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

Software Documentation

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 2 2022

Document Change Control

Version	Date	Authors	Summary of Changes
0.8	21/10/22	Preenit Kshirsagar, Parth Patel	Initial commit
0.81	25/10/22	Preenit Kshirsagar	Minor Changes
1.00	26/10/22	Daniel Leong	Formatting and minor edits

Document Sign Off

Name	Position	Signature	Date
Daniel Leong	Team Leader	Daniel	26/10/2022
Parth Patel	Coordinator	PARTH	27/10/2022
Preenit Kshirsagar	Support Manager	Preenit	27/10/2022
Dylan Were	Monitor	Dylan	27/10/2022

Client Sign off

Name	Position	Signature	Date
Jinchuan Zheng	Associate Professor		
Organisation			
Swinburne University of Technology			

Table of Contents

Document Change Control 2
Document Sign Off 2
Client Sign off 2
1 System Documentation 4
1.1 Product Requirement 4
1.1.1 Hardware Requirements 4
1.1.2 Software Requirements 4
1.2 Design and Architecture 4
1.2.1 Overview and Background 4
1.2.2 Architecture and Design Principles 5
1.2.3 User Story Description 7
1.2.4 Solution details 8
1.2.5 Diagrammatic Representation 17
1.3 Source code documentation 18
1.3.1 GUI.py 18
1.3.2 Calibrate.py 23
1.3.3 GenerateTags.py 28
1.3.4 PoseDetector.py 30
1.3.5 Utils.py 34
1.4 Usability Report 35
1.4.1 How to do? 35
1.4.2 Testing Procedure 35
1.4.3 Expanded Details 35
1.5 Testing Documentation 37
1.6 Standards, Reports, and Metrics 38
1.7 Help and Maintenance 39
2 User Documentation 40
3 References 40

1 System Documentation

1.1 Product Requirement

The current system is currently working on a Raspberry Pi 4 with a Raspberry Pi Camera Module. However, the system is designed to work on any older raspberry pi as long as all the supported libraries, drivers and softwares are available on a given device.

The current system runs using a raspberry pi supported camera. However, the system has been modified to use any USB camera with appropriate drivers and resolutions. Following list is the minimum components which are required for a successful replication of the product. The hardware can be sourced from anywhere. The software installation and other details will be covered throughout the document.

1.1.1 Hardware Requirements

- 1x Raspberry Pi (4 GB preferred)
- 1x Raspberry Pi Camera Module
- 1x 16GB SD Card Storage
- 1x Monitor/Screen
- 1x Mini HDMI Cable
- 1x Keyboard
- 1x Mouse

1.1.2 Software Requirements

- Raspberry Pi OS
- OpenCV and Contributors Modules
- Dependency Packages

1.2 Design and Architecture

1.2.1 Overview and Background

To understand the implementation of the solution, it is critical to understand the goals of the project, the problems the software solves and expected achievement outcomes.

Goals and Objectives:

- Determine the pose (6 degrees of freedom (DoF) coordination data - x, y, z, pitch, roll, yaw) of an object using AR tags with 1 millimeter of precision. This pose is achieved on the basis of core mathematical concepts mainly focusing on principles such as translation and rotational vector geometry (He et al. 2020).
- Execute on microcontroller based on 6 degrees of freedom calculated data in Eye-to-hand and eye-in-hand modes to mimic pose of object.
- View and edit live functional parameters of the system via a simple user interface (e.g., tag generation, camera calibration etc) to allow for system usage in a greater field of environments with adaptability to precision requirements of users.

What Problems Does the System Aim to Solve:
The importance of being able to detect free flying objects and get the relative position of it in reference to us in 3D space is a very standard and critical task required by not one specific but in many industrial applications. Some familiar examples include bin picking, autonomous driving, and autopilot systems in aircraft (He et al. 2020). Therefore, by having knowledge of the wide range of environments the application can be used in has led us to research and implement a robust and flexible detection system that can be richly scaled to cater for a range of application environments.

Expected Achievement Outcomes:
In line with the goals and objectives of the system, the fully developed and tested system should be able to achieve the following:

- Real-time 3D positional data accurate to one millimeter using vision-based system API integrating camera and logic.
- Offer flexibility in terms of camera calibration for desired precision including enhanced depth perception and reduced processing times - all of these enhancements eventually require the use of more expensive and highly capable hardware (such as powerful cameras) which the software has been designed to adapt to by merely changing configuration criteria through a user interface.
- Offer modifiability to the program for integration with different microcontrollers using relevant libraries and dependencies as required.

1.2.2 Architecture and Design Principles

Choosing the correct architecture to engineer the 6 degrees of freedom pose plays a critical role in the flexibility and modifiability of the system and thus enhances the life cycle of the product. The solution is structured using a rather modern and popular microservices architecture which helps us implement a concept called legacy system modernization which aids in the long-term evolution of the application (He et al. 2020). This architecture defines that the system is built up of different but most importantly separate modules that perform specific tasks (He et al. 2020), this allows reducing the complexity of the system which not only reduces testing overheads but also provides the flexibility that our application offers mentioned in the above section in terms of application uses. Legacy system modernization simply means that the application is critical in its current application environment but needs an upgrade to cater for evolving technologies without throwing away the core functionality (He et al. 2020).

Refer to the figure 1 below for an example of how microservice architecture differs from traditional architecture with our application specific example:

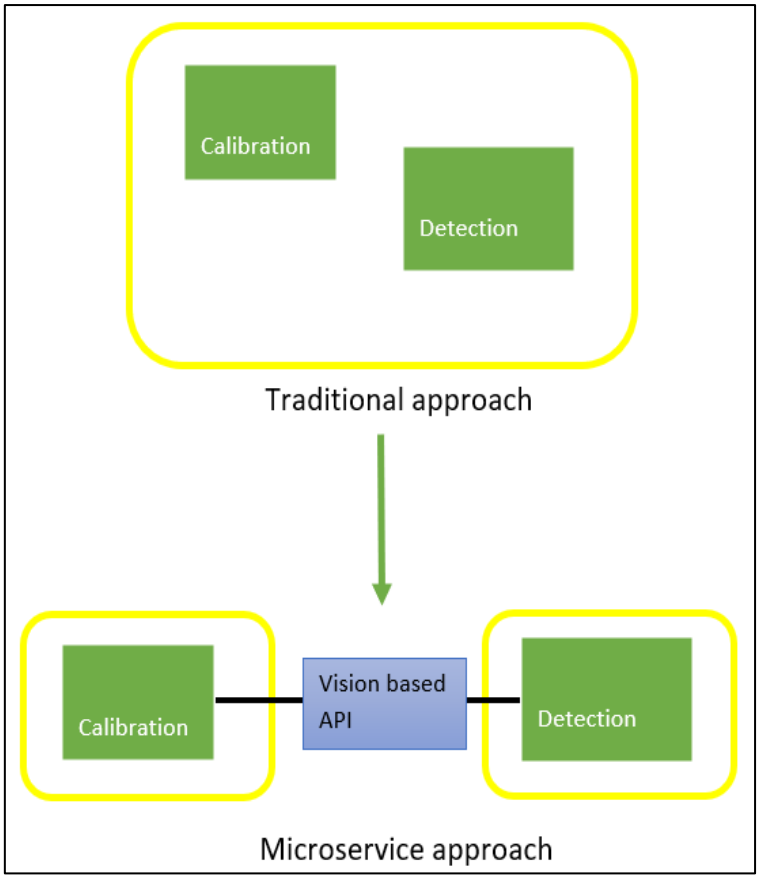

Figure 1 - Example of the use of Microservices Architecture in the design of 6 DoF pose detector

Referring to figure 1, clearly separating the core modules which perform core functionality define the microservices approach. The following benefits of this strategy allow for flexibility, maintainability, and modernization of the application (He et al. 2020):

- **Modules are able to be scaled separately:**
  Calibration can be carried out for various cameras on different operating systems separately according to the precision requirements of the user. Detection functionality can be implemented for varying frame sizes, frames per second etc to scale functionality on-demand allowing application to be used in various environments in which cameras with low performance are used or expensive cameras with frames per second exceeding > 300.

- **Forces DevOps approach:**
  DevOps implements the integration of teams concerned with development and operation of the system, this approach is suitable for smaller teams where human resources are restricted and the delivery phase is generally encountered more frequently. By developing separate modules incrementally allows for frequent testing and deployment hence ideal for the short semester spam to develop the project.

- **Each module can be integrated with most suitable technology:**

Technology here can be referred to in terms of programming language used or hardware technology. To put this into context, camera calibration in its own module can be done using any programming language and this does not have to be the language used to implement the mathematical algorithms in the detection module, again allowing for product flexibility.

In conclusion, by using the microservice architecture, not only is the application able to perform pose detection but also allow for a robust and flexible product for a wide range of industrial usage.

1.2.3 User Story Description

The highlight of the application is that it is not application specific, meaning, there exists no predefined user story for this application mainly due to its wide industry usage and adaptability. However, this being said, here are some common/popular scenarios in which the software can be used:

Obstacle Detection in Autonomous Driving (He et al. 2020):
As vehicles are moving towards autonomous driving technologies, they need to implement systems that can detect objects in their surroundings via sensors such as cameras. The detection of objects and calculation of its relative location to the car allows us to avoid the obstacles when a potential risk is calculated in terms of being too close to the object. When obstacle parameters are within threshold values an action, e.g., stop the car is triggered. The use case critically relies on the detection of objects in 3D space which saves lives of passengers onboard.

Assembly Factories (He et al. 2020):
As manufacturing these days is mostly driven by robots programmed to perform specific tasks of repetitive nature. These tasks usually involve high precision tasks which humans cannot manually execute. A classic example here is on a car assembly line, the robot that assembles the wheels has to locate the exact position of the axle relative to itself and place the wheel accordingly, clearly this encompasses the same concept of pose detection that our application provides. Failure to detect the axle location can cause misalignment and possible damage to the car.

Augmented Reality (He et al. 2020):
One of the most useful applications of our system is in the field of augmented reality which many industries rely on to model and test their systems. Our system allows us to map the pose of the object in the real world to virtual objects which an AR program can use to perform its related operations such as simulations that represent the real-world surroundings.

1.2.4 Solution details

As previously covered briefly, each component in the system implements a vital functionality of the program, this not only includes the code and what it does but why it has to be done since the application relies on core mathematical concepts, integration of software and hardware which presents us with challenges to overcome and usability specifications. The following is a list of modules within the system that perform certain tasks based on fundamental underlying concepts.

1.2.4.1 GUI.py

Starting point of the application on boot. This module defines a simple user interface for performing key features of the application which include:

- Calibration
- Tag generation
- Pose detection - Eye-in-hand and Eye-to-hand

More selection criteria are offered down the tree depending on what the user wants to do which will be referenced in their respective modules in this section.

In the GUI.py module, apart from providing the user with a user interface, it also encompasses source code for some basic configuration including:

- Displays available cameras connected via USB and allows for selection of which camera to use for calibration. This is essential in terms of flexibility when the user wants to interface with different cameras depending on their precision requirements and camera available. Implementing this feature generates flexibility for the application.
- After the selection of the desired camera to be used. User is prompted to select the resolution types initially comprising of:
    - Standard 480p
    - High 720p
    - Full HD 1080p
- Depending on the user selection, the respective frame width and height will be set. It is important to note here that the user should be aware of the supported maximum resolution of the camera they are using when selecting resolution for optimum performance.
- Following resolution, the user is prompted to select the Frames Per Second (FPS). Users should check the maximum FPS supported by their camera. For users requiring higher precision and lower procession times using expensive cameras, they are able to set maximum FPS supported by their camera. This allows greater flexibility for the application and hence enlarges the application environment.

