**疲劳度检查**

**汇报总结**

**首都师范大学**

**信息工程学院**

**三组**

**项东**

**1201002034**

目录

[一、 疲劳度检查实验说明 1](#_Toc16556)

[二、 疲劳度检查算法和数据集介绍 2](#_Toc4736)

[三、 代码和环境说明 3](#_Toc6409)

[四、 个人总结 5](#_Toc32598)

* 1. **疲劳度检查实验说明**

1. 疲劳度检测的意义所在：

随着世界经济的快速发展，汽车保有量与日俱增，由驾驶员疲劳驾驶造成的交通事故也越来越多，为了保障行驶安全和预防交通事故的发生，研究一种能有效检测驾驶员疲劳并及时给出报警的方法有着重要的现实意义。

1. 疲劳度检测的研究方向

经查阅相关文献，疲劳在人体面部表情中表现出大致三个类型：

打哈欠（嘴巴张大且相对较长时间保持这一状态）;眨眼（或眼睛微闭，此时眨眼次数增多，且眨眼速度变慢）;点头（瞌睡点头）。

所以我们可以通过人脸朝向、位置、瞳孔朝向、眼睛开合度、眨眼频率、瞳孔收缩率等数据入手，对人的疲劳度进行检测。

* 1. **疲劳度检查算法和代码介绍**

1. 算法结构

目前算法用到了MTCNN网络和CNN网络，主要进行眨眼检测。

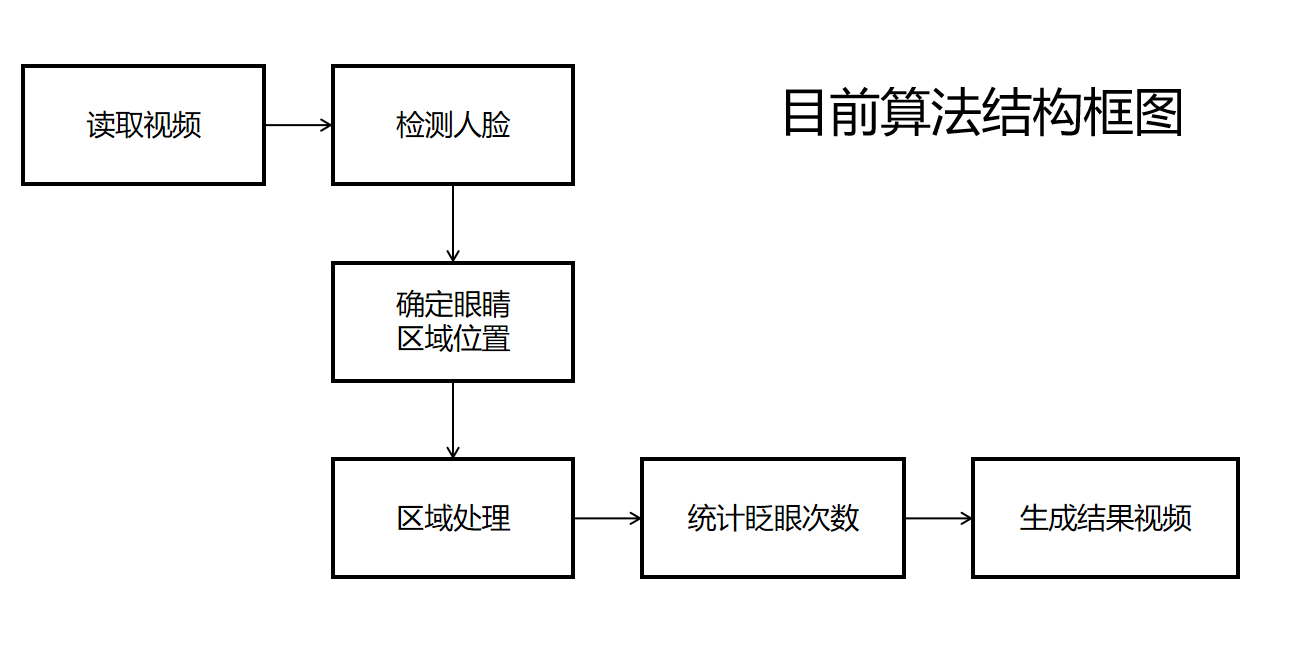
其中还用到了以下库：

OpenCV库：用于图像处理和计算机视觉。

math：Python内置库，用于数学运算。

other\_func：自定义模块，其中包含了多个函数，用于获取多个MER（眼部区域）的信息和判断眼睛状态。

model：自定义模块，用于加载和调用mtcnn（多任务级联卷积神经网络）模型。



新的算法可以利用人脸特征点进行实时疲劳驾驶检测。首先对驾驶员驾驶时的面部图像进行实时监控，检测人脸，并利用ERT算法定位人脸特征点；然后根据人脸眼睛区域的特征点坐标信息计算眼睛纵横比EAR来描述眼睛张开程度，根据合适的EAR阈值可判断睁眼或闭眼状态;最后基于EAR实测值和EAR阈值对监控视频计算闭眼时间比例(PERCLOS)值度量驾驶员主观疲劳程度，将其与设定的疲劳度阈值进行比较即可判定是否疲劳驾驶。新的算法使用了Dlib人脸识别库。Dlib包含了不少的机器学习的成熟算法与模型，相对于tensorflow和PyTorch，它用于图像处理以及人脸面部特征提取、分类及对比这几个方面比较具有通用性和优越性，因此，Dlib正在越来越广泛地应用在人脸识别技术领域。

