Universitetet
i Stavanger

FACULTY OF SCIENCE AND TECHNOLOGY

BACHELOR'S THESIS

Study programme/specialisation: Bachelor of Science in Computer Science	Spring semester, 2021 Open
Author: Jari Kunnas	
Program Coordinator: Karl Skretting Supervisor(s): Øyvind Meinich-Bache	
Title: Object Detection, simulated conscious Norsk tittel: Objekt Deteksjon, simulert våkenhet	
Credits: 20	
Keywords: Object Detection, Tensorflow, Convolutional Neural Network SSDmobileNet, Raspberry Pi,	Page Numbers: 43 + Appendix: 39 pages Stavanger 15. May 2021

Contents

Summary

Medical emergencies and trauma situations are stressful events. Training and repetition in controlled environment is used for health professionals to gain experience and retain the learning for longer. Laerdal Medical creates medical equipment and training equipment meant for health personnel.

SimMan is a high-fidelity patient simulator created by Laerdal Medical to train teams in treating medical emergencies and trauma. To make the simulations more realistic to increase the training effect work has been done to make prototypes that can replicate realistic behaviour.

The project in this thesis builds on a head prototype with LCD monitors as eyes and with a joystick and switch controller attached for changing eye modes. One of those modes were used in this thesis for receiving target angles the eyes should be rotated to look at a detected object. The detected object was found using a neural network trained on detecting faces.

Edge devices like Raspberry Pi with lower computing capability are cheap and flexible for many use cases. The effectiveness object detection network can achieve on these edge devices makes this eye prototype system flexible for further implementation and more advanced functionality.

The modified eye prototype and object detection pipeline developed for this thesis performs well and appear realistic when there is a single target person in the depth camera's field of view. Multiple people visible will make the eyes change who it looks at in a way that does not seem realistic. There are also some angles relative to the prototype where the eye contact looks unfocused.

Chapter 1

Introduction

1.1 Motivation of Thesis

SimMan is a high-fidelity patient simulator used to train teams in treating medical emergencies and trauma. The training done on this product help save the lives of trauma victims, COVID-19 patients, and many others every day. The installed base of SimMan is over 10 000 simulators. The simulator already contains microphones, speakers, an on-board computer and network connection.

To improve the quality and realism of the training Laerdal Medical aim at making the simulators more realistic both in appearance and responses. A key element in assessing a patient's consciousness is the eye movement. An alert and conscious patient will naturally follow people and their movements in the environment. In addition, Laerdal Medical wants the simulator to respond realistically to clinical procedures involving eye movement, e.g. "follow my finger with your eyes".

1.2 Topic of Thesis

Goal of Thesis

The goal of this thesis was to use object detection together with an eye focus system to provide inputs that guide where the SimMan Patient Simulator eyes should focus.

A neural network that uses input from a depth camera to find the position of objects and send that to the prototype for visualisation was the target of this thesis. The objects to train the object detection network on was identified to be faces and fingers/pens for the clinical procedure "follow my finger test". This use of object detection networks for the eye prototype was targeted as a proof of concept prototype for potential further development for more advanced uses or implementation into the commercial SimMan patient simulator

Work Completed in this Thesis

A complete prototype system with depth camera, a face detection neural network and direction of the LCD monitor eyes to the faces was implemented. The face detection neural network used was a network trained by Github user: *"yeephycho"* on the *"WIDERFACE"* dataset. [20] [21]

The object detection network used and the surrounding pipeline in this thesis was only trained on faces and logic for changing objects to track with the eyes when multiple different objects are detected was not implemented.

A training pipeline using the Tensorflow object detection API to train a neural network on custom objects was set up and run on a very small custom dataset for testing the training setup. Collecting and labeling a large and diverse enough dataset for training on a relevant object was not completed.

The eye prototype using object detection on faces was tested with basic experiments to check the realism of the eye prototype system.

1.3 Thesis Report Overview

This thesis report contains these main parts:

- Background and Method
- Implementation
- Experiments, Results and Discussion

The *"Background and Method"* chapter contains brief explanation and references to the technologies used in the implementation of the project in this thesis.

The *"Implementation"* chapter has code examples of the novel and modified code created for this project. There are illustrations and explanations on how the different parts are set up to create a complete data sampling and interpretation pipeline from camera inputs to the eye visualisation outputs.