All these inputted details are saved on a data structure in python named a data dictionary to store data on key-value mapping basis.

This dictionary is then passed to a function camCapture() which resides in Calibrate.py and will be explained in the next subsection.

Calibration() function in Calibrate is called at the end and also will be explained in the next subsection

1.2.4.2 Calibrate.py

This module is used when the user wants to perform camera calibration, this is not usually done unless the user is interfacing a different camera from previous. This calibration step is required for each different camera that is used and following is an explanation of why it needs to be done:

- The camera image is essentially a two-dimensional representation of a three dimensional world. Thus, the real world is projected onto the camera sensor through a lens. The lens gives rise to a problem of image distortion. This problem can cause images to be stretched, compressed etc which cause unwanted phenomena such as poor depth perception and radial distortion, which eventually leads to inaccurate pose estimation of an object in the real world. Therefore, the image has to be undistorted mathematically using calibration in our software. The calibration takes into account the properties of the distortion that are implied by the camera lens being used and attempts to reverse these.
- To gather information of the distortion properties for the camera lens, multiple images must be taken of a known object such as a chessboard to retrieve distortion parameters. The reason for using a chessboard is that we know the relative real-world coordinates of these squares in the chess board and we receive the coordinates in the image. These two sets of data allow us to calculate the distortion coefficients.
- In this module, source code is implemented to run a video feed in which the chessboard must be presented to the camera in varying orientations and the program will automatically save a .jpg image every 30 frames into the specified directory. This is also where the resolution types and FPS are set which were predefined by the user in GUI.py for the camera setup when taking images. Again, the user can take as many frames as required, generally, increased precision is proportional to greater number of frames captured to an extent.

The above is done after invoking the camCapture() in GUI.py.

As we saw later, the Calibration() function was invoked in the GUI.py. What this function now does is, imports the chessboard images from the directory and prompts the user to enter the width, height, and size of squares for the chessboard to ensure flexibility in chess boards with differing dimensions (these dimensions need to be measured manually).

What the function then does is where the magic resides:

- It creates a numpy array to hold the real world 3D points of the chess board corners. Each image saved in the directory is loaded and cycled through one by one,

converting it to grayscale, finding the chessboard corners, adding them to the list of 3D points, refining the corners in the image to pixel accuracy and adding to the list of 2D image points. Draw these on the image and display image one by one for all images. After all images are cycled through, the window is closed. Now what the application does is, call the cv2.calibrateCamera method, this method uses the list of 3D and 2D points to calculate the distortion matrix and the calibration matrix for the camera/lens we are using and saves these in a .npy file. So that we do not need to do the calibration again unless we change cameras.

The calibration and distortion matrix define the properties of the camera lens and help us to undistort the images as will be seen in PoseDetector.py.

The distortion matrix is calculated for both radial distortion and tangential distortion. Radial distortion causes straight lines to appear curved (OpenCV n.d.) and tangential distortion causes some areas in the image to appear closer than they actually are (OpenCV n.d.). The following equations represent the radial distortion and tangential distortion:

$$x_{distorted} = x(1 + k_1 r^2 + k_2 r^4 + k_3 r^6)$$
$$y_{distorted} = y(1 + k_1 r^2 + k_2 r^4 + k_3 r^6)$$

Figure 2 - Radial Distortion (OpenCV n.d.)

$$x_{distorted} = x + [2p_1 xy + p_2(r^2 + 2x^2)]$$
$$y_{distorted} = y + [p_1(r^2 + 2y^2) + 2p_2 xy]$$

Figure 3 - Tangential Distortion (OpenCV n.d.)

These equations are embedded in the cv2.calibrateCamera method which uses all the x and y coordinates (list of 2D image points) to calculate the Distortion coefficients similar to how we can find the gradient of a line using two x and y coordinates.

$$Distortion\ coefficients = (k_1 \quad k_2 \quad p_1 \quad p_2 \quad k_3)$$

Figure 4 - Distortion Coefficients (OpenCV n.d.)

The image can be undistorted by solving for the coefficients above.

In regard to the camera matrix, this matrix defines the intrinsic and extrinsic properties of a camera. These are the focal lengths (fx, fy) and the optical centers (cx, cy) of the camera (OpenCV n.d.). Thus the camera matrix can be used to remove the distortion implied by lenses of cameras and is represented by a 3 x 3 matrix:

$$camera\ matrix = \begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix}$$

Figure 5 - Camera Matrix (OpenCV n.d.)

This matrix is calculated by the cv2.calibrateCamera method using the 3D and 2D points. The matrix represents the intrinsic parameters of the camera. The extrinsic properties are also calculated by the method in terms of rotation and translation vectors (OpenCV n.d.) which are not required to be stored for the purpose of this application.

1.2.4.3 GenerateTags.py

To provide greater flexibility to the client using the application, a modification was implemented that allows the client to select the ArUco tag library based on what they have available or if they want to generate a tag for detection. The selection of ArUco tags for pose estimation aids to implement a robust application. ArUco tags are referred to as binary square fiducial markers (OpenCV n.d.), they entail two properties which makes them an ideal choice (OpenCV n.d.):

- **Greater number of correspondences**
  This simply means that a single marker provides enough points (corners) to obtain the pose of the camera since the detection depends on the boundary of the white and black interface of the markers.

- **Finer binary codification**
  Since the markers provide a simple two-colour interface, the binary matrix coding within the maker is robust and efficient which aids in accurate detection and even error detection.

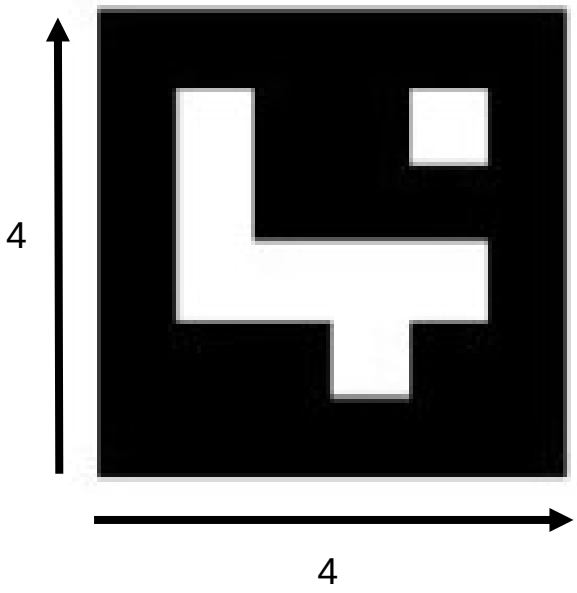

Figure 6 - AcUco Marker

The above is a sample ArUco marker, the marker is made of square blocks put together, each representing a 'bit'. Therefore, if there is a marker that is 4 x 4 bits, then the binary matrix of the marker is defined by 16 bits (OpenCV n.d.).

As can be guessed, there can exist different markers sizes. ArUco Dictionaries are developed to markers segregated by the following properties (OpenCV n.d.):

- Marker size
- Dictionary size

Thus, the objective here is to allow the user to generate and use any markers from different dictionaries based on their requirements. The brief overview of how this module supports flexibility in selecting/generating marker is provided below:

- When the user requests to generate markers from the GUI module, they are provided a list of all dictionaries to select from. Based on the user selection, the tagType and range are put into python data dictionary TagSettings and passed to a function GenerateTags in this module. This function will allow the user to select a specific tag id and size in pixels to generate. The tag will be generated, a preview of it will be shown to the user using the drawMarker() function and saved to an arbitrary directory. This tag will be used for pose detection. This module can be executed over and over again to generate different markers without having any effect on other modules as they are flexible enough to adapt to different ArUco markers.

1.2.4.4 PoseDetector.py

The crux of the application lies within this module, this is where the markers are detected and the relevant 6 DoF coordinates calculated using core mathematical concepts and essential libraries such as numpy which will be discussed briefly in this section.

When the user selects the option for detecting the pose, if the calibration and camera matrix does not exist, the program will not allow the user to proceed as this will cause the application to function inaccurately. This is implemented as a part of the non-functional requirements which are stated in the testing documentation of this report.

If the above constraints are satisfied, users are prompted to select between two detection techniques, eye to hand and eye in hand (Flandin, Chaumette & Marchand 2000).

- **eye-to-hand:**
  as the name suggests, the camera is mounted at a fixed location and observes the robot/marker from that point of view, this allows for wider perception angle, but the accuracy might be hindered.

- **eye-in-hand:**
  as the name suggests, the camera is mounted on the robot hand/controller and the camera centre moves with the robot, this allows for greater accuracy however the perception angle is decreased hence only can detect markers in its immediate field of vision.

In the previous section we defined a function to generate marker and draw it, we need to tell the program which marker type from which dictionary to detect, therefore, the function setTag is executed again and this time instead of passing the selection to GenerateTags.py, we pass it to this module. The purpose of this is the module needs to know which marker from which dictionary it needs to look for, hence it should be ensured that the same dictionary of markers must be selected. This might seem a constraint from initial judgment however this feature increases the reliability of the application as it allows us to control the behaviour of the application to a desirable extent.

Finally, we pass the tag setting and camera setting which have been configured from the previous module from GUI.py to this module as this information needs to be available for accurate detection of markers using mathematical and library techniques that are optimized for changes in hardware equipment which will be detailed below.

The code and library used to detect the markers and its position will be discussed in detail in the section source code documentation. Below we will discuss the core principles of pose detection from real world to the camera and the concepts researched and implemented to achieve results.

To get the pose of the marker in real world coordinates, it is important to understand the difference between the camera's coordinates and real-world coordinates. We will continue the discussion by referring to the figure 2 below.

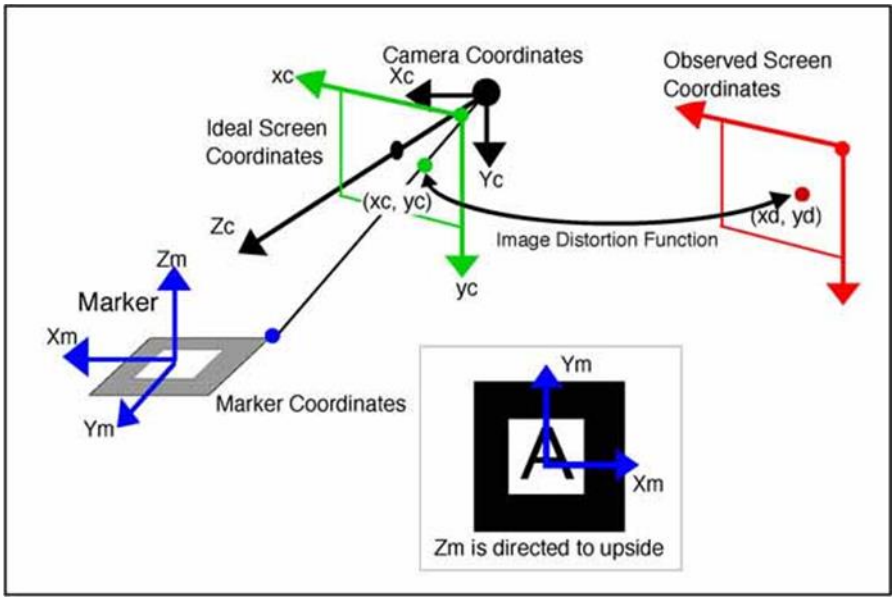

Figure 7 - Camera to Real World (ARToolKit n.d.)

