, https://youtu.be/kk40EY_mUFQ

To provide a modular design for the application as well as a lower dependency of external applications required, the system has been equipped with its own feature to generate the markers that can be used for detection. This will allow the user to select markers to create from a range ArUco dictionaries and save them in a file directory, these markers can be then used to detect the pose of an object on which these markers are attached to. The following figure 3 shows the creation of markers from different dictionaries:

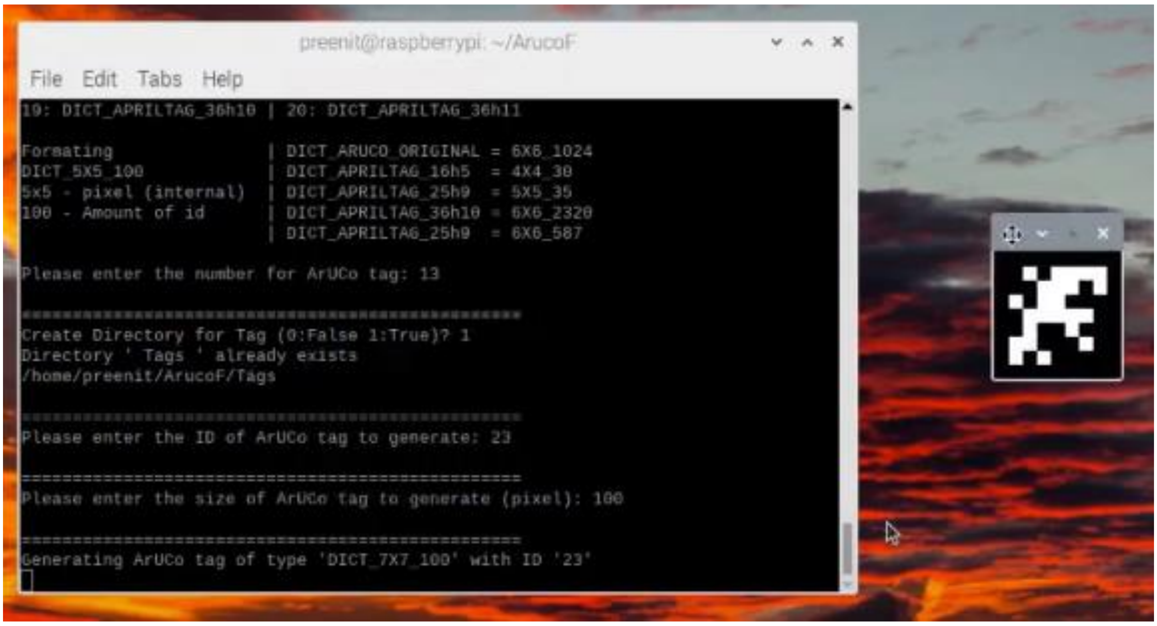

Figure 3 - Generating Tag

As can be seen from above figure 3, a marker from dictionary 13 (DICT_7x7_100) is generated by specifying the size to allows for further flexibility as well as greater accuracy when detecting the pose due to the fact that the user will know the exact size of the marker which they can provide to the Pose Detector module increasing the accuracy of the estimation.

The outcome of this test was successful. Refer to the video for the test.

7.2.4 Detect and Follow Pose

Video References – https://youtu.be/uSWewK_Ha7s, https://youtu.be/sgTzLg3-OME, https://youtu.be/4sI-ZR5JbAE

With the use of the camera, information collected during calibration and a tag generated to detect, we now test the actual algorithm for pose detection. This test aims to verify if the application can detect the maker generated using the previous modules and verifies the detection algorithm works as we are able to get accurate results in terms of x, y, z coordinates and pitch, roll and yaw Euler angles. The figure 4 below shows the evidence that we have selected a camera to detect a pose from, the calibration exists (both tests completed previously):

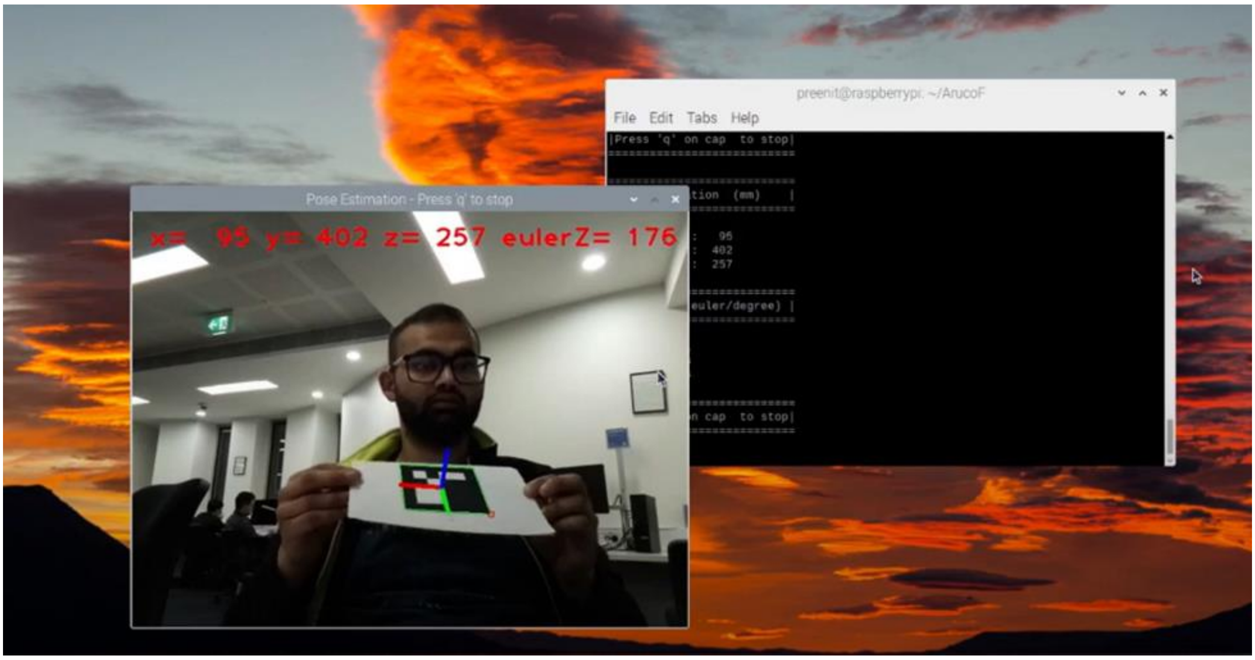

Figure 4 - Detection of Pose

Referring to Figure 17, it can be verified that we are able to detect the pose of the marker accurately, this verifies that we have access to our selected camera, the camera and distortion matrix are calculated as we do not have a distorted image (image is clear). The algorithm discussed in the architecture design is implemented correctly as can be seen from the pose detection values (in red) at the top of the window. Refer to the video for the full test.