The *"Experiments, Results and Discussion"* part has three experiments performed to test and document the realism of the object tracking prototype.

Chapter 2

Background and Methods

This Chapter explains the technologies and theories of the methods used in this project

2.1 Method Overview

The method of the project in this thesis uses a combination of technologies. A face detection neural network is used on the inputs from the depth camera and sent for positioning of the eyes of the premade LCD monitor eye prototype from Laerdal Medical.

2.2 Artificial Neural Network for Object Detection

Artificial neural network design is influenced by the way neurons in brains communicate and function. [8] A large variations of designs and structures are used for different learning tasks and applications. The type of neural nets and their use in this thesis is explained briefly in the following sub chapters.

2.2.1 Convolutional Neural Networks

Many variations of convolution neural networks has been developed that build on the ideas from the network "Neocognotion" proposed by Dr. Kunihiko Fukushima in 1980. [7] Convolutional neural networks has a layer structure that is different than a classic neural network with fully connected layers. It filters in regions of an input and has a final fully connected layer that learns to recognize the complete objects and position of it. Earlier layers are sensitive to basic features and shapes and later layers detect combinations of features that make up part of the final object. [13] [16]

2.2.2 Transfer Learning

Transfer learning is where a pretrained model is used as the starting point in training a model to perform a new learning task. In this project a pretrained multiple object detection network was used to set up the training pipeline for custom images and classes. The pretrained model was trained on custom images of faces to only detect faces. Convolutional neural networks are suitable for transfer learning since the first layers recognize basic shapes and features. In transfer learning locking the first layers and only train the later ones that detect the complete objects achieves the transfer learning. [4]

2.2.3 SSDMobileNet

High performing neural networks for object detection are often large and require large computing resources to work fast enough for useful real time applications. For edge devices like Raspberry Pi smaller detection networks optimized for speed are needed. A Single Shot Detector was presented by Liu et al. on 2016. [14] The detector only processes the input once and returns multiple boxes of detections and their accuracy. Combining this detector with a MobileNet results in an efficient network that has good accuracy on edge devices. [9] Pretrained versions of SSDMobileNet trained using transfer learning on faces was used in this thesis.

2.2.4 Tensorflow Object Detection API

Tensorflow is a platform for machine learning and has an API set up for object detection and custom training. [1] A comprehensive tutorial is made by Lyudmil Vladimirov and was followed to test custom training in this project. Priorities led to implementation of a custom trained object detection network to not be completed. [19]

Using Tensorflow on a computers GPU(graphical proccessing unit) requires care in selecting compatible versions and following tested procedures like the one from Lyudmil Vladimirov is recommend. [19]

2.3 Laerdal Medical's Patient Simulator SimMan

Laerdal Medical's Patient Simulator SimMan is an advanced full body patient simulator. [15] To make the patient simulator feel more realistic a prototype for eyes that look realistic and can move around naturally was made prior to this thesis's project. The prototype is explained more in detail in the subsection below.

2.3.1 Eye Prototype

The eye prototype that was made by Laerdal Medical is a head mounted on a platform. The head has two LCD monitors as eyes connected to a Raspberry Pi that runs the eye simulation.[2][6] The eye simulator has curved lenses on top of the monitors to make the eyes look spherical. A 3D printed socket and mount for the spherical lenses makes if fit realistically inside the head platform. To avoid the challenges the spherical lenses creates on optics the prototype in this thesis project was developed an tested with only the flat LCD monitors without lenses on top. See figure 2.1 and the eye without the lens inserted. That was how both the eyes were used in this project.