As seen in figure 7, there are few inconsistencies between the alignment of the real world coordinates and camera coordinates. The Z axis comes straight out of the camera but in the real world it is upwards, similarly X and Y are also not aligned. The benefits here is that the scale for both environments is the same e.g. 1 mm for the camera is 1 mm in the real world and that the axes are 90 degrees apart from both perspectives. Only the orientation differs.

ArUco's function estimatePoseSingleMarkers() eradicates this above issue based on inputs such as marker size, camera matrix, camera distortion created earlier and generates a list of rotational and translational vectors. These vectors correspond to x, y and z coordinates and aid us in detection of the pose. The translational vector generated gives us the transition in x, y and z axis which can be outputted immediately, however, some more work needs to be done on the rotational vector to output pitch, roll and yaw values.

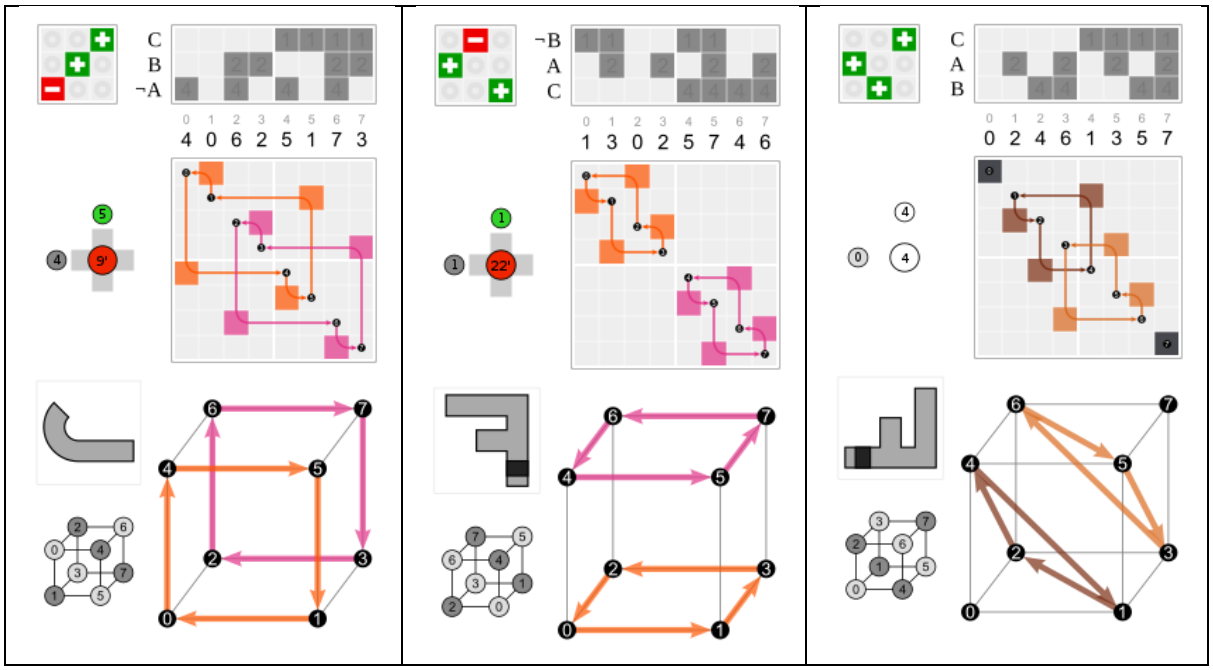

Figure 8 - Rotation Matrices (Wikipedia Contributors 2019)

A representation of the rotation can be seen above, a rotation around the y-axis (left) followed by a rotation around the z-axis (mid) is equivalent to one single 120 degrees rotation around the diagonal axis (right), this is called the rotation vector. Therefore, the rotation vector will be in the direction of the axis the rotation happens with the length of the vector being the magnitude of the rotation.

We already have the rotation vector from the estimatePoseSingleMarkers() method however it is important to take into account the camera and real world perspective once again here. When we flip the rotation vector, the rotation is still around the same axis however the direction changes.

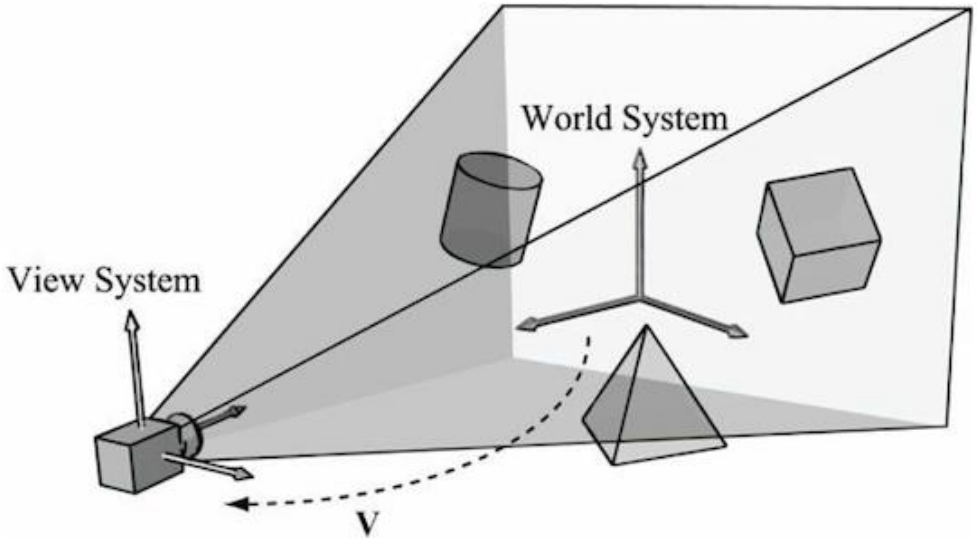

Figure 9 - Real World to Camera Coordinates

The translational vector takes us from the camera to the center of the marker, the rotation vector rotates us so that we are facing in the same direction as the marker coordinates. Since we want to know our rotation with reference to the marker, we need to go from the marker back to the camera thus we need to rotate around the same axis but in the opposite direction or flip the vector. For the translational vector we also need to flip the vector to go from marker to camera to allow for de-rotation and de-translation.

Therefore, here exists two coordinate systems, camera coordinate system and marker coordinate system, due to the misaligned orientation of the perspective axes resulting in discrepancy between coordinates affecting rotational vectors, we need to fix this by having a changes of base on the vector from the camera coordinate system to the real world coordinate system. To get the vector from another coordinate system, the concept of change of base simply means we need to multiply the vector by the transformation matrix. To go from the marker to the camera, we already have the rotational vector flipped, but we require a rotation matrix to accurately detect the pitch, yaw, and roll. Therefore, to change from rotation vector to rotation matrix (Figure 10), we require the use of Rodrigues' rotation formula (Figure 11).

$$R_x(\theta) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\theta & -\sin\theta \\ 0 & \sin\theta & \cos\theta \end{bmatrix}$$

$$R_y(\theta) = \begin{bmatrix} \cos\theta & 0 & \sin\theta \\ 0 & 1 & 0 \\ -\sin\theta & 0 & \cos\theta \end{bmatrix}$$

$$R_z(\theta) = \begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Figure 10 - Rotation Matrix X, Y, Z (Wikipedia Contributors 2019)

$$\mathbf{v}_{\text{rot}} = \mathbf{v}\cos\theta + (\mathbf{k}\times\mathbf{v})\sin\theta + \mathbf{k}\,(\mathbf{k}\cdot\mathbf{v})(1-\cos\theta)\,.$$

Figure 11 - Rodrigues' rotation formula (Wikipedia Contributors 2019)

The above formulation is used to convert corresponding rotational vectors to their corresponding rotation matrices, each matrix providing the three bits of rotational information required, figure 7 shows the correlation between rotational matrices in x, y, and z —-> roll, pitch and yaw.

$$R = R_z(\gamma)\,R_y(\beta)\,R_x(\alpha) = \begin{bmatrix} \cos\gamma & -\sin\gamma & 0 \\ \sin\gamma & \cos\gamma & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \cos\beta & 0 & \sin\beta \\ 0 & 1 & 0 \\ -\sin\beta & 0 & \cos\beta \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\alpha & -\sin\alpha \\ 0 & \sin\alpha & \cos\alpha \end{bmatrix}$$
$$= \begin{bmatrix} \cos\beta\cos\gamma & \sin\alpha\sin\beta\cos\gamma - \cos\alpha\sin\gamma & \cos\alpha\sin\beta\cos\gamma + \sin\alpha\sin\gamma \\ \cos\beta\sin\gamma & \sin\alpha\sin\beta\sin\gamma + \cos\alpha\cos\gamma & \cos\alpha\sin\beta\sin\gamma - \sin\alpha\cos\gamma \\ -\sin\beta & \sin\alpha\cos\beta & \cos\alpha\cos\beta \end{bmatrix}$$

Figure 12 - Rx, Ry and Rz rotational matrices (Wikipedia Contributors 2019)

The selection of python development was chosen due to the fact that the complex geometric calculations which lay the base for the system are provided by python development. The code specifications to implement formulas and data structures used to store this data will become evident in the source code documentation section.

By capturing the rotational and translational vectors generated by special functions we are able to decipher the pose of the camera in relation to the marker, the translation in x, y and z axes as well as the rotational properties pitch, yaw and roll will always be in reference from the center of the marker and hence any deviation whether be translational or rotational will be from the center of the marker.

1.2.4.5 Utils.py

This module defines previously discussed ArUco dictionaries as variables.