7.2.5 Execution with Simulation

Video Reference – https://drive.google.com/file/d/1WeoGCFUby_Qc9EIlt4B97O0Z-4lvVXkj/view?usp=sharing

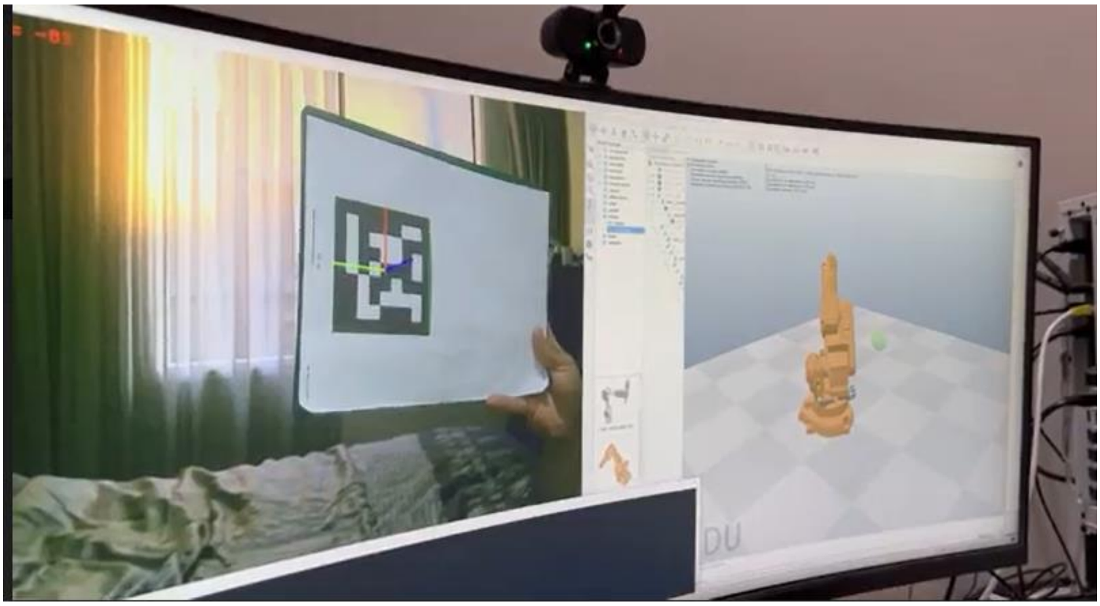

Figure 5 - Pose Detection with Simulation

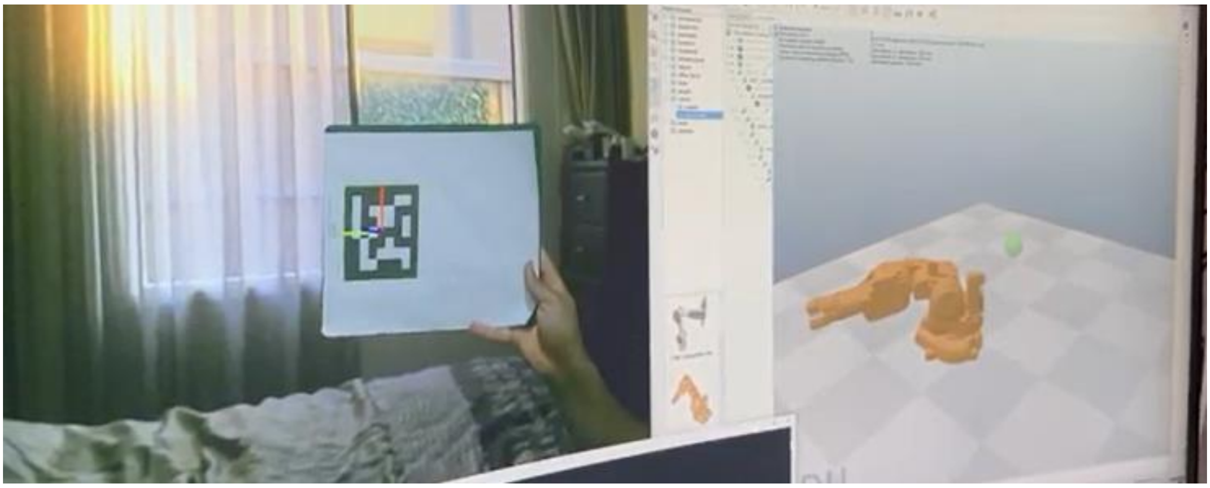

Figure 6 - Another Pose Detection with Simulation

To demonstrate the pose data that has been calculated can be put to use, we have developed a robotic arm simulation which mimics a robotic arm in the real world, the 'head' of the arm moves to align itself with the center of the marker by using the output pose values (x, y, z, pitch, yaw and roll) to reference its position in 3D space.

7.3 Non-Functional Test Cases

7.3.1 Functional Suitability:

	The accuracy of the 6-DoF data should be within the allowed error range of +- 1 millimeter for all x, y, z coordinates and the pitch, yaw, and roll.	**System should provide 100% capacity to obtain and display the relevant 6-DoF data and send parameters to the system for both eye-in-hand and eye-to-hand variations.**
1	True	True
2	True	True
3	True	True
4	True	True

Since the system is used to detect pose estimation values, there is no real way to test the accuracy of the system without having another system to compare the values with. Therefore, we are limited here in terms of the hardware available to test the system.

There exists however a mathematical formula which, provided with x, y coordinates gives us the 'ideal' z coordinate in mathematical terms without actually reading the z coordinate from the lens. This methodology is not 100% reliable due to the fact that we are still depending on reading the x and y coordinates from the camera to calculate z. However, the deviation from both the scenarios will tell us whether there is some discrepancy in the readings and thus this will prove the error rate.

