cameraCalibration.py

```python
# credit to https://learnopencv.com/camera-calibration-using-opencv/
# https://docs.opencv.org/4.x/dc/dbb/tutorial_py_calibration.html
import cv2
import numpy as np
import os
import glob

Checkerboard = (6, 9)
criteria = (cv2.TERM_CRITERIA_EPS + cv2.TERM_CRITERIA_MAX_ITER, 30, 0.001)
objp = np.zeros((Checkerboard[0] * Checkerboard[1], 3), np.float32)
objp[:, :2] = np.mgrid[0:Checkerboard[0], 0:Checkerboard[1]].T.reshape(-1, 2)

objectPoints = []
imagePoints = []

gray = None


files = glob.glob("./OpenCV-APCSP_Project/assets/iloveimg-resized/*.png")

for file in files:
    img = cv2.imread(file)
    gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
    ret, corners = cv2.findChessboardCorners(
        gray, Checkerboard, cv2.CALIB_CB_ADAPTIVE_THRESH + cv2.CALIB_CB_FAST_CHECK + cv2.CALIB_CB_NORMALIZE_IMAGE)

    if ret == True:
        objectPoints.append(objp)
        cornersTwo = cv2.cornerSubPix(
            gray, corners, (11, 11), (-1, -1), criteria)
        imagePoints.append(cornersTwo)

cv2.destroyAllWindows()


ret, mtx, dist, rvecs, tvecs = cv2.calibrateCamera(
    objectPoints, imagePoints, gray.shape[::-1], None, None)


def getProjectionError():
    mean_error = 0
    for i in range(len(objectPoints)):
        imgpoints2, _ = cv2.projectPoints(
            objectPoints[i], rvecs[i], tvecs[i], mtx, dist)
        error = cv2.norm(imagePoints[i], imgpoints2,
                         cv2.NORM_L2)/len(imgpoints2)
        mean_error += error
    print("total error: {}".format(mean_error/len(objectPoints)))


# ret, mtx, dist, rvecs, tvecs = cv2.calibrateCamera(objectPoints, imagePoints, gray.shape[::-1], None, None)

# print(mtx)
# print(dist)
# print(rvecs)
# print(tvecs)
```

main.py

```
import keyboard as key
import detectCone
import detectCube
import cameraCalibration as calib


while True:
    if key.is_pressed('u'):
        detectCube.run()
        break
    elif key.is_pressed('o'):
        detectCone.run()
        break
```