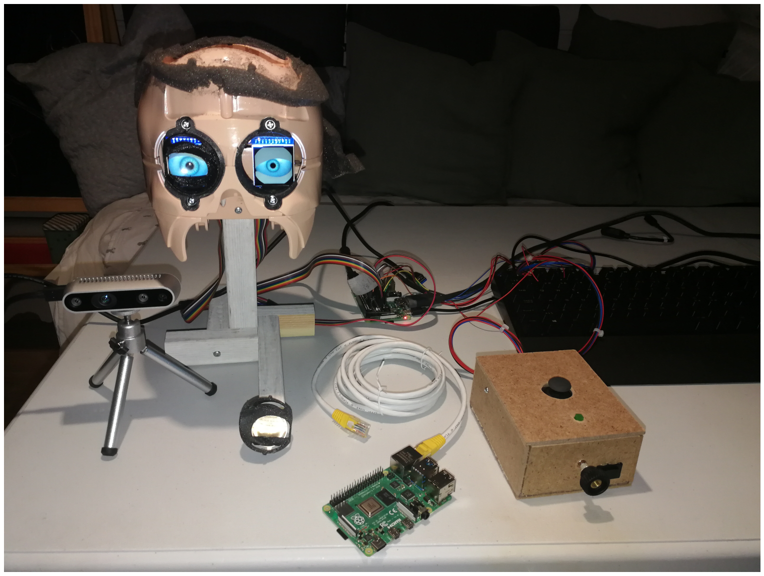

Figure 2.1: Overview of all equipment setup in this prototype. One eye lense is removed to demonstrate the LCD monitor behind it. Depth camera can be seen below, to the side of the head prototype. Raspberry Pi 3 running the eyes can be seen in the background.

Adafruit Animated Eyes Bonnet for Raspberry Pi

The LCD monitors the eyes uses are two 1.54" monitors with 240x240 resolution with full angle viewing. [2]. This is designed to connect to the GPIO pins and hardware of the Raspberry Pi 3 Model B. [2] [6]

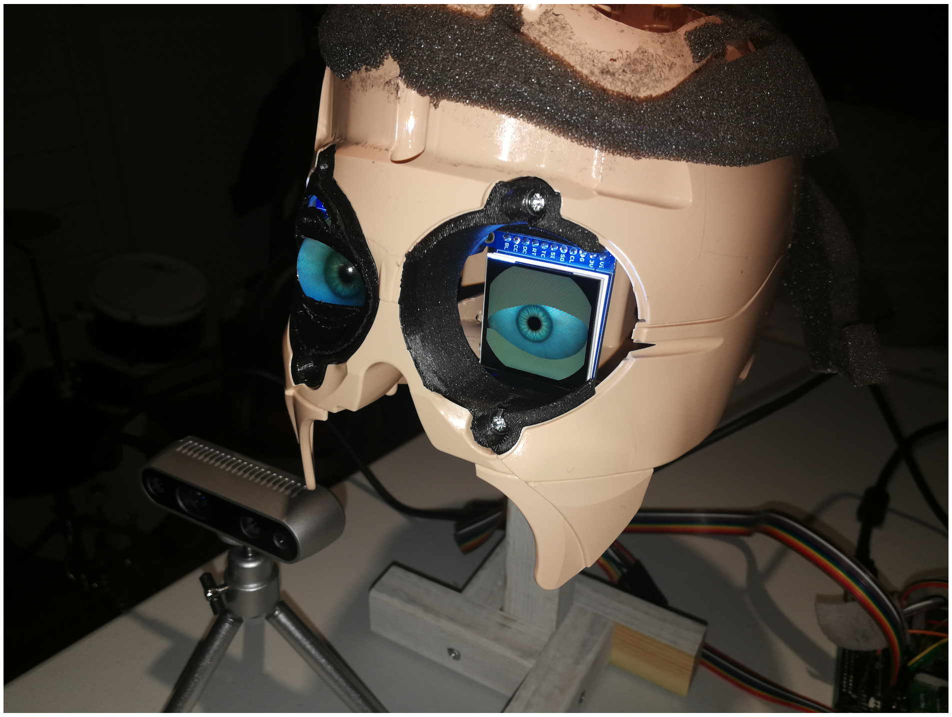

Figure 2.2: Closeup of eyes. One lens is removed for demonstration. Depth camera can be seen below and to the side of the head prototype

2.3.2 Intel Realsense D435 [10] Depth Camera

Depth camera was identified to help in the positioning of the eyes so the focus would not be cross-eyed. When eyes look at an object closer to itself the individual positioning of the eyes become crucial to appear realistic. This camera is a stereo camera that has depth sensing capability. It comes with a development kit and python library that is compatible with the other parts used in this project. Using the python *"pyrealsense2"* library there are two arrays of data that can be used from the camera.[11] One array from the normal RGB (Red, Green, Blue) camera and a depth array of same frame. [10]

Chapter 3

Implementation

This chapter explains the novel work and the modifications of existing solution that was done in this project. Code snippets are included and explained in this chapter. The full code can be found in the Appendix or the projects Github repository. [12]

3.1 Flowchart of Object Detection Eye Prototype

The flowchart in figure 3.1 below show the files used in the eye prototype system made in this project and where they are used and the device running them. Details on the individual scripts and the novel and modified code inside is explained below in this chapter. This flowchart shows an overview for context on where they are run.

