1.ขั้นตอนการทำงานของโปรแกรมและ การใช้งานโปรแกรม

โปรแกรม GPLidar Monitor

ขั้นตอนการทำงานของโปรแกรม

-Python Version 3.11.3

-Import python library

```python
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from matplotlib.widgets import Button
import matplotlib.animation as animation
import math
import utm
import time
```

-อ่านค่าจากไฟล์ CSV ด้วย pandas

```python
# Read GPS_data from csv file
gps_data = pd.read_csv("GPS_data/gpsPlus_20230612164330.csv")
# Read LIDAR_data from csv file
lidar_data = pd.read_csv("LIDAR_data/ydlidar_20230612164330.csv")
```

-ดึงค่าที่อ่านจาก pandas มาเก็บไว้

```python
# Setup for GPlidar3 function
# status of animation process (play/pause)
status = True
# Get the number of rows from lidar_data
n = lidar_data.shape[0]
# offset heading angle of polar plot
offset = np.pi/2
# Get array of range of lidar_data
r = lidar_data['lidar_range_meter']
# Get array of angle of lidar
theta = lidar_data['lidar_angle_degree']

# Get Lat Long value from gps_data
Lat = gps_data['gps_recentLatitudeN'].to_numpy()
# print(Lat)
Long = gps_data['gps_recentLongitudeE'].to_numpy()
# Get heading angle from gps_data
theta_offset = gps_data['compass_heading_degs'].to_numpy()
```

-แปลงค่า ละติจูดและ ลองจิจูดจาก GPS เป็นพิกัด X, Y ด้วยการแปลงแบบ UTM ซึ่งได้หน่วยเป็นเมตรออกมา

```
38  # Convert Lat Long to UTM (X,Y)
39  # u[0] is East, X) and u[1] is North, Y
40  u = utm.from_latlon(Lat, Long)
```

-ประกาศตัวแปรไว้สำหรับใช้ในการ plot

```
42  # Polar Plot lidar_data
43  # Polar Plot Setting
44  fig = plt.figure(figsize=(11, 6))
45  # define axises for Polar Plot
46  ax1 = plt.subplot(1,2,1, projection='polar')
47  # define axises for position plot
48  ax2 = plt.subplot(1,2,2)
49  # tuple variable for update plot with animation function
50  line1, = ax1.plot([], [], 'ro', markersize=2)
51
52  # variables for update value each frame
53  xpos, ypos = [], [] # ax2
54  polar_theta, polar_r   = [], [] # ax1
55
```

-ฟังก์ชันสำหรับคำนวณมุม offset ของ polar plot ให้แกน theta หมุนทิศ 0 องศาไปตามทิศการเคลื่อนที่ของหุ่นยนต์บนกราฟแสดงตำแหน่งการเคลื่อนที่ของหุ่นยนต์ เพื่อให้การแสดงผลไปในทิศทางเดียวกัน

```
57  def Polar_ax_offset(degree_offset): # this function for offset angle in Polar plot
58      return (90-degree_offset)*np.pi/180
```

-ในฟังก์ชันสำหรับการทำ animation plot จะประกอบไปด้วยส่วนสร้างปุ่มสำหรับกด play/pause, ส่วนการ setup plot เพิ่มเติม และ ส่วนของฟังก์ชันอัพเดทค่าที่ใช้ในการแสดงผลข้อมูล

```
60  def GPlidar_plot3(gps_data, lidar_data): # Run animation plot function
61      # Setup Button
62      axes = plt.axes([0.4, 0.0001, 0.1, 0.075]) # button position
63      bpause = Button(axes, 'Play/Pause', color="yellow") # create button
64
```

```
65  def init():
66      # additional setup plot
67      ax1.set_title("Radius of obstacles from lidar and \n and robot heading Orientation respect to North", fontsize = 8)
68
69      ax2.set_title("Position of robot from GPS")
70      ax2.set_xlabel('Position X (m.)')
71      ax2.set_ylabel('Position Y (m.)')
```

-ในฟังก์ชันอัพเดทข้อมูล จะมีการนำข้อมูล lidar_range_meter และ lidar_angle_degree มาแปลงให้เก็บเป็นตัวเลขให้ได้ก่อน เนื่องจากข้อมูลที่ได้มาเป็นรูปแบบของ string ในบรรทัดที่ 78 และ 85