detectCube.py

```
import cv2
import numpy as np
import numpy.linalg as lin
import cameraCalibration as calib
import keyboard as key
import linearTrajectory as traj
import matplotlib.pyplot as plot

# used to control what color the camera should be looking, this interval can detect, say a purple cube.
# hopefully I can use a trained HaarCascadeClasifier xml for better tracking.
low = np.array([128, 50, 128])
high = np.array([255, 255, 255])
# used to blur images if camera gets too close to object, matrix computes weighed average of each pixel by matrix multiplication
# opencv tracks objects way better when image(s) are blurred
# according to https://en.wikipedia.org/wiki/Kernel_(image_processing)
# a 5*5 Gaussian matrix provides most blur
kernelMatrix = np.multiply(1/256, np.array([
    [1, 4, 6, 4, 1],
    [4, 16, 24, 16, 4],
    [6, 24, 36, 24, 6],
    [4, 16, 24, 16, 4],
    [1, 4, 6, 4, 1]]))

cubePointsInches = np.array(
    [(0, 0, 0), (0, 9.5, 0), (9.5, 9.5, 0), (9.5, 0, 0)])
axis = np.float32([[0, 0, 0], [0, 9.5, 0], [9.5, 9.5, 0], [9.5, 0, 0], [
                  0, 0, -9.5], [0, 9.5, -9.5], [9.5, 9.5, -9.5], [9.5, 0, -9.5]])

dilationKernel = np.ones((5, 5), np.uint8)

mtx = calib.mtx
dist = calib.dist
tvecs = calib.tvecs
rvecs = calib.rvecs


def distance(objectDimensions, focalLength_mm, objectImageSensor):
    distanceInches = (objectDimensions * focalLength_mm/objectImageSensor)/25.4
    return distanceInches


def getPose(contours):
    largest_contour = max(contours, key=cv2.contourArea)
    (x, y, w, h) = cv2.boundingRect(largest_contour)
    imagePoints = np.array(
        [(x, y), (x, y+h), (x+w, y+h), (x+w, y)], dtype=np.float32)
    ret, rvec, tvec, inliers = cv2.solvePnPRansac(
        cubePointsInches, imagePoints, mtx, dist, iterationsCount=100, reprojectionError=2.00, confidence=0.9, flags=cv2.SOLVEPNP_ITERATIVE)
    rvec2, tvec2 = cv2.solvePnPRefineLM(
        cubePointsInches, imagePoints, mtx, dist, rvec, tvec)
    rvec2, _ = cv2.Rodrigues(rvec2)
    return rvec2, tvec2, inliers


def drawBox(img, corners, imgpts, color):
    imgpts = np.int32(imgpts).reshape(-1, 2)
    img = cv2.drawContours(img, [imgpts[:4]], -1, color, -3)
    for i, j in zip(range(4), range(4, 8)):
        img = cv2.line(img, tuple(imgpts[i]), tuple(imgpts[j]), color, 3)
    img = cv2.drawContours(img, [imgpts[4:]], -1, color, 3)
    return img


def run():
    cap = cv2.VideoCapture(0)
    while True:
        ret, frame = cap.read()
    # can use GaussianBlur function, but want to modify with matrix
        contrast = cv2.convertScaleAbs(frame, 0, 1.25)
        filter = cv2.GaussianBlur(contrast, (11, 11), 0)
        convert = cv2.cvtColor(filter, cv2.COLOR_BGR2HSV)
        range = cv2.inRange(convert, low, high)
        range = cv2.morphologyEx(range, cv2.MORPH_OPEN, dilationKernel)
    # unused
        contours, _ = cv2.findContours(
            range, cv2.RETR_TREE, cv2.CHAIN_APPROX_NONE)

        for i in contours:
            (x, y, w, h) = cv2.boundingRect(i)
            if cv2.contourArea(i) > 150:
                cv2.rectangle(filter, (x, y), (x+w, y+h), (255, 0, 0), 2)

        if contours or len(contours) > 0:
            pose = getPose(contours)

            cv2.putText(filter, str(pose[1]), (50, 100),
                        cv2.FONT_HERSHEY_COMPLEX, 0.25, (0, 255, 0), 1)
            cv2.putText(filter, str([np.degrees(angle) for angle in pose[0]]), (50, 200),
                        cv2.FONT_HERSHEY_COMPLEX, 0.25, (0, 255, 0), 1)

            imagePoints, jacobian = cv2.projectPoints(
```

```
                axis, pose[0], pose[1], mtx, dist)

            correctRvec = correctRotation(
                pose[0], pose[1], cap, pose[2], 2, jacobian)[2]

            secondImagePoints, jacobian = cv2.projectPoints(
                axis, correctRvec, pose[1], mtx, dist)

            cv2.drawFrameAxes(filter, mtx, dist, pose[0], pose[1], 20, 10)
            drawBox(filter, axis, secondImagePoints, (255, 0, 0))

            points = np.array(
                [(0, 0), (pose[1][0].item(), pose[1][2].item())], dtype=np.float32)
            coefficents = traj.generateLinearTrajectory(points)
            traj.draw(coefficents, points)

        cv2.imshow("cube video", filter)
        if cv2.waitKey(1) == ord('q'):
            break

    cap.release()
    cv2.destroyAllWindows()

# correct for wrong rotation brought on by limitations of perspective n'perspective model (flipped rvec signs)


def correctRotation(measurement, tvec, cap, poseInliers, minKalmanInliers, jacobian):

    kalman_filter = cv2.KalmanFilter(9, 3, 0)
    if cap.isOpened():
        dt = cap.get(cv2.CAP_PROP_POS_MSEC)/1000
        if not cap.isOpened():
            dt = 0.01
    if (measurement.shape == (3, 3)) or (poseInliers.shape[0] >= minKalmanInliers):

        measurement, _ = cv2.Rodrigues(measurement)
        measurement = measurement.astype(np.float32)
        measurementMatrix = np.eye(3, 9, dtype=np.float32)
        transitionMatrix = np.array([[1, 0, 0, dt, 0, 0, 0, 0, 0],
                                     [0, 1, 0, 0, dt, 0, 0, 0, 0],
                                     [0, 0, 1, 0, 0, dt, 0, 0, 0],
                                     [0, 0, 0, 1, 0, 0, 0, 0, 0],
                                     [0, 0, 0, 0, 1, 0, 0, 0, 0],
                                     [0, 0, 0, 0, 0, 1, 0, 0, 0],
                                     [0, 0, 0, 0, 0, 0, 1, dt, 0],
                                     [0, 0, 0, 0, 0, 0, 0, 1, dt],
                                     [0, 0, 0, 0, 0, 0, 0, 0, 1]], dtype=np.float32)

        processNoiseCov = np.eye(9, dtype=np.float32) * 1e-6
        measurementNoiseCov = np.eye(3, dtype=np.float32) * 1e-3
        errorCovPre = np.ones((9, 9), dtype=np.float32)
        statePre = np.zeros((9, 1), dtype=np.float32)
        jacobianRot = jacobian[:9, :3]
        multiple = np.dot(jacobianRot, measurementNoiseCov)

        X = np.dot(multiple, jacobianRot.T)
        X = X.astype(np.float32)
        errorCovPost = X

        statePost = np.zeros((9, 1), dtype=np.float32)

        kalman_filter.measurementMatrix = measurementMatrix
        kalman_filter.transitionMatrix = transitionMatrix
        kalman_filter.processNoiseCov = processNoiseCov
        kalman_filter.measurementNoiseCov = measurementNoiseCov
        kalman_filter.errorCovPre = errorCovPre
        kalman_filter.errorCovPost = errorCovPost
        kalman_filter.statePre = statePre
        kalman_filter.statePost = statePost

        for _ in range(1000):
            prediction = kalman_filter.predict()
            kalman_filter.correct(measurement)
            estimate = kalman_filter.correct(measurement)
            kalman_filter.errorCovPost = errorCovPost

        final_estimate = prediction[:3, :3]
        final_estimate = final_estimate.astype(type(tvec[0][0]))

        second_final_estimate = kalman_filter.statePost[:3, :3]
        second_final_estimate = second_final_estimate.astype(type(tvec[0][0]))

        third_final_estimate = estimate[:3, :3]
        four = kalman_filter.statePre[:3, :3]

        return final_estimate, second_final_estimate, third_final_estimate, four
```