具体的算法包括眼睛检测、打哈欠检测和点头检测。

眼睛检测算法：基于EAR算法的眨眼检测，当人眼睁开时，EAR在某个值域范围内波动，当人眼闭合时，EAR迅速下降，理论上接近于0。当EAR低于某个阈值时，眼睛处于闭合状态；当EAR由某个值迅速下降至小于该阈值，再迅速上升至大于该阈值，则判断为一次眨眼。为检测眨眼次数，需要设置同一次眨眼的连续帧数。眨眼速度较快，一般1～3帧即可完成眨眼动作。闭眼次数为进入闭眼、进入睁眼的次数。通过设定单位时间内闭眼次数、闭眼时间的阈值判断人是否已经疲劳了。

眼睛检测部分的代码：

from scipy.spatial import distance as dist

from imutils.video import FileVideoStream

from imutils.video import VideoStream

from imutils import face\_utils

import numpy as np # 数据处理的库 numpy

import argparse

import imutils

import time

import dlib

import cv2

import math

import time

from threading import Thread

def eye\_aspect\_ratio(eye):

# 垂直眼标志（X，Y）坐标

A = dist.euclidean(eye[1], eye[5]) # 计算两个集合之间的欧式距离

B = dist.euclidean(eye[2], eye[4])

# 计算水平之间的欧几里得距离

# 水平眼标志（X，Y）坐标

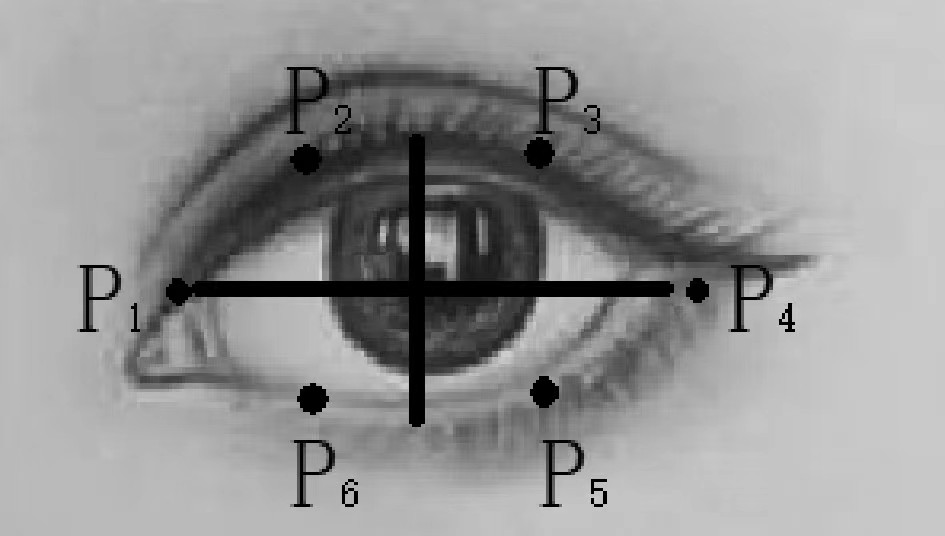
C = dist.euclidean(eye[0], eye[3])

# 眼睛长宽比的计算

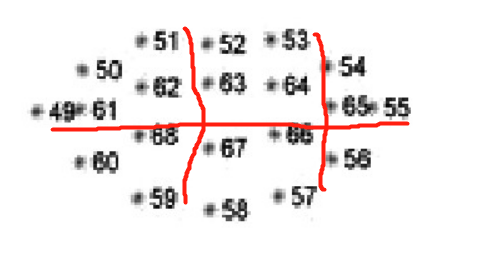
ear = (A + B) / (2.0 \* C)