จากนั้นในการ plot ค่าแต่ละเฟรมจะมีนำค่า lidar_range_meter ที่สูงสุดในเฟรมนั้นมากำหนดขอบเขตแกน r ของ polar plot ในบรรทัดที่ 83

และได้นำ compass_heading_degs จาก GPS มาใช้ในการหมุนแกนให้ด้านหน้าของหุ่นยนต์อยู่ทิศทางเดียวกับกราฟตำแหน่งการเคลื่อนที่ของหุ่นยนต์ ในบรรทัดที่ 89

ในบรรทัดที่ 93 – 97 จะเป็นการอัพเดทค่าสำหรับเตรียม plot ในเฟรมถัดไป

ค่าของ polar plot สำหรับข้อมูล lidar จะถูกส่งไป plot ในบรรทัดที่ 100

ส่วนค่าตำแหน่งการเคลื่อนที่ของหุ่นยนต์จะทำการ plot เองในฟังก์ชันอัพเดท ในบรรทัดที่ 102-109

เนื่องจากหากทำวิธีเดียวกับ กราฟข้อมูลของlidar จะทำให้ scale ของกราฟไม่อัพเดทตามข้อมูลที่เพิ่มเข้าไป เช่นเดียวกับ กราฟข้อมูลของlidarหาก plot ในฟังก์ชันอัพเดทเองจะเกิดการทับกันของค่าใหม่และค่าเก่า

```python
def update(frame): # input is index i of frames in FuncAnimation (This function will be run many times by FuncAnimation)
    # call global variables
    global r, theta, n
    # ax1
    # Convert string to list then Convert to float32 numpy array
    irow_r = np.array(r[frame].strip(']').split(', '), dtype=np.float64)
    # print(irow_r)
    # determine max radius of polar plot for each timestep
    maxr = int(np.max(irow_r))+1
    # set radius limit of polar plot
    ax1.set_rmax(maxr)
    # Convert string to list and Convert float32 numpy array
    irow_theta = np.array(theta[frame].strip(']').split(', '), dtype=np.float64)
    irow_rad = irow_theta*np.pi/180.0
    # additional Polar Plot Setting
    # set theta_offset with respect to gps_data
    ax1.set_theta_offset(Polar_ax_offset(theta_offset[frame]))

    # update variable
    # ax1
    polar_theta = irow_rad
    polar_r = irow_r
    # ax2
    xpos.append(u[0][frame])
    ypos.append(u[1][frame])

    # update line2D
    line1.set_data(polar_theta, polar_r)
    # Position plot
    if frame == 0: # plot ax2 in update function because This axis can't update scale when new data come.
        ax2.plot(xpos, ypos, 'ro', markersize= 9)   # plot initial data as red mark

    elif frame == n-1:
        ax2.plot(xpos[frame], ypos[frame], 'ro', markersize= 9) # plot the lastest data as redmark

    else:
        ax2.plot(xpos, ypos, 'o',color='green', markersize= 4) # plot data with green mark

    # return value for ax1 plot(polar plot)
    return line1 #
```

-จากนั้นโปรแกรมจะทำการเรียกใช้ฟังก์ชันสำหรับ animation plot

```python
# initial additionadl plot setup
init()

# input of update is Line2D or List of Line2D
# Blitting changes the content of the axes, not the decorators so set it to False
print('n = ',n)
ani = animation.FuncAnimation(fig, update, frames=np.arange(0, n, 1),interval=1000,
                              repeat= False, blit=False) # interval unit is milliseconds, blit is False so the setup of axises can be updated
```

-เมื่อกดปุ่ม Play/Pause ในหน้าต่าง plot ปุ่มจะเรียกใช้ฟังก์ชัน pause_ani เพื่อเช็ค status ของโปรแกรมแล้วสั่งให้โปรแกรมหยุดหรือเล่นต่อไป

```
def pause_ani(event): # fuction which use with pause button
    global status
    if status :
        print('True = ',status)
        ani.pause()
    else:
        ani.resume()
    status = not status # change value of the status variable
    print('out', status)

# run callback funtion when button is onclick
bpause.on_clicked(pause_ani)
# show the plot
plt.show()
```