1.2.5 Diagrammatic Representation

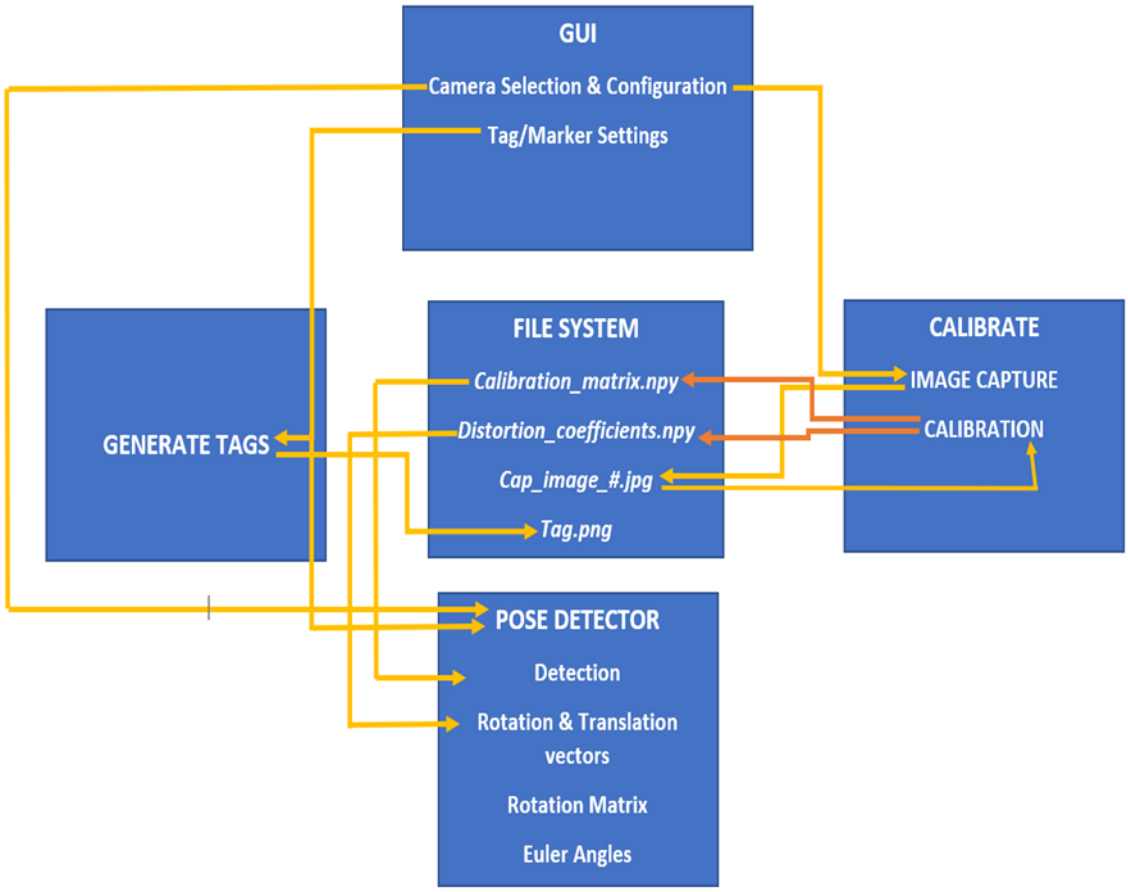

Figure 13 - Diagrammatic Representation

1.3 Source code documentation

1.3.1 GUI.py

```
#Initialise Modules
from Calibrate import Calibrate as Calibrate
from GenerateTags import GenerateTags as GenerateTags
from PoseDetector import PoseDetector as PoseDetector
import cv2
import os


#Get Camera Data
def getCameras():
	print("\n================================================")
	print("Obtaining Camera/s - Please wait")

	#List camera/s
	cameraList = []
	i = 0
	while i < 10:
		cap = cv2.VideoCapture(i)
		if cap.read()[0]:
			cameraList.append(i)
			cap.release()
		i += 1

	#Set Camera
	set = False
	while set == False:
		try:
			print("\n================================================")
			print("List of avaliable camera/s index")
			i = 0
			for cam in cameraList:
				print("{}: Camera Index {}".format(i,cam))
				i += 1
			userInput = int(input("\nSelect camera index: "))
		except ValueError:
			input("Please input Numeric Values.")
		else:
			if userInput < 0 or userInput > len(cameraList) - 1:
				input("Please input from Command List.")
			else:
				print("Creating Camera Preview")
				index = cameraList[userInput]
				cap = cv2.VideoCapture(index)
				while True:
					ret, frame = cap.read()
					cv2.imshow("Camera index {} - Press q to close".format(index), frame)
					key = cv2.waitKey(1) & 0xFF
					if key == ord("q"): break

				cap.release()
				cv2.destroyAllWindows()

				while True:
					try:
						confirm = int(input("Use this camera? (0:False | 1:True)? "))
					except ValueError:
						input("Please input Numeric Values.")
					else:
						if userInput < 0 or userInput > 1:
							input("Please input from Command List.")
						elif confirm == 1:
							set = True
						break

    #Resolution of Camera
	while True:
		try:
			print("\n================================================")
			print("Resolution Types")
			print("1: Standard 480p [ 640, 480]")
			print("2: High 720p     [1280, 720]")
			print("3: Full HD 1080p [1920,1080])")

			userInput = int(input("\nPlease enter the Resolution number: "))
		except ValueError:
			input("Please input Numeric Values.")
		else:
			if userInput < 1 or userInput > 3:
				input("Please input from Command List.")
			else:
				if userInput == 1: width = 640; height = 480
				if userInput == 2: width = 1280; height = 720
				if userInput == 3: width = 1920; height = 1080
				break

	#Camera's FPS
	while True:
		try:
			print("\n================================================")
			fps = int(input("Please enter the Frames Per Second (FPS): "))
		except ValueError:
			input("Please input Numeric Values.")
		else:
			if fps <= 0:
```

```
                                        input("Error: Please enter a positive number")
                                else:
                                        break
            CameraSettings = {}
            CameraSettings["index"] = index
            CameraSettings["width"] = width
            CameraSettings["height"] = height
            CameraSettings["fps"] = fps

            return CameraSettings

#=============================================================================================================================
#=============================================================================================================================

#Set Tag type and range
def setTag():
            #Type of ArUco Tag (Maybe have standard one e.g. DICT_5X5_100 or select of list)
            good = False
            while good == False:
                        try:
                                    #https://docs.opencv.org/4.x/d9/d6a/group__aruco.html#gac84398a9ed9dd01306592dd616c2c975
                                    print("\n================================================")
                                    print("ArUco Tag List")
                                    print("0: DICT_4X4_50  | 1: DICT_4X4_100  | 2: DICT_4X4_250  | 3: DICT_4X4_1000")
                                    print("4: DICT_5X5_50  | 5: DICT_5X5_100  | 6: DICT_5X5_250  | 7: DICT_5X5_1000")
                                    print("8: DICT_6X6_50  | 9: DICT_6X6_100  | 10: DICT_6X6_250 | 11: DICT_6X6_1000")
                                    print("12: DICT_7X7_50 | 13: DICT_7X7_100 | 14: DICT_7X7_250 | 15: DICT_7X7_1000")
                                    print("16: DICT_ARUCO_ORIGINAL | 17: DICT_APRILTAG_16h5 | 18: DICT_APRILTAG_25h9")
                                    print("19: DICT_APRILTAG_36h10 | 20: DICT_APRILTAG_36h11\n")

                                    print("Formating               | DICT_ARUCO_ORIGINAL = 6X6_1024")
                                    print("DICT_5X5_100            | DICT_APRILTAG_16h5  = 4X4_30")
                                    print("5x5 - pixel (internal)  | DICT_APRILTAG_25h9  = 5X5_35")
                                    print("100 - Amount of id      | DICT_APRILTAG_36h10 = 6X6_2320")
                                    print("                        | DICT_APRILTAG_25h9  = 6X6_587")

                                    userInput = int(input("\nPlease enter the number for ArUCo tag: "))

                        except ValueError:
                                    input("Please input Numeric Values.")
                        else:
                                    if userInput < 0 or userInput > 20:
                                                input("Please input from Command List.")
                                    else:
                                                if userInput == 0: tagType = "DICT_4X4_50"; range = 50
                                                elif userInput == 1: tagType = "DICT_4X4_100"; range = 100
                                                elif userInput == 2: tagType = "DICT_4X4_250"; range = 250
                                                elif userInput == 3: tagType = "DICT_4X4_1000"; range = 1000
                                                elif userInput == 4: tagType = "DICT_5X5_50"; range = 50
                                                elif userInput == 5: tagType = "DICT_5X5_100"; range = 100
                                                elif userInput == 6: tagType = "DICT_5X5_250"; range = 250
                                                elif userInput == 7: tagType = "DICT_5X5_1000"; range = 1000
                                                elif userInput == 8: tagType = "DICT_6X6_50"; range = 50
                                                elif userInput == 9: tagType = "DICT_6X6_100"; range = 100
                                                elif userInput == 10: tagType = "DICT_6X6_250"; range = 250
                                                elif userInput == 11: tagType = "DICT_6X6_1000"; range = 1000
                                                elif userInput == 12: tagType = "DICT_7X7_50"; range = 50
                                                elif userInput == 13: tagType = "DICT_7X7_100"; range = 100
                                                elif userInput == 14: tagType = "DICT_7X7_250"; range = 250
                                                elif userInput == 15: tagType = "DICT_7X7_1000"; range = 1000
                                                elif userInput == 16: tagType = "DICT_ARUCO_ORIGINAL"; range = 1024
                                                elif userInput == 17: tagType = "DICT_APRILTAG_16h5"; range = 30
                                                elif userInput == 18: tagType = "DICT_APRILTAG_25h9"; range = 35
                                                elif userInput == 19: tagType = "DICT_APRILTAG_36h10"; range = 2320
                                                elif userInput == 20: tagType = "DICT_APRILTAG_36h11"; range = 587
                                                good = True

            TagSettings = {}
            TagSettings["tagType"] = tagType
            TagSettings["range"] = range

            return TagSettings

#=============================================================================================================================
#=============================================================================================================================

#GUI Components
running = True
cameraSetting = {}

#Program Start
while running:
            #Get User Input + Error Testing
            while True:
                        print("\n================================================")
                        print("|           Pose Detector using ArUco           |")
                        print("|              Created by Group 45              |")
                        print("|   https://github.com/DSLeong/PoseDetectorPi   |")
                        print("================================================")
                        print("|                  Command  List                |")
                        print("|                                               |")
                        print("|               1 : Camera Select               |")
                        print("|                 2 : Calibrate                 |")
                        print("|                3 : Generate Tags              |")
                        print("|         4 : Detect Pose (Eye to Hand)         |")
                        print("|         5 : Follow Pose (Eye in Hand)         |")
                        print("|                0 : Close Program              |")
                        print("|                                               |")
                        print("================================================")
```

```
                    try:
                              command = int(input("\nCommand? "))
                    except ValueError:
                              input("Please input Numeric Values.")
                    else:
                              if command < 0 or command > 5:
                                        input("Please input from Command List.")
                              else:
                                        break

          #Switch for Modules
          if command == 0: #Close Program
                    print("EXIT")
                    running = False

          elif command == 1: #Camera Select
                    print("CALIBRATION")
                    cameraSetting = getCameras()
                    print("\n================================================")
                    input("You can now run calibration")

          elif command == 2: #Calibration
                    if not cameraSetting:
                              print("\n================================================")
                              print("Camera selection does not exist.")
                              input("Please run Camera selection.")
                    else:
                              while True:
                                        try:
                                                  print("\n================================================")
                                                  userInput = int(input("Create Images (0:False | 1:True)? "))
                                        except ValueError:
                                                  input("Please input Numeric Values.")
                                        else:
                                                  if userInput < 0 or userInput > 1:
                                                            input("Please input from Command List.")
                                                  else:
                                                            if userInput == 1:
                                                                      Calibrate.camCapture(cameraSetting)
                                                            break

                              Calibrate.Calibration()

                    input("Press Enter to continue")

          elif command == 3: #Generate Tags
                    print("GENERATE TAGS")
                    tagSetting = setTag()
                    GenerateTags(tagSetting)

          elif command == 4 or command == 5: #Pose

                    #if Camera selection does not exist
                    if not cameraSetting:
                              print("\n================================================")
                              print("Camera selection does not exist.")
                              input("Please run Camera selection.")

        #if Calibration does not exist
                    elif not os.path.isfile("calibration_matrix.npy") or not os.path.isfile("distortion_coefficients.npy"):
                              print("\n================================================")
                              print("Calibration does not exist.")
                              input("Please run Calibration first.")

                    #Detect Pose
                    elif command == 4:
                              print("EYE TO HAND")
                              tagSetting = setTag()
                              PoseDetector.poseDetector(PoseDetector, None, None, None, tagSetting, cameraSetting)

                    #Follow Pose
                    elif command == 5:
                              print("EYE IN HAND")

                              while True:
                                        try:
                                                  print("\n================================================")
                                                  print("Set Values:")
                                                  inputX = int(input("X: "))
                                                  inputY = int(input("Y: "))
                                                  inputZ = int(input("Z: "))
                                        except ValueError:
                                                  input("Please input Numeric Values.")
                                        else:
                                                  break

                              tagSetting = setTag()
                              PoseDetector.poseDetector(PoseDetector, inputX, inputY, inputZ, tagSetting, cameraSetting)

                    input("Press Enter to continue")

          #ERROR CASE
          else:
                    input("ERROR")


print("EXITED VIA COMMAND")
```