# 返回眼睛的长宽比

return ear



打哈欠检测算法：基于MAR算法的哈欠检测，利用Dlib提取嘴部的6个特征点，通过这6个特征点的坐标(51、59、53、57的纵坐标和49、55的横坐标)来计算打哈欠时嘴巴的张开程度。当一个人说话时，点51、59、53、57的纵坐标差值增大，从而使MAR值迅速增大，反之，当一个人闭上嘴巴时，MAR值迅速减小。嘴部主要取六个参考点，如下图：



嘴部检测相关代码：

from scipy.spatial import distance as dist

from imutils.video import FileVideoStream

from imutils.video import VideoStream

from imutils import face\_utils

import numpy as np # 数据处理的库 numpy

import argparse

import imutils

import time

import dlib

import cv2

import math

import time

from threading import Thread

def mouth\_aspect\_ratio(mouth): # 嘴部

A = np.linalg.norm(mouth[2] - mouth[10]) # 51, 59

B = np.linalg.norm(mouth[4] - mouth[8]) # 53, 57

C = np.linalg.norm(mouth[0] - mouth[6]) # 49, 55

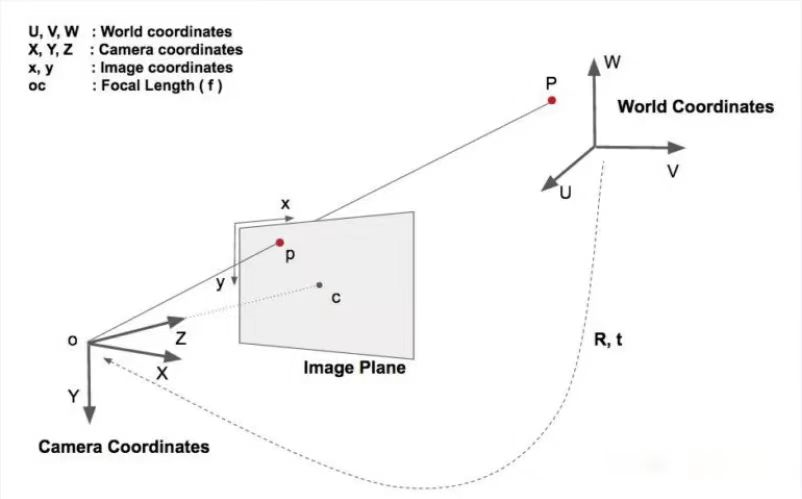
mar = (A + B) / (2.0 \* C)