การใช้งานโปรแกรม

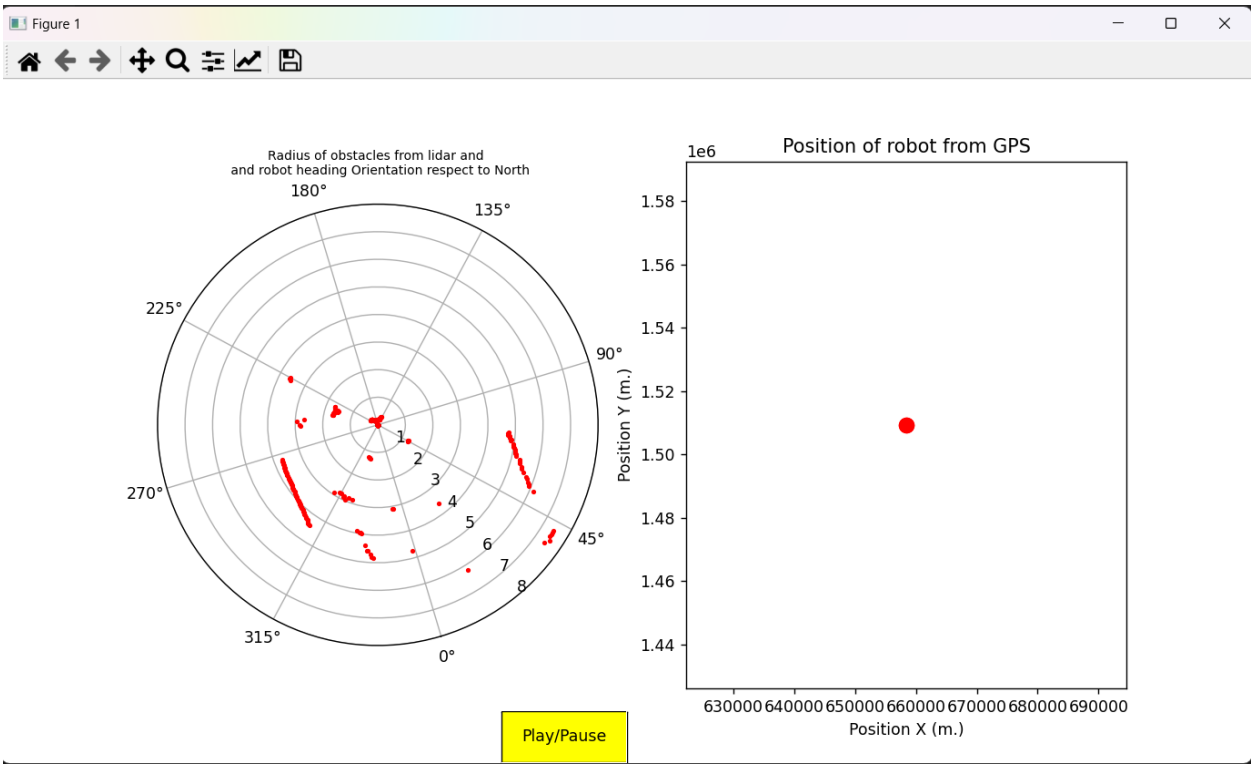
-โปรแกรมจะเป็นไฟล์ python ชื่อว่า GPLidar_monitor.py

-ให้นำไฟล์ csv ที่เป็นข้อมูลของ GPS ใส่ในโฟลเดอร์ GPS_data และนำข้อมูลของ LIDAR ใส่ในโฟลเดอร์ LIDAR_data

-สามารถปรับความเร็วของการรัน animation ได้โดยปรับที่ตัวแปร interval ของฟังก์ชัน FuncAnimation โดยมีหน่วยเป็น millisecond

```
114     # initial additionadl plot setup
115     init()
116
117     # input of update is Line2D or List of Line2D
118     # Blitting changes the content of the axes, not the decorators so set it to False
119     print('n = ',n)
120     ani = animation.FuncAnimation(fig, update, frames=np.arange(0, n, 1),interval=1000,
121                                   repeat= False, blit=False) # interval unit is milliseconds, blit is False so the setup of axises can be updated
122
```