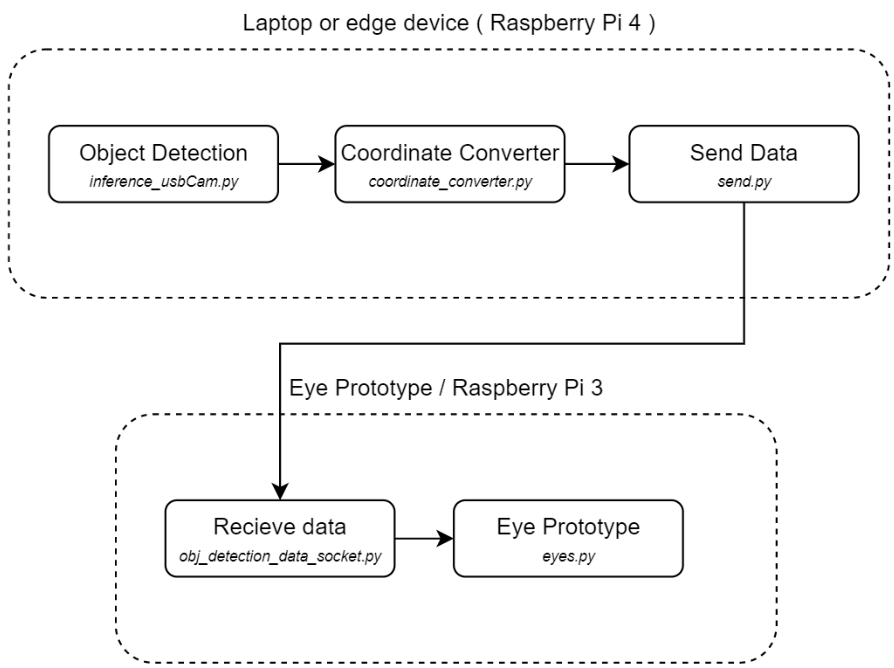

Figure 3.1: Flowchart of Files and Devices using them

3.2 Object Detection

Two object detection neural networks were tested in this project. One trained on a custom dataset captured and labeled manually and one pre-trained on faces.[21] Due to time constraints it was not prioritised to complete preparing a good dataset and running training on it. The pipeline for training on custom data and using the network in the prototype pipeline was implemented and explained below in this chapter.

3.2.1 Dataset for Custom Transfer Learning

The data i.e. images collected in this project was only of the author and was not of sufficient variability for good results. The dataset contained only 10 images where 8 was used for training and 2 for testing.

Prioritisation of tasks led to this dataset not being expanded and used for a self trained custom network to be used in the prototype pipeline.

To properly train a new custom model to detect faces or potentially other object of interest in patient simulation scenarios, larger datasets with more variations are needed so the models are not overfitted to very specific data.

3.2.2 Transfer Learning Parameters on Custom Dataset

The transfer learning was set up with a low learning rate of 0.08 and 50000 steps. These are inputs in the pipeline.config file that is set up using the Tensorflow object detection API. [19] The files and model trained can be found in this thesis's Github repository under *"person_ event-detect-recognition/custom_from_ scratch/tensorflow_face_ model_jk "*. [12]

3.2.3 Pretrained Face Detection Network

Using a pretrained face network was prioritised in this project so that a complete pipeline and prototype could be completed and tested. The network used was a "*SSDmobileNet*" trained on the "*WIDER FACE*" benchmark dataset. [21] [20] The pretrained model has a python script for detection and visualisation that was modified in this project. Details of this is in subsection 3.2.4 below.

3.2.4 Object Detection Inference Script

This subsection will show code snippets of modified code of the original inference script in the repository by Github user "yeephycho" (*"inference_ usbCam_face.py"*). [21].

This inference script is used to collect the input data from the camera, run object detection on them, convert the inputs using the *"ConvertCoordinates"* class and then send it over the network to the eye prototype using the *"SendData"* class.