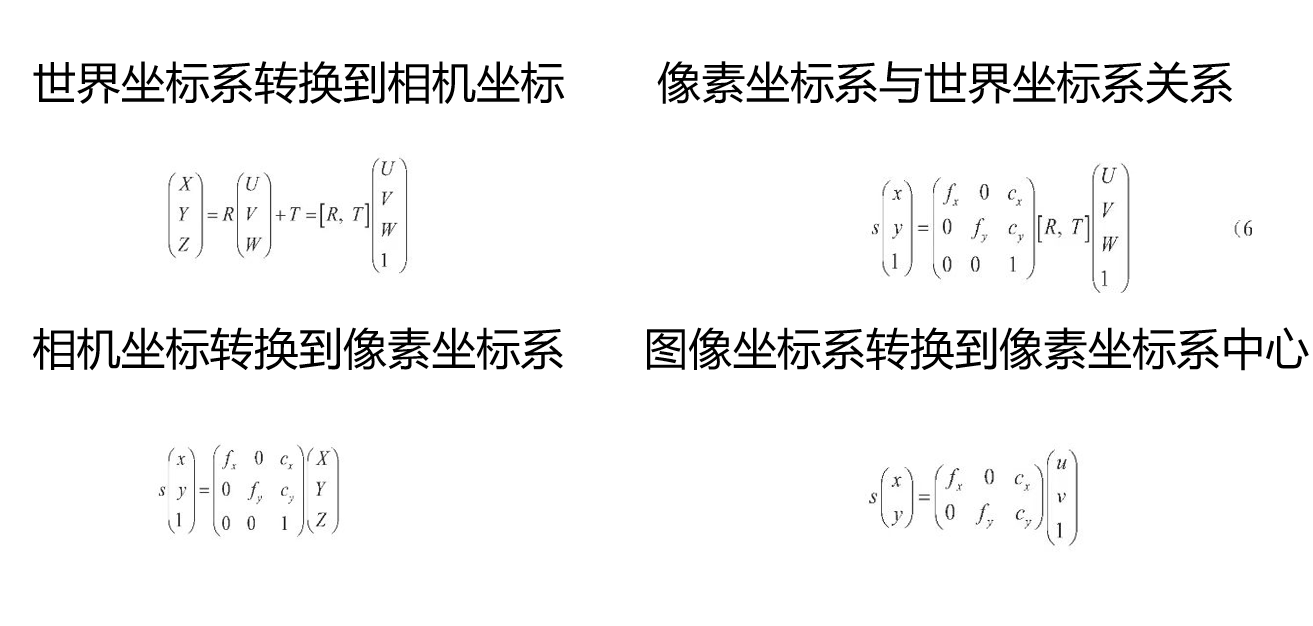
return mar

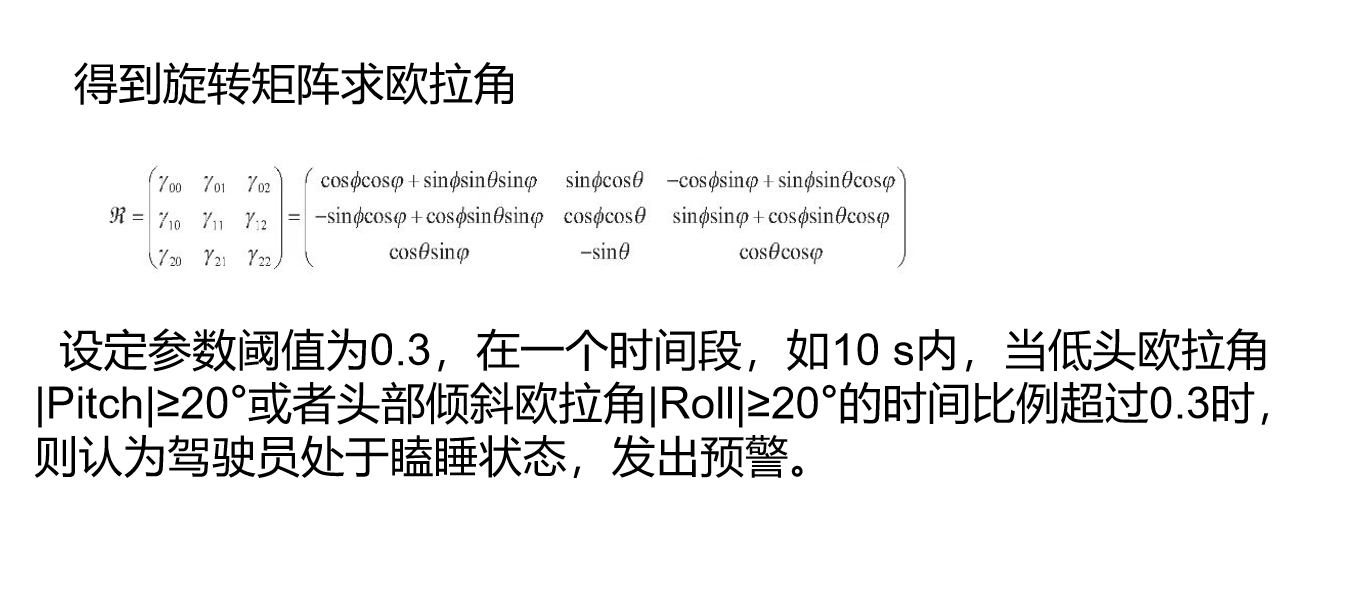
点头检测算法：基于HPE算法的点头检测：算法步骤：2D人脸关键点检测，3D人脸模型匹配，求解3D点和对应2D点的转换关系，根据旋转矩阵求解欧拉角。

检测过程中需要使用世界坐标系(UVW)、相机坐标系(XYZ)、图像中心坐标系(uv)和像素坐标系(xy)。

坐标系转化如图







* 1. **运行结果及优化方案**

实验效果详见PPT内视频演示。

优化方案：

我们对眨眼检测的算法了解比较深入 ，对于这种算法，我们可以改进的地方是该程序代码在被检测人戴眼镜时对于眨眼的判断不准确。需要去除反光和镜框等其他干扰因素才能更为准确的对眨眼进行判断与计数。我们对打哈欠检测和点头检测算法进行了一定程度的了解，对于这两种算法我们可以学习改进的地方是：

1.学习使用Dlib人脸识别库

2.在自己的算法中加入打哈欠检测和人脸检测

3.尝试使用摄像头输入图像，优化算法以满足实时性的要求。

* 1. **个人总结**

我们小组的综合实力较差，并且平时缺乏交流，所以作业完成度不高。在环境搭建上是通过其他同学的帮助才顺利完成的。这种经历，从一方面说明了我们需要更多的专业知识才能较好的完成任务，另外，也要合理的运用同学给予的帮助，合作往往是效率最高的学习方式。