-เมื่อกดรัน โปรแกรมจะขึ้นหน้าต่างสำหรับ plot และโปรแกรมจะเริ่มการ plot ค่า



-สามารถกดปุ่ม Play/Pause สีเหลืองเพื่อทำโปรแกรมหยุดทำงานชั่วคราว หรือให้โปรแกรมทำงานต่อ

-เมื่อโปรแกรมแสดงผลจนครบทุกข้อมูลโปรแกรมจะหยุดทำงานแล้วแสดงผลลัพธ์สุดท้ายไว้ และ ปุ่ม Play/Pause จะใช้งานไม่ได้

-หากต้องการปิดโปรแกรมก็สามารถปิดหน้าต่างที่ชื่อ Figure 1 ได้เลย

2. จงอธิบาย<u>ขั้นตอนโดยละเอียด</u> (ไม่ต้องเขียน code) ของวิธีการพัฒนาโปรแกรมเพื่อแสดงระยะและทิศทางระหว่างหุ่นยนต์กับป้าย ARUCO code ว่าต้องทำขั้นตอนอะไรบ้าง ตั้งแต่การset upกล้อง การcalibrate การประมวลผลภาพ การทดสอบความแม่นยำ ว่าต้องใช้กระบวนการ และ library หรือ function อะไรบ้าง

Library

-Numpy

-Opencv (cv2)

-matplotlib

-argparse

-imutils

-time

ขึ้นตอนการทำ

1) การเซ็ทอัพกล้องที่จับภาพ Aruco

ทำการติดตั้งกล้องไว้ในตำแหน่งที่สามารถถ่ายภาพ Aruco ได้ชัดเจนไม่มีสิ่งกีดขวางมาบังระหว่างกล้องกับ Aruco และ ต้องรู้ขนาดจริงของป้าย Aruco เพื่อใช้สำหรับโปรแกรมในการหาระยะและทิศทางระหว่างหุ่นยนต์กับป้าย Aruco

การเซ็ทอัพกล้องใน OpenCV เพื่อเก็บภาพ

- ใช้งานฟังก์ชัน cv2.VideoCapture() ในสร้าง object มาเปิดใช้งานกล้อง

- ใช้ฟังชัน object.read() เพื่อเก็บค่าภาพจากกล้อง

- ใช้ฟังก์ชัน cv2.imshow() เพื่อแสดงผลลัพธ์ภาพที่เก็บได้

-ใช้ฟังก์ชัน cv2.imwrite() บันทึกภาพลงคอมพิวเตอร์

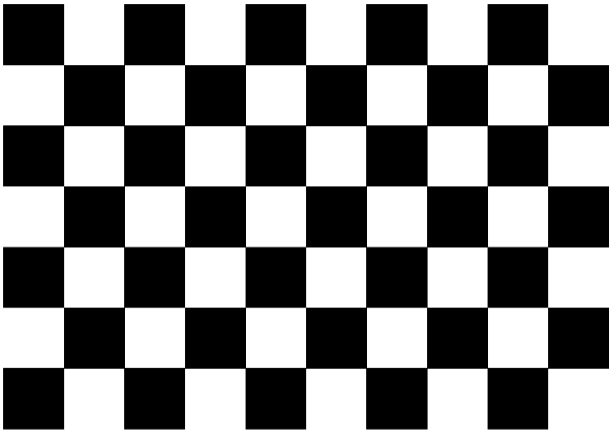
-ใช้ฟังก์ชัน cv2.waitkey() เพื่อหยุดรอเช็คว่ามีการกดค่าที่คีบอร์ด เพื่อใช้ในการ break loop แล้วปิดโปรแกรม

-ใช้ฟังก์ชัน object.release() และ cv.destroyAllWindows() หลัง break loop เพื่อปิดโปรแกรม

-หากใช้ VdoStream ของ imutils จะใช้ object ของ Vdosteam.read() แทนของ cv2 และ Objectของ cv2.release() จะใช้เป็น object ของ Vdosteam.stop() แทน