The mathematical formula below has been embedded into the PoseDetector module to calculator z based on x and y:

$\text{Distance}(z) = \text{sqrt}(x^2 + y^2 + z^2)$

Where x, y and z are the coordinates obtained from the translation vectors.

Below is the code fragment that was added in python to get distance using numpy's square root (sqrt) function.

```
#Translation (mm)
x = realworld_tvec[0]
y = realworld_tvec[1]
z = realworld_tvec[2]

#greater accuracy distance calc formula
distance = np.sqrt(x ** 2 + y ** 2 + z ** 2);
```

Figure 7 - Getting Mathematical Distance

During the test, a ruler was used (millimeter accuracy) to place the marker at specific length from the camera of the system. The marker was placed 27 centimeters (270 millimeters) from the camera and the display observed for the z distance.

Video Reference – https://youtu.be/tuX77j06DSg

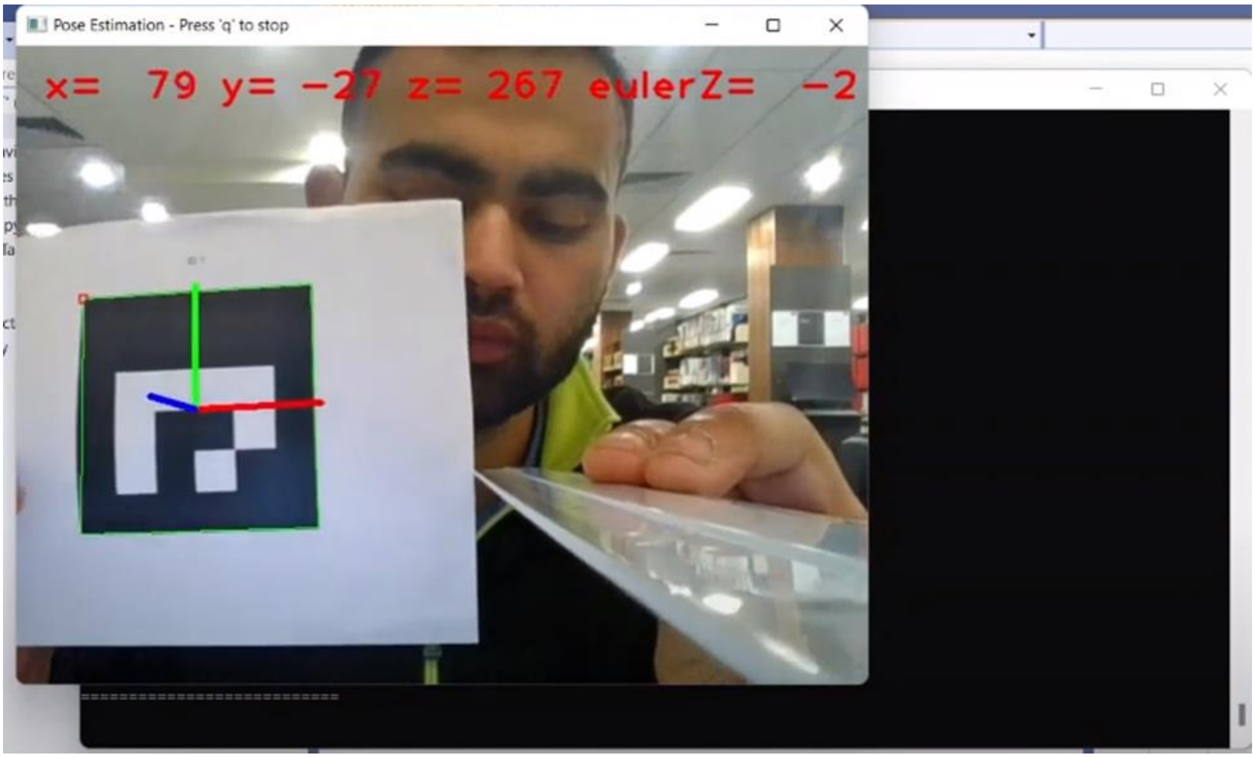

Figure 8 - Distance at 270 mm without using Formula

Video Reference – https://youtu.be/0yxqPmT83dc

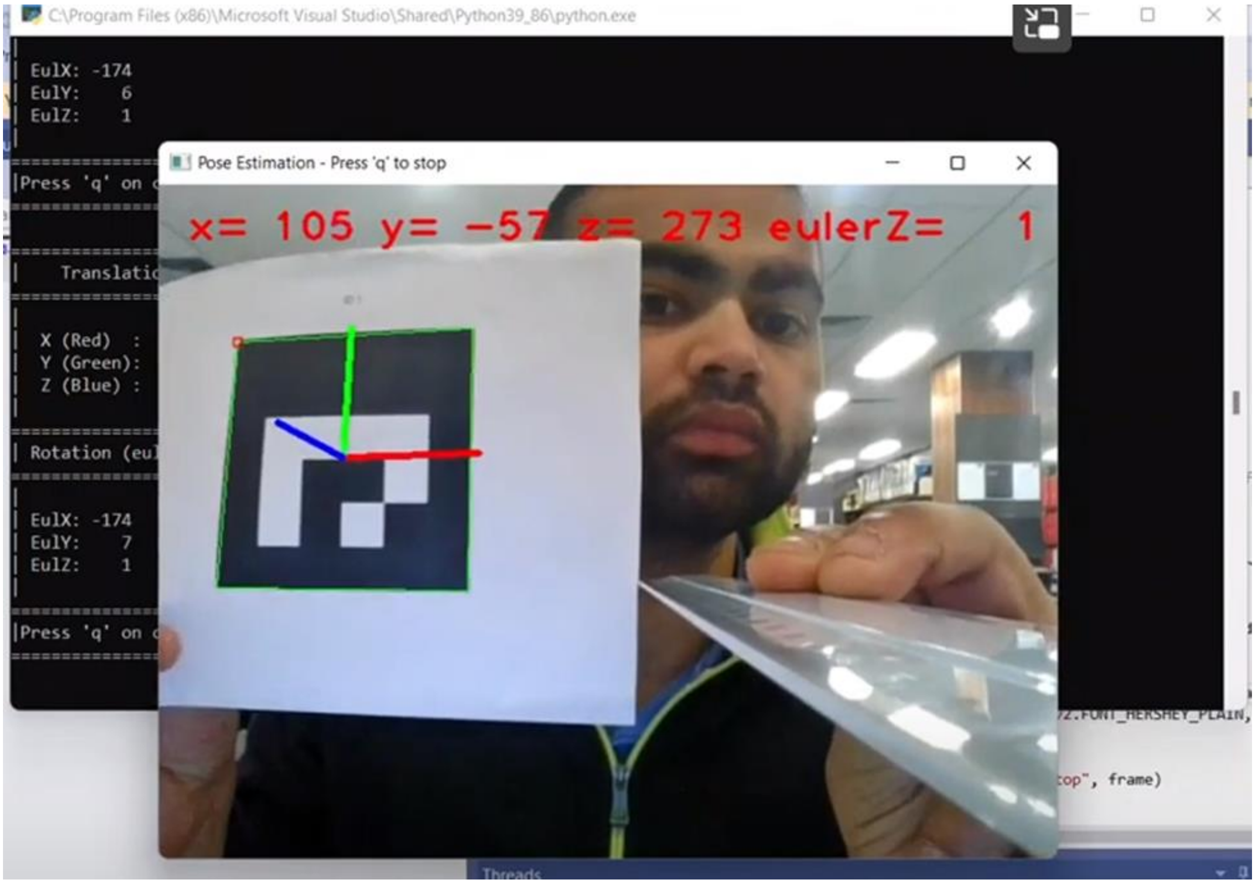

Figure 9 - Distance at 270 mm with using Formula

Therefore, in conclusion as can be seen from the measurements, the distance without using the algorithm was short by 3 mm and the distance obtained by using the algorithm overshot by 3 mm.

Therefore, a rule can be formed where the program takes the average of the two numbers and then outputs that value as the z coordinate.

This ensures that the error remains within the acceptable range of +-1 mm.

Refer to the videos for full test coverage.

7.3.2 Performance Efficiency:

	Detection and response are almost instantaneous to the human eye, at computational level processing delay should be less than 800 milliseconds.	Under normal operation, no more than 50% of a system's full memory should be utilized.
1	True	True
2	True	True (max ~40%)
3	True	True
4	True	True

7.3.2.1 Process Delay

Video Reference – https://youtu.be/n6LNN677em4

For ideal real-time pose detection, it is important that the processing delay of the application from detecting the object, calculating its pose in 3D and outputting the result must be minimal as any significant delays defies the purpose of the application for obvious reasons. The processing speed depends majorly on the hardware that the application is run on and minorly on the software design. The software design has been designed with modularisation techniques (discussed in design and architecture) to allow for efficient code and optimal code refactoring. Therefore, the requirement was reasonable to say that the processing delay of each frame (mostly hardware dependent) should be no more than 800 milliseconds at which point it can be detected by the naked eye. This test was conducted on a raspberry pi 4B with 8 Gb of RAM. The following figures 5 and 6 show the code embedded into the application source code for testing the time delay between capturing a frame and outputting pose values:

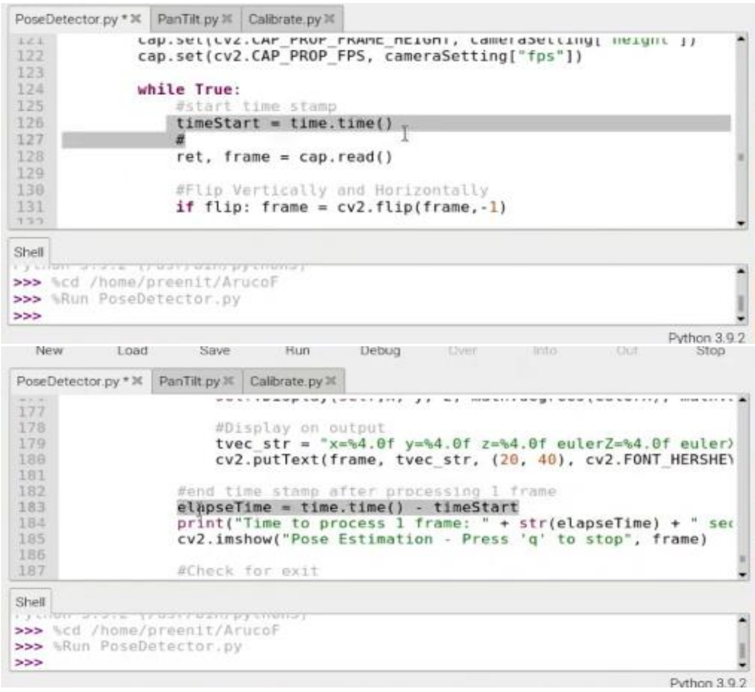

Figure 10 - Code Addition to add Timestamp

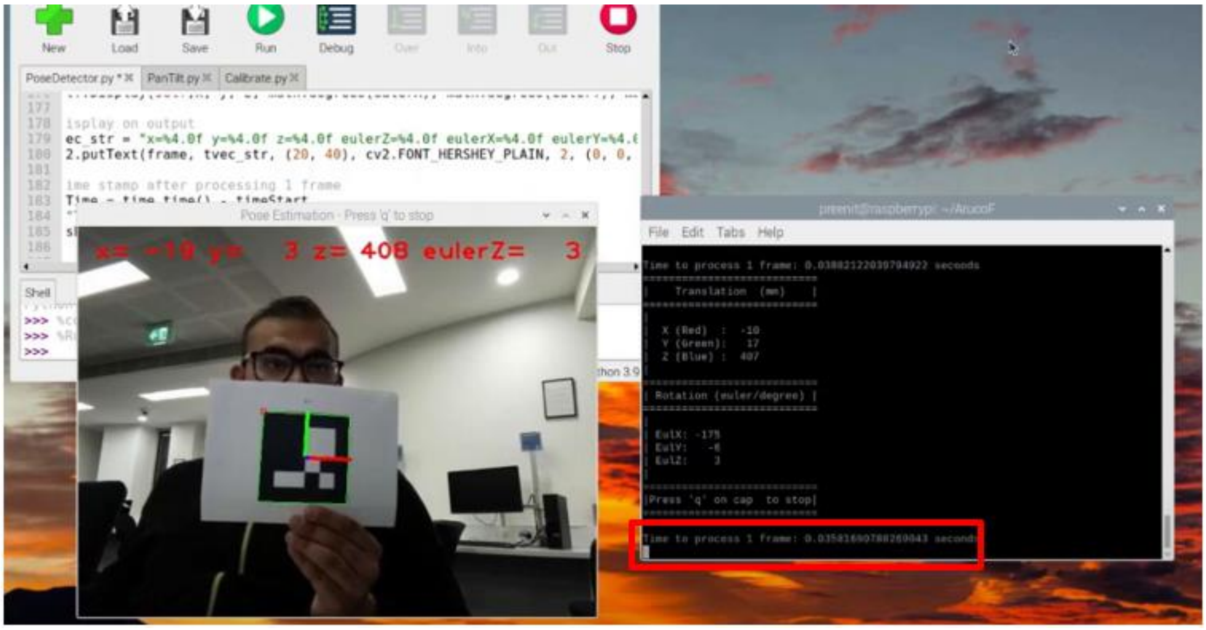

Figure 11 - Processing Delay Output

As can be seen in figure 11 above, the time taken to process one frame is around ~0.036 seconds, roughly equal to 36 milliseconds.

Therefore, we can confirm that this is less than 800 milliseconds and hence the test was successful. Refer to the video for full test coverage.

7.3.2.2 Memory Usage

The following installation package was needed to be installed which allowed us to view hardware utilization using htop:

sudo apt-get install htop

As briefly mentioned in the previous test that the processing time was a critical aspect to efficient pose estimation and hence all factors that affect the speed of execution whether it be software or hardware need to be thoroughly tested to ensure that the memory being occupied by the any functionality in the program does not exceed 50%, this is based on the trend observed by team members of FourExplore based on their experience with computer systems. Generally, the performance of a computer is seen to decline as more and more memory is used. CPU utilization is also another metric that is considered in this test to verify that only the program that is executing is utilizing the majority of the CPU and not any other program that does not need to