detectCone.py

```
import cv2
import numpy as np
import cameraCalibration as calib


# used to control what color the camera should be looking, this interval can detect, say a yellow cone.
# hopefully I can use a trained HaarCascadeClasifier xml for better tracking.
low = np.array([0, 100, 200])
high = np.array([50, 255, 255])
# used to blur images if camera gets too close to object, matrix computes weighed average of each pixel by matrix multiplication
# opencv tracks objects way better when image(s) are blurred
# according to https://en.wikipedia.org/wiki/Kernel_(image_processing)
# a 5*5 Gaussian matrix provides most blur
kernelMatrix = np.multiply(1/256, np.array([
    [1, 4, 6, 4, 1],
    [4, 16, 24, 16, 4],
    [6, 24, 36, 24, 6],
    [4, 16, 24, 16, 4],
    [1, 4, 6, 4, 1]]))


dilationKernel = np.ones((5, 5), np.uint8)

conePointsInches = np.array([(0, 0, 0), (4.1875, 0.25, 0), (
    8.375, 0.25, 0), (4.1875, 12.8125, 0)], dtype=np.float32)

mtx = calib.mtx
dist = calib.dist
tvecs = calib.tvecs
rvecs = calib.rvecs

# find the xy(later z) coordinates of an tracked object relative to the camera.


def getPose(contours):
    largest_contour = max(contours, key=cv2.contourArea)
    (x, y, w, h) = cv2.boundingRect(largest_contour)
    imagePoints = np.array(
        [(x, y + h), ((x+w)/2, y+h), (x+w, y+h), ((x+w)/2, y)], dtype=np.float32)
    ret, rvec, tvec = cv2.solvePnP(
        conePointsInches, imagePoints, mtx, dist, cv2.SOLVEPNP_ITERATIVE)
    return rvec, tvec


def getContourCorners(contours):
    intersections = []
    if len(contours) > 1:
        for i in range(0, len(contours) - 1):
            corners = cv2.intersectConvexConvex(contours[i], contours[i+1])
            intersections.append(corners)

    return intersections


def run():
    cap = cv2.VideoCapture(0)
    # detecting yellow requires higher exposure
    cap.set(cv2.CAP_PROP_EXPOSURE, 0.5)
    while True:
        ret, frame = cap.read()
        exposure = cv2.convertScaleAbs(frame, dst=0, alpha=1.25)
    # can use GaussianBlur function, but want to modify with matrix
        filter = cv2.GaussianBlur(exposure, (5, 5), 0)
        convert = cv2.cvtColor(filter, cv2.COLOR_BGR2HSV)
        range = cv2.inRange(convert, low, high)
        range = cv2.morphologyEx(range, cv2.MORPH_OPEN, dilationKernel)
    # unused
        ret, threshold = cv2.threshold(range, 150, 200, cv2.THRESH_BINARY)

        contours, hierarchies = cv2.findContours(
            range, cv2.RETR_TREE, cv2.CHAIN_APPROX_NONE)

        for i in contours:
            (x, y, w, h) = cv2.boundingRect(i)
            if cv2.contourArea(i) > 175:
                cv2.rectangle(filter, (x, y), (x+w, y+h), (255, 0, 0), 2)

        if contours or len(contours) > 0:
            pose = getPose(contours)
            cv2.drawFrameAxes(filter, mtx, dist, pose[0], pose[1], 20, 10)
            cv2.putText(filter, str(pose[0]), (0, 50),
                        cv2.FONT_HERSHEY_COMPLEX, 1, (0, 255, 0), 1)
            print(pose[0])
```

```
        cv2.imshow("cone video", filter)
        # cv2.imshow("exposure", cv2.convertScaleAbs(filter, dst = 1.5, alpha=1.43))

        if cv2.waitKey(1) == ord('q'):
            break

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

linearTrajectory.py