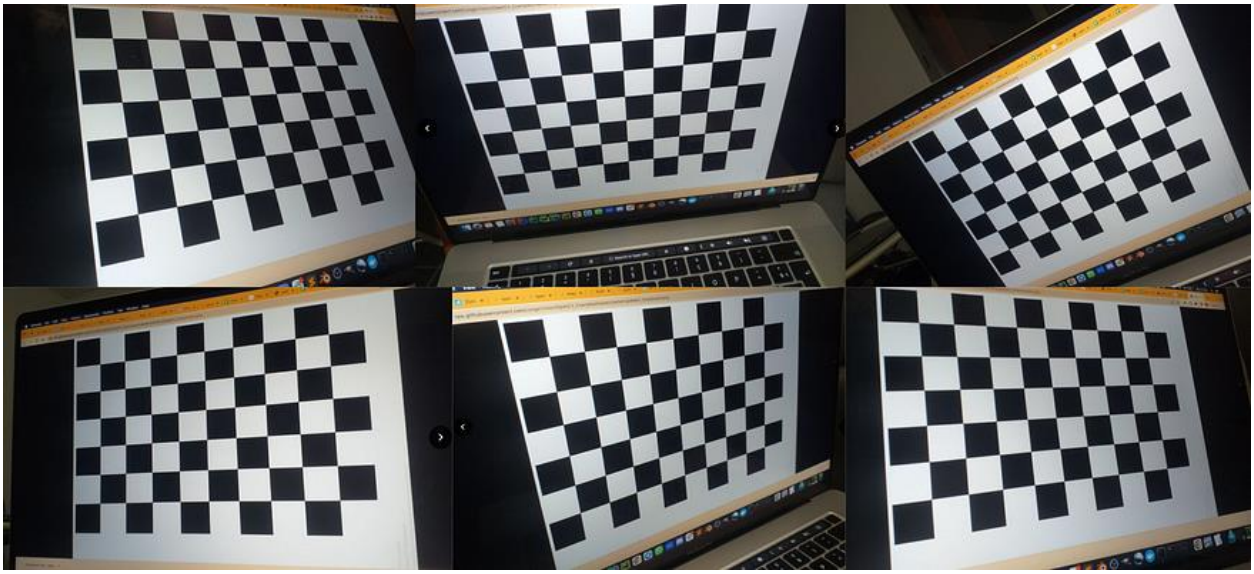
2) การทำ Camera Calibration

-ปริ้นรูป chessboard ออกมา หรือเปิดในจอคอมพิวเตอร์เตรียมพร้อมที่จะใช้กล้องถ่าย



รูปตารางหมากรุกที่นำมาใช้ calibration

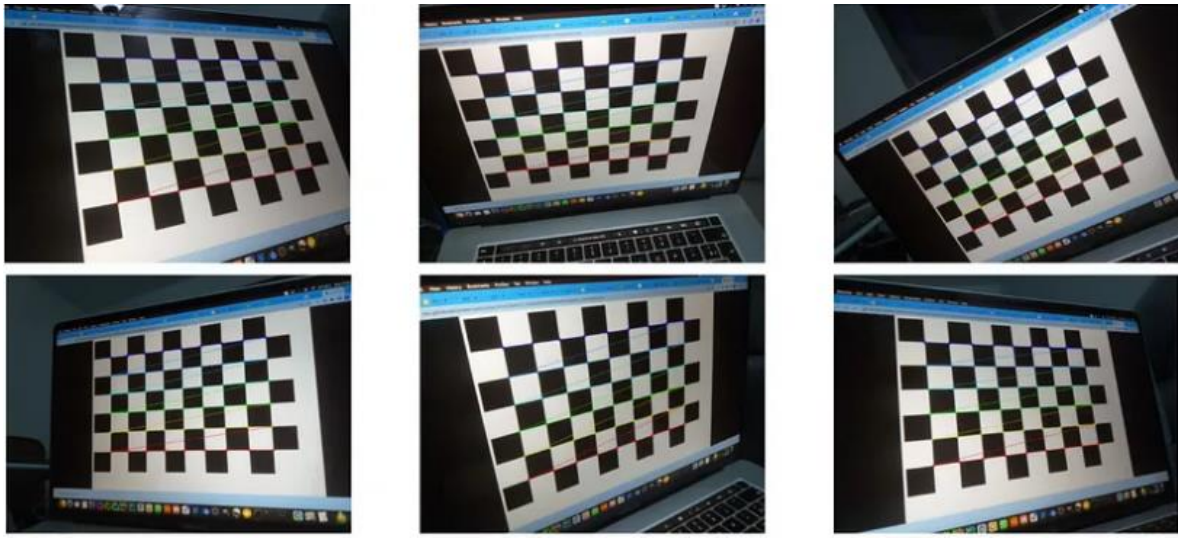
-ทำการถ่ายภาพกระดานหลายๆมุมเพื่อให้ calibration ได้แม่นยำมากขึ้น