Line 1-6: importing relevant modules and libraries to invoke required functions

Line 10: getCamera() function

Line 15-22: Obtaining list of cameras available, appending them to cameraList[]

Line 25-51: display list of available cameras connected to device, allow user to choose camera based on index from list, preview from chosen camera to verify

Line 53-63: confirm from user to use selected camera, if yes, proceed to resolution configuration otherwise go back to selecting camera

Line 66-84: display to user range of selection criterion for resolution of camera, restrict user to enter numeric values between 1-3, if satisfied, set width and height variables according to user input

Line 87-104: prompt user to specify frames per second (fps) provided input is numeric and positive number. Save index, width, height, fps to data dictionary cameraSettings = {}. Return the data dictionary to the caller of the function. The use of this will become clear later on.

Line 110: define setTag() function.

Line 112-166: display all available ArUco dictionaries and prompt users to select one dictionary based on numeric input between 0-20, based on input, a string tagType and integer range are saved into TagSettings data dictionary, and the dictionary is returned to the function caller. The use of this will become clear later on.

Line 176-203: display a command list in the command prompt to allow for selection of features to the system and save input in ‘command’, restrict user to input values between 0 - 5, if satisfied break out of loop.

Line 206-208: if user input is ‘0’ based on command list, exit the application by setting ‘running’ global variable to false

Line 210-214: if user input is ‘1’ based on command list, call getCamera() function and save returned details (cameraSettings data structure) to cameraSetting variable of same type.

Line 216-238: if user input is ‘2’ based on command list, implies user wants to do calibration, checks if cameraSetting exists, if not prompts user to choose camera configuration, else, let user run camCapture() function in Calibrate module by passing in cameraSetting to that function. Then, run the Calibration() function in the Calibrate module.

Line 240-244: if user input is ‘3’ based on command list, implies user wants to generate tags, setTag() function is called and its returned contents (data dictionary of TagSettings with tagType and range) are assigned to tagSetting. Then, the tagSetting is passed to Generatetags functions in the GenerateTags module. The use of this will be detailed later.

Line 245-251: if user input is ‘4’ or ‘5’ based on command list, firstly check if camera settings exist, if not, prompt user to run camera configuration first, otherwise proceed.

Line 254-258: check if calibration matrix and distortion coefficients are saved in files ‘calibration_matrix.npy’ and ‘distortion_coefficients.npy’, if not prompt user to run calibration otherwise proceed.

Line 260-264: if user input is ‘4’ (this option is for eye-to-hand pose detection) based on command list, invoke setTag() function and save its returned data dictionary TagSettings into tagSetting, pass tagSetting and CameraSetting to poseDetector function in PoseDetector module, when information is needed to accurate pose estimation as will be seen later.

Line 266-282: if user input is ‘5’ (this option is for eye-to-hand pose detection) based on command list, prompt user to input X, Y and Z with restrictions being only numeric input is valid, then invoke setTag() function and save its returned data dictionary TagSettings into tagSetting, pass tagSetting, input X,Y,Z and CameraSetting to poseDetector function in PoseDetector module, when information is needed to accurate pose estimation as will be seen later.

1.3.2 Calibrate.py

```
import argparse
import cv2
import numpy as np
import os
import re
import time
import tkinter as tk
from   tkinter import filedialog

class Calibrate:

    #Produce Images from Camera for Calibration
    def camCapture(cameraSetting):

        #Find/Create Directory
        while True:
            try:
                print("\n================================================")
                userInput = int(input("Create Directory for Images (0:False 1:True)? "))
            except ValueError:
                input("Please input Numeric Values.")
            else:
                if userInput < 0 or userInput > 1:
                    input("Please input from Command List.")
                else:
                    if userInput == 1: #Create Directory
                        dirName = "Capture_" + str(cameraSetting["index"]) + "_" + str(cameraSetting["height"])
                        try:
                            os.mkdir(dirName)
                        except FileExistsError:
                            print("Directory '" , dirName ,  "' already exists")
                        else:
                            print("Directory '" , dirName ,  "' Created")

                        dirpath = os.path.join(os.getcwd(), dirName)
                        print(dirpath)
                        break

                    elif userInput == 0: #Find Directory
                        while True:
                            print("\n================================================")
                            print("Please choose the folder where to save checkerboard images:")
                            input("Press enter to continue")
                            root = tk.Tk()
                            root.withdraw()
                            dirpath = filedialog.askdirectory()
                            print(dirpath)
                            if dirpath == "":
                                input("Please select folder")
                            else:
                                break
                        break

                    else:
                        print("ERROR")

        #Flip Camera
        while True:
            try:
                print("\n================================================")
                userInput = int(input("Flip Camera? (0: No | 1: Yes)? "))
            except ValueError:
                input("Please input Numeric Values.")
            else:
                if userInput < 0 or userInput > 1:
                    input("Please input from Command List.")
                else:
                    if userInput == 0: flip = False
                    else: flip = True
                    break


        print("Press 'q' on capture to stop")

        cap = cv2.VideoCapture(cameraSetting["index"])
        cap.set(cv2.CAP_PROP_FRAME_WIDTH, cameraSetting["width"])
        cap.set(cv2.CAP_PROP_FRAME_HEIGHT, cameraSetting["height"])
        cap.set(cv2.CAP_PROP_FPS, cameraSetting["fps"])

        frameCount = 0
        capImageCount = 0

        timeStart = time.time()
        while True:
            #reading camera frame
            ret, frame = cap.read()
            if flip: frame = cv2.flip(frame,-1)

            #Capture Image
            frameCount += 1

            if frameCount == 30:
                img = "cap_image_" + str(capImageCount) + ".jpg"
                cv2.imwrite(os.path.join(dirpath, img), frame)

                capImageCount += 1
```

```
                frameCount = 0

            #Display Video
            cv2.imshow("Image Feed - Press 'q' to stop", frame)

            key = cv2.waitKey(1) & 0xFF
            if key == ord("q"): break

        cap.release()
        cv2.destroyAllWindows()

        elapseTime = time.time() - timeStart
        print("\n================================================")
        print("Created " + str(capImageCount) + " image/s")
        print("Elapsed Time for Image Creation: " + str(elapseTime) + " seconds")

    #=============================================================================================================================
           #=============================================================================================================================

    #Calibration of Camera
    def Calibration():

        #Find Directory
        while True:
            print("\n================================================")
            print("Please choose the folder where the checkerboard images are located:")
            input("Press enter to continue")
            root = tk.Tk()
            root.withdraw()
            dirpath = filedialog.askdirectory()
            print(dirpath)
            if dirpath == "":
                input("Please select folder")
            else:
                #Check for images (in .jpg format)
                images = [file for file in os.listdir(dirpath) if re.findall(r"\w+\.jpg", file)]
                if len(images) < 1:
                    input(str(dirpath) + " - has no images (.jpg), please select a new folder")
                else:
                    break

        #Width of Checkerboard
        while True:
            try:
                print("\n================================================")
                width = int(input("Please enter the width of the checkerboard (no. of corners): "))
            except ValueError:
                input("Please input Numeric Value.")
            else:
                if width <= 0:
                    print('Error: Please enter a positive number')
                else:
                    break

        #Height of Checkerboard
        while True:
            try:
                print("\n================================================")
                height = int(input("Please enter the height of the checkerboard (no. of corners): "))
            except ValueError:
                input("Please input Numeric Value.")
            else:
                if height <= 0:
                    print('Error: Please enter a positive number')
                else:
                    break

        #Checkerboard square length
        while True:
            try:
                print("\n================================================")
                square_size = float(input("Please enter the size of the squares (mm): "))
            except ValueError:
                input("Please input Numeric Value.")
            else:
                if square_size <= 0:
                    print('Error: Please enter a positive number')
                else:
                    break


        # Apply camera calibration operation for images in the given directory path.

        # termination criteria
        criteria = (cv2.TERM_CRITERIA_EPS + cv2.TERM_CRITERIA_MAX_ITER, 30, 0.001)

        # prepare object points, like (0,0,0), (1,0,0), (2,0,0) ....,(8,6,0)
        objp = np.zeros((height*width, 3), np.float32)
        objp[:, :2] = np.mgrid[0:width, 0:height].T.reshape(-1, 2) * square_size

        # Arrays to store object points and image points from all the images.
        objpoints = []  # 3d point in real world space
        imgpoints = []  # 2d points in image plane.

        #Calibrate Images
        print("Rendering Calibration")
        usableImages = 0
        for fname in images:
            print(fname)
```

```
            #Retrieve and grayscale image
            img = cv2.imread(os.path.join(dirpath, fname))
            gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)

            # Find the chess board corners within image
            ret, corners = cv2.findChessboardCorners(gray, (width, height), None)

            # If found, add object points, image points (after refining them)
            if ret:
                usableImages += 1
                objpoints.append(objp)

                corners2 = cv2.cornerSubPix(gray, corners, (11, 11), (-1, -1), criteria)
                imgpoints.append(corners2)

                # Draw and display the corners
                img = cv2.drawChessboardCorners(img, (width, height), corners2, ret)

            #Display image with Calibration (May not need)
            cv2.imshow("Image Calibration",img)
            cv2.waitKey(1000)

        cv2.destroyAllWindows()


        #Calibration of Camera
        print("\nCalibrating Camera using Images - Please Wait")
        timeStart = time.time()
        ret, mtx, dist, rvecs, tvecs = cv2.calibrateCamera(objpoints, imgpoints, gray.shape[::-1], None, None)
        elapseTime = time.time() - timeStart

        print("\n================================================")
        print("Elapsed Time for Calibration: " + str(elapseTime) + " seconds")
        print("Amount of Used Images: " + str(usableImages) + " out of " + str(len(images)))

        print("\nCalibration Matrix:")
        print(mtx)
        print("\nDistortion:")
        print(dist)

        np.save("calibration_matrix", mtx)
        np.save("distortion_coefficients", dist)
```

Line 1-7: import required libraries, cv2, numpy, os, re, time.

Line 10: declare class Calibrate

Line 13: define function camCapture(cameraSetting), the parameter it accepts is the data dictionary that contains camera configuration information (height, width, fps etc) which was configured in the GUI module and passed to this module when Calibrate option was selected by the user. This function is to capture images of uniform patterns and save them for camera calibration purposes.