be run. The final metric that will be monitored is RES, this tells us basically how much of the programs memory is kept in the RAM which is arguably a better measure for memory utilization as modern computer systems make use of concepts such as spatial locality to store most of the program memory closer to the CPU (cache, RAM).

The following will test each functionality and attempt to capture the related metrics:

Usage during Camera Select

Video Reference – https://youtu.be/RG6srBUkX-g

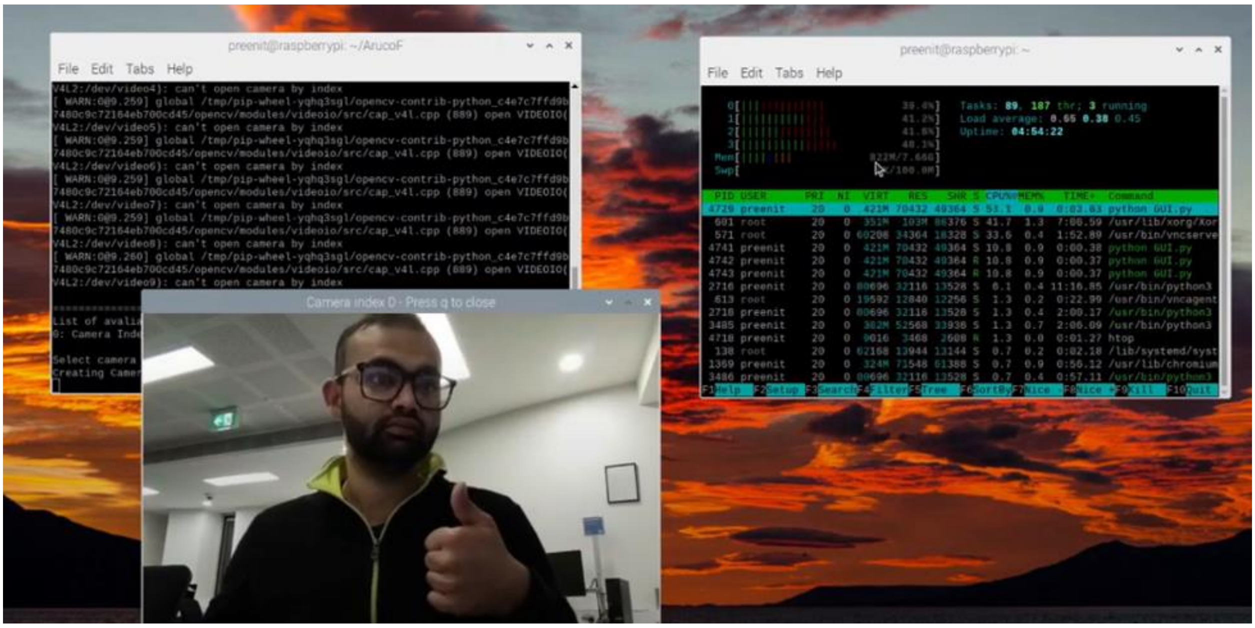

Figure 12 - RES, CPU and MEM statistics

It should be noted that the RAM size for the raspberry pi 4B under test is 8 gb.

As can be observed from running the camera preview, the RES is roughly 70432 which means that the task that is executing (camera select) requires around 70 Megabytes of space in the memory, this is significantly less than 50% of the total memory size hence we have passed the test in this case/

The %CPU represents the percentage of basically how much CPU time the program that is executing takes. Here is a simple diagram to visualize this parameter:

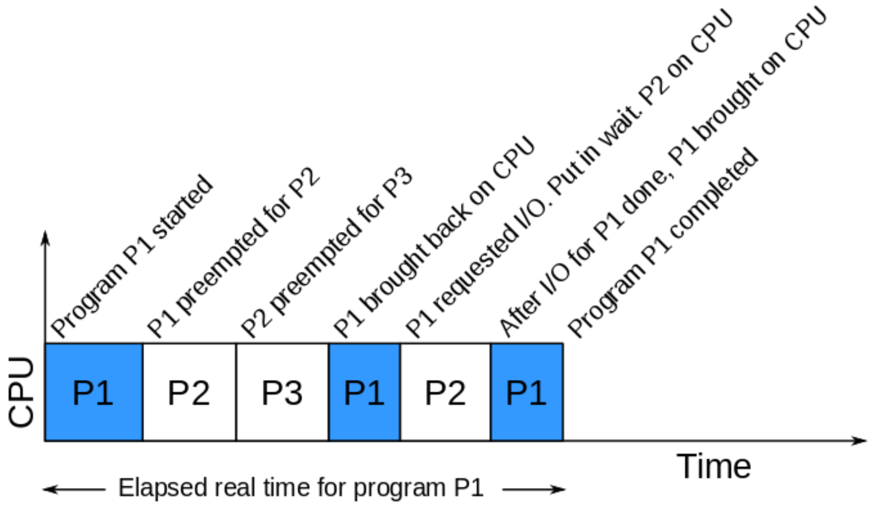

Figure 13 - Representation of CPU time (Wikipedia Contributors 2021)

As can be seen, the more dramatic the changes in between executions, the harder the and longer the CPU is working to accommodate for the task.

