1 Task 1

The first task was to create a client-service model where the client sends the service a path of the image containing Aruco markers and the service basically processes and sends the back the ID and the corner coordinated back to the client in an array. Since its a custom client-service model, I was supposed to define a new interface package using the command 'build type: c-ament' where the .srv and .msg file are stored for the model to access. So there are two packages, one containing the interface and the other containing the executable client-service python files. The .srv file describes the service. It is divided in two parts that is the request and response separated by —.

```
sensor_msgs/Image image  # Input: the image data
---
int32[] ids              # Output: detected ArUco marker IDs
geometry_msgs/Polygon[] polygon     # Output: bounding box coordinates [x1, y1, x2, y2, ..., xn,
```

The first line tells how the request is an uncompressed image. The response is a simple 1-D array containing the Ids. The bounding box coordinates are stored in the array called polygon which is imported from geometry msgs which to my understanding are nodes of the connected polygon where data is being stored. When I used a normal array to define the coordinates though, there was an error while running the file.
We are creating a bridge between the OpenCV server and our server in order to use the OpenCV commands to detect the marker and its parameters. The tutorial uploaded on the website use outdated commands and that was a significant error in my program. I later learnt how we are supposed to define each parameter as an instance/object of the class and then use the defined commands to actually run them properly. In my service file, the method imgmsg to cv2 is being used to convert the ROS2 image to OpenCV format. The image is also converted to grayscale first because it's required for threshold operations. Grayscale conversion also simplifies algorithms, reduces computational requirements. Later using the detectMarkers method the corners and id are respectively stored in the array I created in the beginning. I also got an error for not importing my srv file to the client file so that is an important thing to remember I guess. Then the program checks if the ids is not an empty set, if its not then it continues to append each id value to the ids array and repeates the same thing for the corners too except there are two parameters this time that is the x and y coordinates. Then the procedure (main) is being defined where first initialiation is done. Then the

class of Aruco Service defined is being instantiating as a node and the work is being done (callbacks, subscriptions etc) using the method rclpy.spin(). In the client file, in the class we are defining a function called send request which converts the open cv image given the image message format to send it to the service. The main function of the client requests an image file path from the user. I assume that the method sys.argv is basically used to extract the manual input given to the client. The imread method is used to load the image from the specified file path. The spin until future complete method executes work under the future is completed. Callbacks and other work will be executed until the future done returns true. I am a little confused on how this works though but I believe the the program keeps spinning till the service sends back a response. I believe this was the basic logic behind how the program ran and I spend a lot of time debugging even though the basic program flow was clear to me is because of the outdated functions and figuring out a way to make the methods work.

2 Task 2

The second task required us to detect the aruco markers from a video and in turn get the ids and the bounding box coordinates from the service. I tried using my webcam to send images however. There were no changes in the .srv and .msg file for this program.
The main addition here is to use the method cv2.VideoCapture() to create a video capture object for the camera in the client file. A frame is being captured and being sent to the service to process the aruco marker. Ret is a boolean variable that returns true if the frame is available. Frame is an image array vector captured based on the default frames per second defined explicitly or implicitly. Cv2.imshow() method is used to show the frames in the video. The service does the same processing as Task1. The only factor that changed in this entire server model is how the client is getting the request.