Line 16-36: prompt user to create a directory to save the images that will be captured, if user input is ‘1’ (wants to create directory), then create directory if doesn’t exist already to path defined by dirpath. A directory will get created to save images and a path will be created.

Line 39-55: if user input is ‘0’ (user does not want to create directory), user is prompted to choose an existing directory in which they wish to save the images, the path to the chosen directory saved in dirpath (line 46). If dirpath is empty, prompt the user to select the directory again.

Line 58-71: prompt user to choose to flip camera, provided that user input is numeric and either 0 or 1, if ‘0’, set flip variable to false, else set flip to true.

Line 75-78: start video capture using VideoCapture(cameraIndex) function from cv2 library, the parameter passed in here is the ‘index’ in the camera setting data dictionary configured in GUI, it essentially tells the function from which camera to capture from. Similarly, camera

frame width, height and fps are used to set the frame parameters of the video feed using cv2 inbuilt functions.

Line 80-81: set frameCount and capImageCount to 0. These are used to record the number of frames processed per image and how many images are taken during capture.

Line 83: get timestamp of current execution using from time library.

Line 84-106: create loop, read camera frame into variable cap, if flip was set true, flip frame, increase frameCount by 1 to signify we have captured a image, when frameCount reaches 30, save an image to the specified directory previously using cv2.imwrite, increase capImageCount by 1 as we have captured one image (an image is captured every 30 frames), set frameCount back to 0.

Line 108-112: capture timestamp after exiting capture, display the number of images captured and the time taken to capture those images.

Line 117: define function Calibration(). This function will use the captured images to calibrate the camera and provide us with the previously discussed camera and distortion matrices.

Line 120-137: prompt user to select the folder where the images were previously saved, save the path of the selected folder in dirpath, load all .jpg images in images.

Line 139-151: prompt the user to enter the width of the chess board used for calibration, this is the number of corners across, saved in width provided input is numeric and positive.

Line 152-163: prompt the user to enter the height of the chess board used for calibration, this is the number of corners vertically, saved in height provided input is numeric and positive.

Line 164-176: prompt the user to enter the size of the squares on the chess board used for calibration, this is the size in millimeters of the square (width or height), saved in square_size provided input is numeric and positive.

Line 180-193: create numpy array to hold the real world 3D points of the chessboard (line 184,185). Create two empty lists, one to hold the 3D points and other to hold the 2D image points.

Line 194: cycle through the images loaded from file previously.

Line 198-199: Opening the images up using cv2.imread, converting to gray.

Line 202: Find the chessboard corners within the image using cv2.findChessboardCorners function.

Line 205-207: If a corner is found, add a 3D point (objp) to the list of 3D points defined previously (objpoints).

Line 209-210: Then we refine the corners found in the image to sub-pixel value for greater accuracy using cv2.cornerSubPix() function and save in corners2, add them to the list of 2D image points (imgpoints).

Line 213: Draw found corners on the chess board image using cv2.drawChessboardCorners() function. Then show the image in a window using cv2.show() function.

Line 219: close window after cycling through all images.

Line 224: create timestamp for current time of executing program (will be used to record the time taken for the calibration).

Line 225: call cv2.calibrateCamera method and pass to it the list of 3D points (objpoints) and the list of 2D points (imgpoints). Then the function works out and returns the calibration matrix (mtx) and distortion coefficients (dist).

Line 226: capture time and subtract from timeStart to record the time taken to complete calibration and save in elapseTime.

Line 228-238: print elapsed time and images used for calibration, print the calibration and distortion matrix and save them to the working directory. These matrices will be used by the PoseDetector module to accurately detect the pose of the object.

1.3.3 GenerateTags.py

```
import argparse
import cv2
import numpy as np
import os
import sys
import tkinter as tk
from tkinter import filedialog
from utils import ARUCO_DICT


class GenerateTags:
    def __init__(self, TagSettings):
        #Find/Create Directory
        while True:
            try:
                print("\n==================================================")
                userInput = int(input("Create Directory for Tag (0:False 1:True)? "))
            except ValueError:
                input("Please input Numeric Values.")
            else:
                if userInput < 0 or userInput > 1:
                    input("Please input from Command List.")
                else:
                    if userInput == 1: #Create Directory
                        dirName = "Tags"
                        try:
                            os.mkdir(dirName)
                        except FileExistsError:
                            print("Directory '" , dirName ,  "' already exists")
                        else:
                            print("Directory '" , dirName ,  "' Created")

                        dirpath = os.path.join(os.getcwd(), dirName)
                        print(dirpath)
                        break

                    elif userInput == 0: #Find Directory
                        while True:
                            print("\n==================================================")
                            print("Please choose the directory where the ArUCo tag will be saved")
                            input("Press enter to continue")
                            root = tk.Tk()
                            root.withdraw()
                            dirpath = filedialog.askdirectory()
                            print(dirpath)
                            if dirpath == "":
                                input("Please select folder")
                            else:
                                break
                        break

                    else:
                        print("ERROR")

        #ID of ArUco Tag
        while True:
            try:
                print("\n==================================================")
                iden = int(input("Please enter the ID of ArUCo tag to generate: "))
            except ValueError:
                input("Please input Numeric Values.")
            else:
                if iden < 0 or iden > int(TagSettings["range"]) - 1:
                    input("Please input value within range of 0 to " + str(int(TagSettings["range"]) - 1))
                else:
                    break

        #Size of Tag (pixel) or just have sets of values or standard value
        while True:
            try:
                print("\n==================================================")
                size = int(input("Please enter the size of ArUCo tag to generate (pixel): "))
            except ValueError:
                input("Please input Numeric Values.")
            else:
                if size <= 0:
                    print("Error: Please enter a positive number")
                else:
                    break

        # Check to see if the dictionary is supported
        arucoDict = cv2.aruco.Dictionary_get(ARUCO_DICT[TagSettings["tagType"]])
```

```
        print("\n==========================================================")
        print("Generating ArUCo tag of type '{}' with ID '{}'".format(TagSettings["tagType"], iden))
        tag_size = size
        tag = np.zeros((tag_size, tag_size, 1), dtype="uint8")
        cv2.aruco.drawMarker(arucoDict, iden, tag_size, tag, 1)

        # Save the tag generated
        tag_name = f'{dirpath}/{TagSettings["tagType"]}_id_{iden}.png'
        cv2.imwrite(tag_name, tag)
        cv2.imshow("ArUCo Tag", tag)
        cv2.waitKey(0)
        cv2.destroyAllWindows()
```

Line 1-8: import required libraries

Line 11: declare class GenerateTags, this class is used to generate tags (aruco markers) based on the library the user has chosen in GUI.

Line 12: define main function, which accepts the data dictionary TagSettings that contains information as to from which particular Aruco dictionary (tagType) the user wants to generate the marker (specified in GUI).

Line 14-54: user is asked if they want to create a directory (if user inputs ‘1’) to save the generated marker in or they wish to save the marker to an already created directory (if user inputs ‘0’). If the user inputs ‘1’, a directory named ‘Tags’ is created and saved to a path in the file system, the path where the directory is created is created. If user inputs ‘0’, user is prompted to select a directory from the file system using filedialog.askdirectory() function. The path of the selected directory is assigned to dirpath.

Line 56-66: user is prompted to select id of the ArUco marker within the chosen dictionary (note that different dictionaries have different number of tags (ids)) a valid tag ID in this case is determined by the ‘range’ attribute set in the TagSettings() dictionary which indicates the number of IDs present in that dictionary, hence the user must chose within the range of their selected (from GUI module) dictionary. The chosen ID is assigned to the iden variable.

Line 68-80: user is prompted to select the size of the square marker to generate in pixels. The size is assigned to variable size.

Line 82: get the ArUco dictionary specified by tagType in TagSettings and assign to arucoDict.

Line 84 - 88: create a tag using the np.zeros() function based on tag size. Use the cv2.aruco.drawMarker() function which accepts the parameters arucoDict, iden (tag id), tag size and tag (generated tag) to draw the tag in a window on the screen.

Line 90-95: name the tag based on the tagType and id, save the tag (using cv2.imwrite()) to a path previously defined by dirpath, as a .png image. Show the tag using cv2.imshow() function which accepts the tag that was created.

1.3.4 PoseDetector.py

```
import cv2
import math
import numpy as np

from PanTilt import PanTilt as PanTilt
from utils import ARUCO_DICT


class PoseDetector:

    #Check if Rotation Matrix
    def isRotationMatrix(R):
        Rt = np.transpose(R)
        shouldBeIdentity = np.dot(Rt, R)
        I = np.identity(3, dtype=R.dtype)
        n = np.linalg.norm(I - shouldBeIdentity)
        return n < 1e-6

    #=============================================================================================================
             #=============================================================================================================

    #Convert Rotation 3x3 Matrix to euler Angles
    def rotationMatrixToEulerAngles(self, R):
        assert (self.isRotationMatrix(R))

        sy = math.sqrt(R[0, 0] * R[0, 0] + R[1, 0] * R[1, 0])

        singular = sy < 1e-6

        if not singular:
            x = math.atan2(R[2,1], R[2,2])
            y = math.atan2(-R[2,0], sy)
            z = math.atan2(R[1,0], R[0,0])
        else:
            x = math.atan2(-R[1,2], R[1,1])
            y = math.atan2(-R[2,0], sy)
            z = 0

        return np.array([x, y, z]) #returns pitch, roll and yaw (units?)


    #=============================================================================================================
             #=============================================================================================================

    #Display Pose
    def Display(self,x,y,z,ex,ey,ez):
        print("==========================")
        print("|    Translation  (mm)    |")
        print("==========================")
        print("|                         ")
        print("|  X (Red)  : {:4.0f}".format(x))
        print("|  Y (Green): {:4.0f}".format(y))
        print("|  Z (Blue) : {:4.0f}".format(z))
        print("|                         ")
        print("==========================")
        print("| Rotation (euler/degree) |")
        print("==========================")
        print("|                         ")
        print("| EulX: {:4.0f}".format(ex))
        print("| EulY: {:4.0f}".format(ey))
        print("| EulZ: {:4.0f}".format(ez))
        print("|                         ")
        print("==========================")
        print("|Press 'q' on cap  to stop|")
        print("==========================")
        print(" ")

    #=============================================================================================================
             #=============================================================================================================


    #Estimate Pose Values
    def poseDetector(self, inputX, inputY, inputZ, tagSetting, cameraSetting):
        #Set Tag Type
        aruco_dict = cv2.aruco.getPredefinedDictionary(ARUCO_DICT[tagSetting["tagType"]])

        #Get Marker Size
        while True:
            try:
                print("\n================================================")
                marker_size = float(input("Please enter the full length of the marker being used (mm): "))
            except ValueError:
                input("Please input Numeric Values.")
            else:
                if marker_size <= 0:
                    print('Error: Please enter a positive number')
                else:
                    break

        #Load Camera Calibration
        camera_matrix = np.load("calibration_matrix.npy")
        camera_distortion = np.load("distortion_coefficients.npy")

        #Eye in hand?
        follow = True
        flip = False
        PanTilt.reset()

        #Detect
        if inputX == None and inputY == None and inputZ == None:
            follow = False
        else:
            #Flip Camera
            while True:
                try:
                    print("\n================================================")
                    userInput = int(input("Flip Camera? (0: No | 1: Yes)? "))
                except ValueError:
                    input("Please input Numeric Values.")
                else:
                    if userInput < 0 or userInput > 1:
                        input("Please input from Command List.")
                    else:
                        if userInput == 1: flip = True
                        break
```

```
        #Camera Setting
        cap = cv2.VideoCapture(cameraSetting["index"])
        cap.set(cv2.CAP_PROP_FRAME_WIDTH, cameraSetting["width"])
        cap.set(cv2.CAP_PROP_FRAME_HEIGHT, cameraSetting["height"])
        cap.set(cv2.CAP_PROP_FPS, cameraSetting["fps"])

        while True:
            #
            ret, frame = cap.read()

            #Flip Vertically and Horizontally
            if flip: frame = cv2.flip(frame,-1)

            #
            gray_frame = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)

            #
            corners, ids, rejected = cv2.aruco.detectMarkers(gray_frame, aruco_dict, camera_matrix, camera_distortion)

            #If Marker detected
            if len(corners) > 0:
                #
                cv2.aruco.drawDetectedMarkers(frame, corners)

                #estimatePoseSingleMarkers
                #rvec_list_all = Rotation Vector of Marker/s from Camera
                #tvec_list_all = Translation Vector of Marker/s from Camera
                #
                rvec_list_all, tvec_list_all, _objPoints = cv2.aruco.estimatePoseSingleMarkers(corners, marker_size, camera_matrix, camera_distortion)

                #Obtain First Marker
                rvec = rvec_list_all[0][0]
                tvec = tvec_list_all[0][0]

                #Draw Axes on Marker
                cv2.drawFrameAxes(frame, camera_matrix, camera_distortion, rvec, tvec, 50)

                #Calculate actual 'rvec' and 'tvec'
                rvec_flipped = rvec * -1
                tvec_flipped = tvec * -1
                rotation_matrix, jacobian = cv2.Rodrigues(rvec_flipped)
                realworld_tvec = np.dot(rotation_matrix, tvec_flipped)

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

                z = np.sqrt(x**2 + y**2 + z**2)

                #Euler Angles (degree)
                eulerX, eulerY, eulerZ = self.rotationMatrixToEulerAngles(self, rotation_matrix)

                #If Eye in Hand
                if follow:
                    PanTilt.EyeInHand(x, y, z, math.degrees(eulerX), math.degrees(eulerY), math.degrees(eulerZ), inputX, inputY, inputZ)

                #Display on Command Prompt
                self.Display(self,x, y, z, math.degrees(eulerX), math.degrees(eulerY), math.degrees(eulerZ))

                #Display on output
                tvec_str = "x=%4.0f y=%4.0f z=%4.0f eulerZ=%4.0f"%(x, y, z, math.degrees(eulerZ))
                cv2.putText(frame, tvec_str, (20, 40), cv2.FONT_HERSHEY_PLAIN, 2, (0, 0, 255), 2, cv2.LINE_AA)

            #
            cv2.imshow("Pose Estimation - Press 'q' to stop", frame)

            #Check for exit
            key = cv2.waitKey(1) & 0xFF
            if key == ord('q'): break

        #
        cap.release()
        cv2.destroyAllWindows()
```