```
import matplotlib.pyplot as plot
import numpy as np
import math
from matplotlib.animation import FuncAnimation
import numpy.linalg as lin

# generate a linear trajectory from a set of points using polynomial regression
# returns coefficents of the linear function of points, and the angle of the line relative to the postive
# x, and positive y axes. All these form a 1d tuple of (coefficent1, coefficent 2, theta1, theta).
# where coefficents1 and 2 represents the equation y=mx+b, m is coefficent1, b is coefficent2.


def generateLinearTrajectory(points):
    secondList = points
    x = [secondList[i][0] for i in range(len(secondList))]
    y = [points[i][1] for i in range(len(points))]
    a, b = np.polyfit(x, y, 1)
    theta = np.degrees(np.arctan(a))
    secondTheta = 90 - theta
    return a, b, theta, secondTheta

# simulate linear trajectories between points using matplotlib graph


def draw(coefficents, points):
    yList = []
    x_points = np.linspace(points[0][0], points[1][0], 50)

    for x in x_points:
        elementsY = (coefficents[0] * x) + coefficents[1]
        yList.append(elementsY)

    fig = plot.gcf()
    ax = fig.gca()
    ax.set_xlim(-100, 100)
    ax.set_ylim(-100, 100)
    line = ax.plot(points[0][0], points[0][1])[0]
    line.set_data(x_points, yList)
    ax.relim()
    anim = FuncAnimation(
        fig, func=line, frames=np.arange(0, 100),  interval=100)
    plot.pause(0.01)
```