-ใช้ฟังก์ชัน cv2.imread() โหลดรูปภาพที่ถ่ายเข้ามาในโปรแกรม

-ใช้ฟังก์ชัน cv2.findChessboardCorners() เพื่อให้โปรแกรมหาตารางในรูปเจอโดยระบุขนาดของตารางให้

-จากนั้นใช้ฟังก์ชัน cv2.drawChessboardCorners() เพื่อวาดเส้นแสดงมุมลงตารางที่อยู่ในรูป ถ้าโปรแกรมตรวจจับตารางได้ จะได้ imagepoints และ objpoints ของรูปตารางมา



รูปหลังการใช้งานฟังก์ชัน drawChessboardCorners()

-ใช้ฟังก์ชัน cv2.calibrateCamera() จะได้ Intrinsic parameters ของกล้องออกมา

3) การหาระยะห่างและทิศทางของป้าย Aruco เทียบกับหุ่นยนต์

Aruco detection

-ใช้ฟังก์ชัน argparse.ArgumentParser() สร้าง object ขึ้นมา

-ใช้ฟังก์ชัน object.add_argument() เพื่อให้เก็บค่าประเภทของ aruco

-ใช้ฟังก์ชัน cv2.aruco.DICT.... มาเรียก aruco แต่ละชนิดที่มีมาเก็บเพื่อใช้ detection

-ใช้ฟังก์ชัน cv2.aruco.Dictionary_get() คู่กับ object ของ argparse ที่สร้างขึ้นเก็บค่า arucoDict

-ใช้ฟังก์ชัน cv2.aruco.DetectorParameters_create() เรียค่า arucoParameters มาเก็บ

-ใช้ฟังก์ชัน VideoStream().start() ของ imutils เพื่อใช้งานกล้อง

-ใช้งานฟังก์ชัน time.sleep() เพื่อรอ VideoStream พร้อมใช้งาน

-สร้างตัวแปรมาเก็บค่า distortion และ camera matrix ที่ได้จาก camera calibration มาเก็บไว้

-ใช้ฟังก์ชัน VideoStream.read() เพื่อเก็บภาพจากวีดีโอ

-ใช้ฟังก์ชัน cv2.aruco.detectMarkers() โดยมีภาพที่เก็บได้จากกล้อง, arucoDict และ arucoParameters เป็น input เพื่อตรวจจับป้าย aruco ฟังก์ชันจะคืนค่า id ของ aruco และ ตำแหน่งแต่ละมุมของป้าย aruco

Estimate the pose of the marker.

-ขั้นตอนนี้ต้องการ 2 parameters จากการทำ calibration คือ cameraMatrix และ distortion Coefficients และ corners (ตำแหน่งแต่ละมุมของป้าย aruco) ที่ได้จากขั้นตอน Detection

-ต้องรู้ขนาดที่แท้จริงของป้าย aruco ที่จะตรวจจับโดยหน่วยที่ออกมาเป็นผลลัพธ์ของ pose จะเป็นหน่วยเดียวกับที่กำหนดเข้าไป

- ใช้ฟังก์ชัน aruco.estimatePoseSingleMarkers() เพื่อหาระยะและทิศทางระหว่างหุ่นยนต์กับป้าย Aruco จะได้ผลลัพธ์อยู่ในรูป rotation vector และ translation vector ที่เทียบกับกล้องบนหุ่นยนต์