Line 1-6: import cv2, math, numpy, PanTilt and ARUCO_DICT from utils.

Line 9: declare class pose detector

Line 12-18: define function isRotationMatrix(), this function takes in a rotation matrix as a parameter which has been obtained from applying the Rodrigues' rotation formula on the rotational vector. This function checks if the matrix is a valid rotation matrix.

Line 22-40: declare function rotationMatrixToEulerAngles(), this function takes in the rotation matrix as a parameter and calculates the corresponding Euler Angles x, y and z corresponding to pitch, yaw and roll properties, returns the angles in a numpy array with x, y and z stored at specific index positions.

Line 46-66: declare display() function which accepts translational x, y, and z coordinates and corresponding Euler angles and displays them in the console.

Line 73: declare poseDetector() function which accepts x, y, z coordinates (to be used if eye-to-hand option is selected), tagSetting which has been specified in GUI module and tells the program which tag ID it should detect from which library, cameraSetting specifies the camera configuration (also set in GUI) to be used when detecting markers (resolution, fps etc) this allows the system to be flexible as it is compatible with cameras ranging from low end to high end, this allows users to detect pose to very accurate level if using high resolution cameras.

Line 75: get the corresponding tagType requested from the user using cv2.aruco.getPredefinedDictionary() function and save in aruco_dict variable.

Line 78-89: prompt user to specify the size of the maker being used provided the input is numeric and positive. This is needed to detect the marker, incorrect specification of size will result in the incorrect coordinate information being perceived. E.g., if specified size is larger than actual size, the marker is perceived to be closer.

Line 91-92: load the calibration matrix and distortion matrix from the file system (calculated and saved during Calibrate module) and save to variables camera_matrix and camera_distortion variables.

Line 94-116: if user chose the eye-in-hand option, then set follow to true, otherwise if it is an eye-to-hand selection (i.e., inputX, inputY, inputZ don't exist) set follow to false. Here we are simply setting flags based on whether it is an eye-in-hand or eye-to-hand operation. If eye-in-hand, allows the user to flip the camera if desired.

Line 118-121: set video capture settings using cv2.VideoCapture(cameraIndex) function from cv2 library, the parameter passed in here is the ‘index’ in the camera setting data dictionary configured in GUI, it essentially tells the function from which camera to capture from. Similarly, camera frame width, height and fps/frame rate are used to set the frame parameters of the video feed using cv2 inbuilt functions. All these configurations were specified by the user in GUI and saved to the cameraSetting data dictionary.

Line 123-132: while in continuous loop, take a frame from the camera using cap.read(). If the flip was set previously, flip the camera frame and convert it to grayscale.

Line 134: call function aruco.detectMarkers() which accepts parameters gray frame, aruco_dict(tells from which dictionary it should detect markers), camera_matrix and camera_distortion (for accurate detecting by eradicating camera lens effects discussed in design and architecture). This function will return a list (corners) of all the corners of all ArUco markers found, a list of all the ids of all the markers it found and also a list of rejected markers.

Line 137-139: if found at least one marker, we call function cv2.aruco.drawDetectedMarkers() by passing in the frame we want to draw on and the list of corners found. This function will draw a green line around the found markers.

Line 145: we call the function cv2.aruco.estimatePoseSingleMarkers() function which accepts the list of corners, marker size, calibration and distortion matrices. The function

returns a list of rotation vectors (rvec_list_all), list of translation vectors (tvec_list_all) and list of object points (_objPoints). The translation vector is a list of vectors for each marker found and it tells us how far forward out of the camera to come, how much to the left, right, up and down (x, y, z) to go from center of camera to center of marker. The rotation vector tells us how much to rotate in 3D to line up perfectly square with respect to the marker. These vectors will be used to work out the real-world coordinates, currently, these vectors go from the camera to the marker.

Line 148-149: obtain the rotational and translational vectors of the first marker from the list of vectors (as the list of vectors correspond to different markers) and assign them to rvec and tvec.

Line 152: call cv2.drawFrameAxes() functions which accepts parameters frame, the two calibration matrices, and the translational and rotational vector of the first marker from the list. This function draws the corresponding x, y and z axis on the makers.

Line 155-156: we flip the rotational and translational vector to go from marker to camera (instead of going from camera to marker as discussed in previous theory section). Save then flipped vectors in rvec_flipped and tvec_flipped variables.

Line 157: As discussed, to obtain the Euler angles for rotation, we need the rotation matrix, currently we only have the rotation vector flipped. Therefore, we obtain the rotation matrix by applying the cv2.Rodrigues() formula to the rotational vector.

Line 158: as the coordinates change in their own perspectives, we need to do the changes of base on vectors so that the same change in coordinates in camera and real world environment is picked, as discussed previously the change of base between real world and camera coordinates can be done by simply by performing a dot product between the rotation matrix and translational vector. This is done by using the np.dot() function of numpy which accepts the rotation matrix and translational vector and returns the real world translational vector (realworld_tvec).

Line 161-163: get x, y and z coordinates from new translation vector.

Line 166: call function rotationMatrixToEulerAngles() previously defined at the top of the module and pass in the rotation matrix which is used to return the Euler x, y and z angles of rotation.

Line 169-188: display the x, y, z coordinates and Euler angles to the console using display() function. Display makers and the data (radians converted to degrees) on the screen output using cv2.putText() function which accepts parameters, frame, the string to write and font & placement properties.

If ‘q’ is pressed, then close the window and return to the main program.

1.3.5 Utils.py

```
import cv2

ARUCO_DICT = {
        "DICT_4X4_50": cv2.aruco.DICT_4X4_50,
        "DICT_4X4_100": cv2.aruco.DICT_4X4_100,
        "DICT_4X4_250": cv2.aruco.DICT_4X4_250,
        "DICT_4X4_1000": cv2.aruco.DICT_4X4_1000,
        "DICT_5X5_50": cv2.aruco.DICT_5X5_50,
        "DICT_5X5_100": cv2.aruco.DICT_5X5_100,
        "DICT_5X5_250": cv2.aruco.DICT_5X5_250,
        "DICT_5X5_1000": cv2.aruco.DICT_5X5_1000,
        "DICT_6X6_50": cv2.aruco.DICT_6X6_50,
        "DICT_6X6_100": cv2.aruco.DICT_6X6_100,
        "DICT_6X6_250": cv2.aruco.DICT_6X6_250,
        "DICT_6X6_1000": cv2.aruco.DICT_6X6_1000,
        "DICT_7X7_50": cv2.aruco.DICT_7X7_50,
        "DICT_7X7_100": cv2.aruco.DICT_7X7_100,
        "DICT_7X7_250": cv2.aruco.DICT_7X7_250,
        "DICT_7X7_1000": cv2.aruco.DICT_7X7_1000,
        "DICT_ARUCO_ORIGINAL": cv2.aruco.DICT_ARUCO_ORIGINAL,
        "DICT_APRILTAG_16h5": cv2.aruco.DICT_APRILTAG_16h5,
        "DICT_APRILTAG_25h9": cv2.aruco.DICT_APRILTAG_25h9,
        "DICT_APRILTAG_36h10": cv2.aruco.DICT_APRILTAG_36h10,
        "DICT_APRILTAG_36h11": cv2.aruco.DICT_APRILTAG_36h11
}
```

Line 1-25: define all ArUco (using cv2.aruco.DICT***) dictionaries as strings and saving them in the data dictionary ARUCO_DICT.

1.4 Usability Report

1.4.1 How to do?

Perform usability testing:

1. Test report (maybe pi) and usability assessment report (windows).
2. Record video of user performing all tasks (this video will be proof of usability testing and some functional tests)
3. Devise tests for non-functional test cases
4. Usability assessment plan
5. Edit test plan, edit SRS, and Project plan.

1.4.2 Testing Procedure

1. Test subject was briefed on the background, theory and aims of the project.
2. Test subject was introduced to the user guide.
3. Test subject was monitored by the moderator and recorded performing the tasks.
4. No intervention from moderator unless requested by subject, if the case, all details of the intervention recorded and reported.