For the camera select scenario, the jump between root and preenit is around 10, thus conveying that the process is not putting load on the CPU which is ideal. Hence this test for this metric has also passed.

The %MEM is simply the percentage of memory that has been occupied by the program (out of 8 gb). As seen in the figure 7, only 0.9% of memory is being occupied by the current program hence this test also passes for this metric.

Therefore, all our tests are successful for the camera select feature. Refer to the video for real-time full test coverage.

Usage during Calibrate

Video Reference – https://youtu.be/BMLuGATUxpI

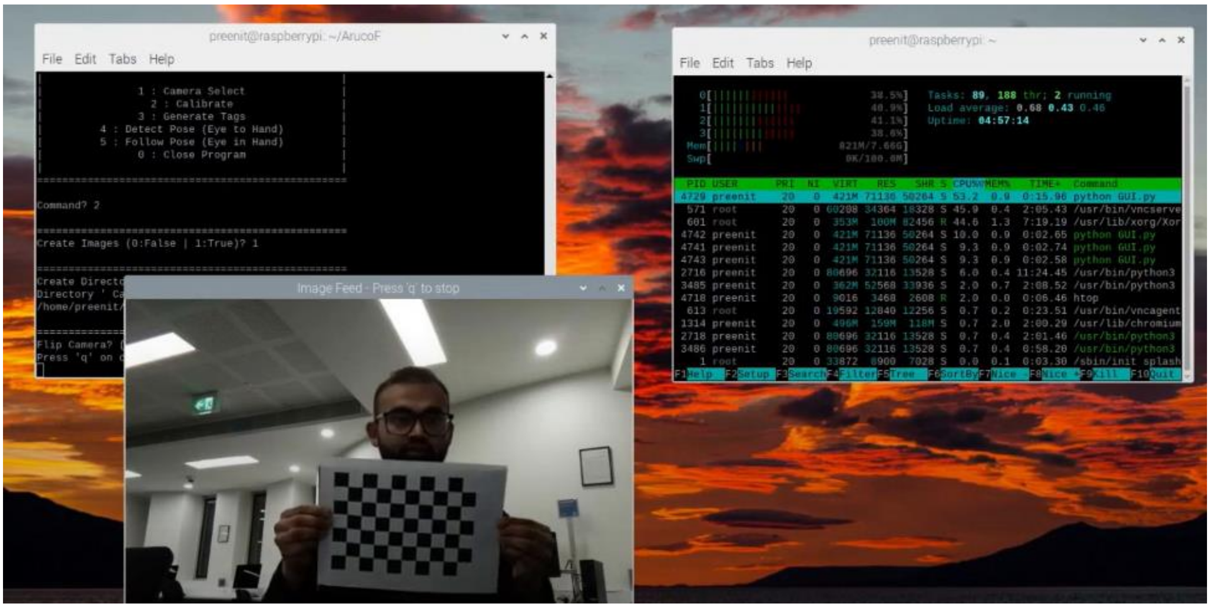

Figure 14 - Metrics when Capturing Images of Chess Board

As seen in Figure 14, the RES value is roughly 71 mb, meaning that the program running in the screenshot above is roughly occupying 71 mb of the memory. Therefore, this is less than 50% of total available memory, thus we have passed this test for this functionality.

There are no surprisingly large jumps in CPU% hence conveying that the CPU is working efficiently without much overhead. This test also passes for this instance.

The MEM% is roughly 0.9% meaning that during this action, only 0.9% of the memory is being utilized. This is significantly less than 50% of total memory (8 gb). Hence this test has also passed for this instance.

Refer to the video for real-time full test coverage.

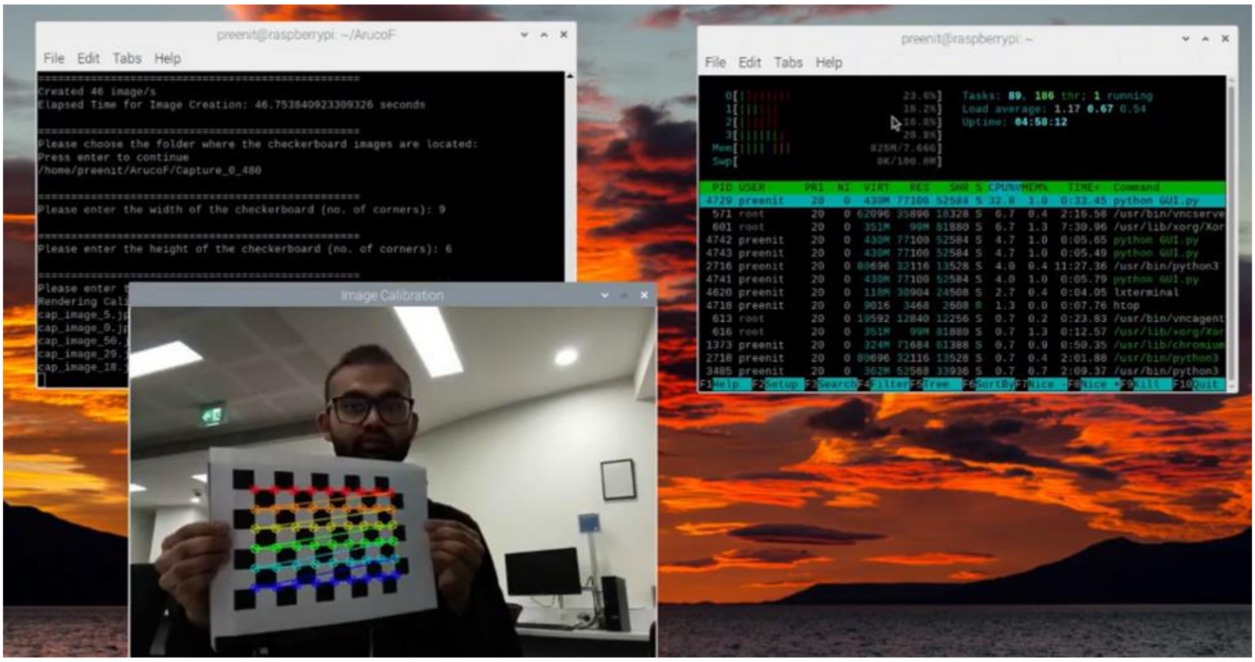

Figure 15 - Metrics when Performing Calibration on Images

Referring to figure 15 above:

- RES: slight hike to 78 mb of memory being occupied by the current program. This is still less than 50% of total memory and thus the test has passed.
- CPU%: here we are also able to see that the current running process is occupying more of the CPU time than for previous processes. However, the program is still operating efficiently. This test has also passed.
- MEM%: The memory being used here is roughly 1%, which is significantly less than 50% of the total available memory (8 gb). Therefore, this test has also passed.

Refer to the video for real-time full test coverage.

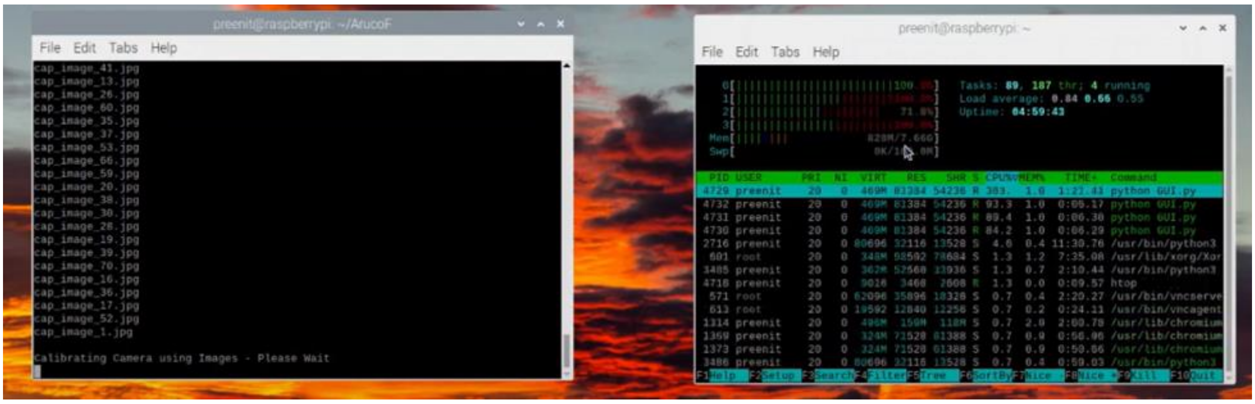

Figure 16 - Metrics when Producing Useful Results from Calibration

Referring to figure 16 above:

- RES: slight hike to 85 mb of memory being occupied by the current program. This is still less than 50% of total memory and thus the test has passed.
- CPU%: here we are also able to see that the current running process is occupying more of the CPU time (greater than double) than for previous processes. However, the program is still operating efficiently. This test has also passed.

- MEM%: The memory being used here is roughly 1.1%, which is still significantly less than 50% of the total available memory (8 gb). Therefore, this test has also passed.

Refer to the video for real-time full test coverage.

Usage during Generate Markers

Video Reference - https://youtu.be/VSQzocP82l8

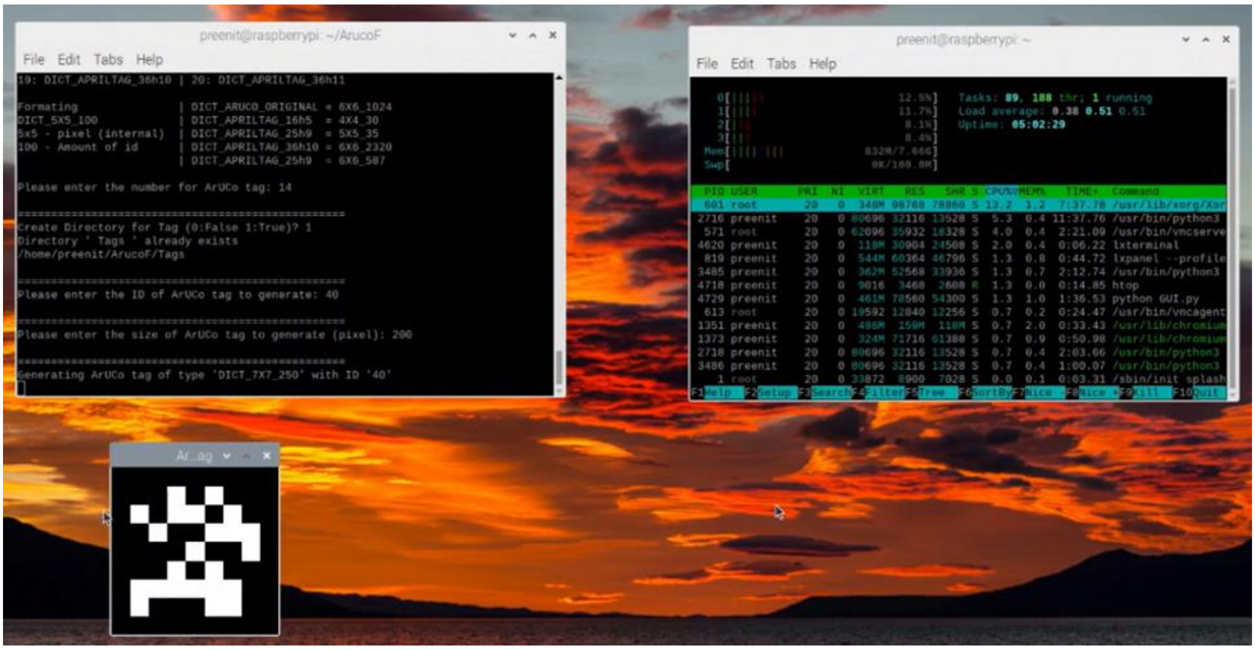

Figure 17 - Metrics Marker Creation

Referring to figure 17 above:

- RES: slight hike to 95 mb of memory being occupied by the current program. This is due to the fact that we are saving the created marker to the file system (memory) This is still less than 50% of total memory and thus the test has passed.
- CPU%: here we are also able to see that the current running process is occupying significantly less of the CPU time as there is not much processing that needs to be done. The program is still operating efficiently. This test has also passed.
- MEM%: The memory being used here is roughly 1.2%, which is still significantly less than 50% of the total available memory (8 gb). Therefore, this test has also passed.