4) การทดสอบความแม่นยำของโปรแกรม

วัดความแม่นยำของระยะห่างด้วยเครื่องมือวัด

-ทำการวัดค่าระยะห่างระหว่างหุ่นยนต์และป้ายAruco ด้วยเครื่องมือวัดในโลกจริง หรือใน Simulation เพื่อเปรียบเทียบกับค่าที่โปรแกรมคำนวณได้หากมีความคลาดเคลื่อนมากเกินไปให้ลองปรับแก้ด้วยการทำ camera calibration ของกล้องใหม่

ทดสอบความแม่นยำในขั้นตอน calibration ได้ด้วยการทำ Reprojection error

-ทำการคำนวณค่าเฉลี่ยแบบ Euclidian ระหว่าง imgpoints ตอนก่อนทำ calibration กับ imgpoints ที่ได้จากการใช้ฟังก์ชัน cv2.projectPoints() ด้วย camera matrix และ distortion parameters ที่ทำ calibration มา โดยค่าเฉลี่ย Reprojection error ที่พอรับได้อยู่ที่ 0.5-0.6

ตรวจสอบความถี่ที่โปรแกรมประมวลผลต่อเฟรมได้ (fps)

-ใช้ time library ในการสร้าง timer โดยใช้ฟังก์ชัน time.time() เพื่อเก็บค่าเวลาตอนเริ่มและสิ้นสุด

-กำหนดให้โปรแกรม Pose estimation ทำงานวนลูปเท่ากับค่าจำนวนเฟรมที่ต้องการจะวัดผล

-fps $= \frac{\text{จำนวนเฟรมที่ต้องการ}}{stopTime - startTime}$

อ้างอิง:

Simon (2021), How can I get the distance from my camera to an OpenCV ArUco Marker. Retrieved 20 June 2023, //////// from https://stackoverflow.com/questions/68019526/how-can-i-get-the-distance-from-my-camera-to-an-opencv-aruco-marker

Luh (2012), lat/lon to utm to lat/lon is extremely flawed, how come?. Retrieved 22 June 2023, //////// from https://stackoverflow.com/questions/6778288/lat-lon-to-utm-to-lat-lon-is-extremely-flawed-how-come

Matplotlib (2012-2023), Pausing and Resuming an Animation. Retrieved 23 June 2023, //////// from https://matplotlib.org/stable/gallery/animation/pause_resume.html

Mv93 (2015), animated subplots using matplotlib. Retrieved 23 June 2023, //////// from https://stackoverflow.com/questions/29832055/animated-subplots-using-matplotlib

Michel Rodrigues Andrade (2018), Python animation polar plot. Retrieved 23 June 2023, //////// from https://stackoverflow.com/questions/50748833/python-animation-polar-plot

ImportanceOfBeingErnest (2019), Matplotlib animation update X-axis limit doesn't work. Retrieved 23 June 2023, //////// from https://stackoverflow.com/questions/56720767/matplotlib-animation-update-x-axis-limit-doesnt-work

Adrian Rosebrock (2020), Detecting ArUco markers with OpenCV and Python. Retrieved 24 June 2023, //////// from https://pyimagesearch.com/2020/12/21/detecting-aruco-markers-with-opencv-and-python/

KaranGupta5 (2023), Python OpenCV: Capture Video from Camera. Retrieved 24 June 2023, //////// from https://www.geeksforgeeks.org/python-opencv-capture-video-from-camera/

Evgenii Munin (2023) Camera Calibration on a Chessboard With Python and OpenCV. Retrieved 24 June 2023, //////// from https://betterprogramming.pub/camera-calibration-on-a-chessboard-with-python-and-opencv-78bb155319cf

Jes Fink-Jensen (2022), How To Calibrate a Camera Using Python And OpenCV. Retrieved 24 June 2023, //////// from https://betterprogramming.pub/how-to-calibrate-a-camera-using-python-and-opencv-23bab86ca194

Longer Vision Technology (2023), Camera Calibration Using a Chessboard. Retrieved 24 June 2023, //////// from https://longervision.github.io/2017/03/16/ComputerVision/OpenCV/opencv-internal-calibration-chessboard/

Satya Mallick (2015), Find frame rate (frames per second-fps) in OpenCV (Python/C++). Retrieved 25 June 2023, //////// from https://learnopencv.com/how-to-find-frame-rate-or-frames-per-second-fps-in-opencv-python-cpp/