Moderator	**Setting Camera**	**Creating Images**	**Calibrating**	**Generate Tag/s**	**Detect Pose**	**Other Comments**
Participant 1 - Daniel	Some Assistance (Guide)	Some Assistance (Guide)	Some Assistance (Guide)	No Difficulty	No Difficulty	Needed Guide/ information
Participant 2 - Daniel	Some Assistance (Guide)	Some Assistance (Guide)	Some Assistance (Guide)	No Difficulty	No Difficulty	Dummy proof system
Participant 3 - Preenit	Some Assistance (Guide)	Some Assistance (Guide)	Some Assistance (Guide)	Some Assistance (Guide)	Some Assistance (Guide)	Dummy proof system
Participant 4 - Preenit	Some Assistance (Guide)	Some Assistance (Guide)	Some Assistance (Guide)	Some Assistance (Guide)	Some Assistance (Guide)	Dummy proof system

1.4.3 Expanded Details

1.4.3.1 Participant 3

Participant 3	**Setting Camera**	**Creating Images**	**Calibrating**	**Generate Tag/s**	**Detect Pose**
Scenario Completion	Completed without assistance except the guide	Completed without assistance except the guide	Completed without assistance except the guide	Completed without assistance except the guide	Completed without assistance except the guide
Error/s	No errors were	No errors were	No errors were	Could not exit the	No errors were

	encountered (including critical and non-critical)	encountered (including critical and non-critical)	encountered (including critical and non-critical)	generating tags command, later discovered the tag window used to show tags created must be closed to move on. (non-critical error).	encountered (including critical and non-critical)
Subjective Evaluation	Easy to navigate and camera preview enforces confidence after selecting the camera to use.	Require more prompts for basic features so user does not go looking through the guide document for things that are not necessarily obvious	Simple process to calibrate camera, requiring minimal procedure, mostly automated	Simple procedure with ease of navigation, quite flexible application as allows to choose from range of tags	Quite an well rounded program, provides increased flexibility with minimal configuration steps, able to detect pose with just a few clicks after initial one-time configuration is done
Time on Task	2 mins, 25 secs	2 mins, 22 secs	2 mins, 10 secs	3 min, 18 secs	3 mins, 19 secs

Video Recordings:

Camera Select – https://youtu.be/1HWdiYyCuaU
Calibration – https://youtu.be/Arq2QwTZxfw
Generate Markers – https://youtu.be/kk40EY_mUFQ
Pose Detection - https://youtu.be/4sI-ZR5JbAE

1.4.3.2 Participant 4

Participant 4	**Setting Camera**	**Creating Images**	**Calibrating**	**Generate Tag/s**	**Detect Pose**
Scenario Completion	Completed without assistance	Completed without assistance	Completed without assistance	Completed without assistance	Completed without assistance

	except the guide	except the guide	except the guide	except the guide	except the guide
Error/s	No errors were encountered (including critical and non-critical)	No errors were encountered (including critical and non-critical)	No errors were encountered (including critical and non-critical)	No errors were encountered (including critical and non-critical)	No errors were encountered (including critical and non-critical)
Subjective Evaluation	The advantage is that the user has the choice to choose between multiple cameras (flexibility). Weakness is the user doesn't know which index represents which camera.	The calibration has great accuracy but the user interface could be improved for better user experience (less confusion).	The calibration has great accuracy but the user interface could be improved for better user experience (less confusion).	Provides a great range of choice and therefore flexibility for the user. No particular weaknesses here.	The ability to detect multiple degrees of freedom values. The weakness is that there needs to be more prompting to the user in terms of emphasis put on the prompt instructions in the application.
Time on Task	40 secs	2 mins, 5 secs	2 mins, 5 secs	1 min, 30 secs	2 mins, 32 secs

Video Recordings:

Camera Select – https://youtu.be/1HWdiYyCuaU
Calibration – https://youtu.be/Arq2QwTZxfw
Generate Markers – https://youtu.be/kk40EY_mUFQ
Pose Detection - https://youtu.be/4sI-ZR5JbAE

1.5 Testing Documentation

For all test cases and result, refer to Test Plan.

1.6 Standards, Reports, and Metrics

In the progression of this project, there were some coding standards taken into consideration which are documented below for future development and/or maintenance/evolution purposes.

Following and implementing coding rules standards yields the following benefits for software (Perforce 2019):

- Safe & reliable
- Security
- Testable
- Maintainable
- Portable

Standards are usually language specific however no matter which language the application is written in, it is necessary to follow the best practices.

The 6 DoF Pose Detector was entirely built using python due its flexible nature which allows ease of implementation. The following Standards were followed for robust application development, evidence from source code is also provided:

- **Code repository and version control**
  We have used GitHub as our repository management system for the python code. Embedding the practice for having version control on a central repository for source code management allows versioning systems meaning the files are archived in the repository and still exist even if you changed something in the near future, the old stuff is still there and the new and old code can be tagged (e.g. V1, V2 etc) so we can keep records of changes that are made to make code tracing easier.

- **Making use of the right python data structures**
  This application particular requires flexibility in terms of data structures as essentially the application depends on the data contained in these data structures, some of the examples of data structures used in the program with evidence are as follows:
    - Array

```
# Arrays to store object points and image points from all the images.
objpoints = []  # 3d point in real world space
imgpoints = []  # 2d points in image plane.
```

    - Data dictionary

```
TagSettings = {}
TagSettings["tagType"] = tagType
TagSettings["range"] = range
```

- Matrices (vectors)

```
#Calculate actual 'rvec' and 'tvec'
rvec_flipped = rvec * -1
tvec_flipped = tvec * -1
rotation_matrix, jacobian = cv2.Rodrigues(rvec_flipped)
realworld_tvec = np.dot(rotation_matrix, tvec_flipped)
```

Therefore, python allows the use of seamless data structures with the need for much declaration of the type of data structure and hence it is to an advantage that python implicitly assigns the data structure that is required to store data ensuring the correct data dictionary is used always.

- **Object Oriented Python development**
  Due to the modular approach of the system and in minimal cases when one module does depend on the other, simple object-oriented principles have been followed so the objects interact with each other to accomplish the end task. A example of object orientation can be seen below:

```
#Estimate Pose Values
def poseDetector(self, inputX, inputY, inputZ, tagSetting, cameraSetting):
    #Set Tag Type
    aruco_dict = cv2.aruco.getPredefinedDictionary(ARUCO_DICT[tagSetting["tagType"]])
```

This function takes in ‘self’ as a parameter meaning it is referring to this instance of the program which is the PoseDetector.

- Writing readable code
  Development standards such as:
  - Line breaks
  - Naming conventions
  - Code indentation
  - Comments

  Were followed throughout the program.

1.7 Help and Maintenance

The application is modularised in terms of executing different features of the system independently for >50% of its functional uses. The Pose Detection module however requires the information from other modules to be able to accurately estimate the pose of an object. A relation diagram is shown below of the dependency of the pose detector module:

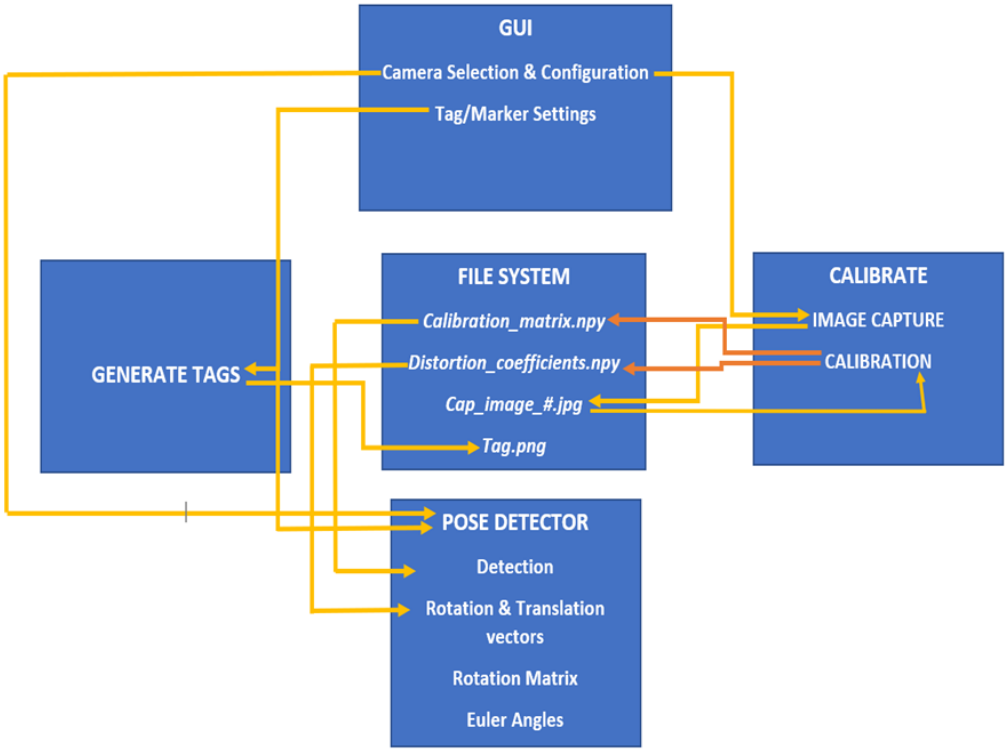

Figure 14 - Dependencies between the modules

The system does not have any known critical issues that were detected during extensive testing. If something unexpected happens which has not occurred previously in which case the developers should be contacted as first contact below:

- Daniel Leong – 102589132@student.swin.edu.au, DSLeong28@gmail.com
- Preenit Kshirsagar – 102486667@student.swin.edu.au
- Dylan Were – 102102160@student.swin.edu.au
- Parth Patel – 101604584@student.swin.edu.au

2 User Documentation

Please refer to the comprehensive user manual provided separately for the application. There are two systems on which the system was designed and built (Windows & Linux (Raspberry Pi)), for which the user manual is provided for.

3 References

He, Z, Feng, W, Zhao, X & Lv, Y 2020, '6D Pose Estimation of Objects: Recent Technologies and Challenges', Applied Sciences, vol. 11, no. 1, p. 228, <https://www.mdpi.com/2076-3417/11/1/228/htm>.

OpenCV n.d., 'OpenCV: Camera Calibration', docs.opencv.org, <https://docs.opencv.org/4.x/dc/dbb/tutorial_py_calibration.html>.

OpenCV n.d., 'OpenCV: Detection of ArUco Markers', docs.opencv.org, <https://docs.opencv.org/4.x/d5/dae/tutorial_aruco_detection.html>.

Flandin, G, Chaumette, F & Marchand, E 2000, 'Eye-in-hand/eye-to-hand cooperation for visual servoing', IEEE Xplore, pp. 2741–2746 vol.3, <https://ieeexplore.ieee.org/document/846442>.

Wikipedia Contributors 2019, 'Rotation matrix', Wikipedia, Wikimedia Foundation, <https://en.wikipedia.org/wiki/Rotation_matrix>.

ARToolKit n.d., 'ARToolKit Coordinate Systems', www.hitl.washington.edu, <http://www.hitl.washington.edu/artoolkit/documentation/cs.htm>.

Wikipedia Contributors 2021, 'CPU time', Wikipedia, viewed 23 October 2022, <https://en.wikipedia.org/wiki/CPU_time>.

Perforce 2019, 'Intro to Coding Standards — Coding Rules and Guidelines | Perforce', Perforce.com, viewed <https://www.perforce.com/resources/qac/coding-standards>.