Refer to the video for real-time full test coverage.

Usage during Pose Detection

Video Reference – https://youtu.be/cjKDq_VEvKo

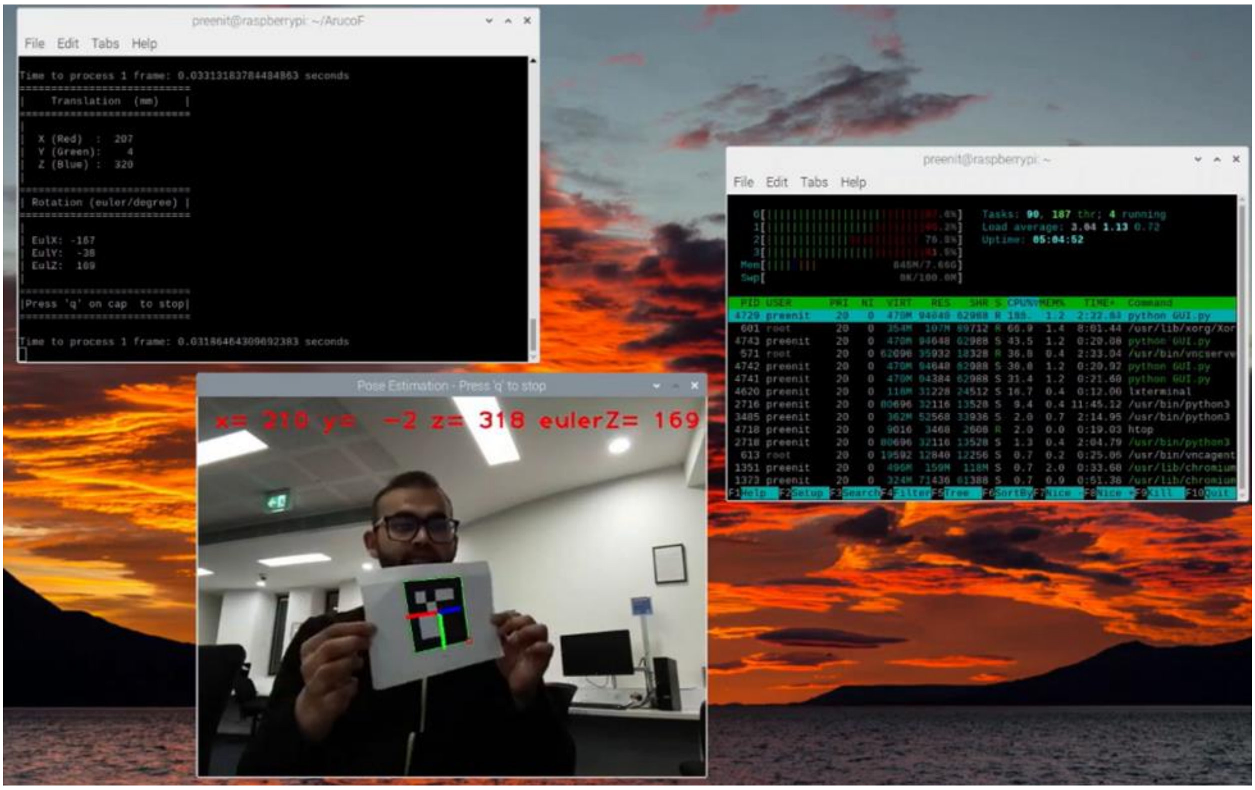

Figure 18 - Metrics Marker Creation

RES: slight hike to 94 mb of memory being occupied by the current program. This is due to the fact that this module makes access to memory for the calibration data and to store the values it has received to perform mathematical calculations on. This is still less than 50% of total memory and thus the test has passed.

CPU%: here we are also able to see that the current running process is occupying significantly greater of the CPU time as there is much processing that needs to be done (calculations using algorithms). The program is still operating efficiently. This test has also passed.

MEM%: The memory being used here is roughly 1.4%, which is still significantly less than 50% of the total available memory (8 gb). Therefore, this test has also passed.

Refer to the video for real-time full test coverage.

7.3.3 Compatibility:

	System must be able to successfully transfer required information to relevant software and hardware components working in the same environment with 100% compatibility and efficiency to avoid data loss.	More than 50% of the system's software/hardware components should be able to function independently.
1	True	True
2	True	True
3	True	True
4	True	True

7.3.4 Usability:

	Use of universal codes, symbols, characters and adhere to general language/cognitive standards so that users can recognize them in less than 5 seconds.	Users should be able to learn to operate the system in less than 1 day.	Allow users to make less than one error per 10 operations	User interface is practically laid out and easy to navigate.	Can be operated by colourblind people as well as deaf people.
1	True	True	True	True	N/A
2	True	True	True	True	N/A
3	True	True	True	True	N/A
4	True	True	True	True	N/A

7.3.5 Reliability:

	System must be available 100% or minimum 99% of the time when in operation.	Error rate < 1%.	Expected time between failures (downtime) is limited to 5-7 minutes.
1	True	True	True (Calibration)
2	True	True	True (Calibration +Camera)
3	True	True	True
4	True	True	True

Refer to the video links from above testing which show that the system was available almost 100% of the time when in operation (there is no evidence of unavailability through the extensive testing).

The downtime when waiting for the system to do some processes for which the speed of the system does not matter still has a downtime no more than 5-7 minutes, examples are waiting for the camera view to open, calculating camera and calibration matrices etc.

7.3.6 Maintainability:

	Testing satisfies >90% of the test criteria.	**No more than 18 days needed to implement, test, and deploy the modification.**	**More than 50% of the system's software/hardware components should be able to function independently.**
1	True	True	True
2	True	True	True
3	True	True	True
4	True	True	True

Through extensive testing, we can conclude that >90% of the tests were successful and the changes that were implemented as a result of the testing (e.g. the user interface needs to be more user friendly) took no more than 18 days to implement, test and deploy.