Modified dependency imports are the pyrealsense2 package (line 15), classes made for this thesis; *"SendData"* and *"ConvertCoordinates"* and native package "*copy*" for holding the last valid depth measurement in memory. See line 1 to 20 below for the packages included in the python script *"inference_usbCam_face.py"*

```
1 #!/usr/bin/python
2 # -*- coding: utf-8 -*-
3 # pylint: disable=C0103
4 # pylint: disable=E1101
5
6 from os import X_OK
7 import sys
8 import time
9 import numpy as np
10 import tensorflow as tf
11 import cv2
12 import collections
13 import six
14 import PIL.Image as Image
15 import pyrealsense2 as rs
16 from utils import label_map_util
17 from utils import visualization_utils_color as vis_util
18 from send import SendData
19 from coordinate_converter import ConvertCoordinates
20 import copy
```

The Tensorflow Object Detection API [19] has a function (*visualize_boxes_and_labels_on_image_array()*) in the "*utils*" module that takes the detection outputs and create bounding boxes and prints the class label and detection accuracy. A modified version of this function that takes in the detection outputs and returns the coordinates for the corners of the bounding box is shown below (lines 38 to 108). This function is used to determine the pixel location on the detection camera the prototype eyes will be directed at. Please see appendix or Github repository for the complete function. [12]

```
38 def get_eye_focus_coordinate(
39     image,
40     boxes,
```

```
    classes,
    scores,
    category_index,
    instance_masks=None,
    instance_boundaries=None,
    keypoints=None,
    keypoint_scores=None,
    keypoint_edges=None,
    track_ids=None,
    use_normalized_coordinates=False,
    max_boxes_to_draw=20,
    min_score_thresh=.5,
    agnostic_mode=False,
    line_thickness=4,
    mask_alpha=.4,
    groundtruth_box_visualization_color='black',
    skip_boxes=False,
    skip_scores=False,
    skip_labels=False,
    skip_track_ids=False):
    """
```

Lines 177 to 213 of *"inference_usbCam_face.py"* sets up the camera for capturing images and depths(line 178-196), sets up the network class for sending data(line 202-204) and sets up the converter class for calculating the correct angles for the eyes(line 206-212). Details on the converter class can be seen in subsection 3.2.5.

```
178     # Configure depth and color streams
179     pipeline = rs.pipeline()
180     config = rs.config()
181
182     # Get device product line for setting a supporting ...
        resolution
183     pipeline_wrapper = rs.pipeline_wrapper(pipeline)
184     pipeline_profile = config.resolve(pipeline_wrapper)
185     device = pipeline_profile.get_device()
186     device_product_line = ...
        str(device.get_info(rs.camera_info.product_line))
187
188     config.enable_stream(rs.stream.depth, 640, 480, ...
        rs.format.z16, 30)
189
190     if device_product_line == 'L500':
191         config.enable_stream(rs.stream.color, 960, 540, ...
        rs.format.bgr8, 30)
```

```
192     else:
193         config.enable_stream(rs.stream.color, 640, 480, ...
        rs.format.bgr8, 30)
194
195     # Start streaming
196     pipeline.start(config)
197     tDetector = TensoflowFaceDector(PATH_TO_CKPT)
198
199     cap = cv2.VideoCapture(camID)
200     windowNotSet = True
201
202     #socket sending
203     send_data_to_socket = SendData()
204     send_data_to_socket.setup_server_sending()
205
206     #Converterclass
207     coordinate_converter = ConvertCoordinates()
208     coordinate_converter.set_camera_resolution((640,480)) ...
        #camera resolution
209     ...
        coordinate_converter.set_eye_center_offset_from_screen(-10) ...
        # distance to fictive eye center behind monitor
210     coordinate_converter.set_mode('3D')
211     # coordinate_converter.set_xyz(50,50,1000) #default ...
        point to look at top left looking at head
212     depth_previous = 0.8
```

Below is the start of the while loop that does the detections on the input data from the camera. The camera data is converted to a "*numpy*" array to be compatible with the Tensorflow detections.

```
214     while True:
215         # Wait for a coherent pair of frames: depth and color
216         frames = pipeline.wait_for_frames()
217         depth_frame = frames.get_depth_frame()
218         color_frame = frames.get_color_frame()
219         if not depth_frame or not color_frame:
220             continue
221
222         # Convert images to numpy arrays
223         depth_image = np.asanyarray(depth_frame.get_data())
224         color_image = np.asanyarray(color_frame.get_data())
```

A list of normalised coordinates 0 to 1 is returned from the function *get_eye_focus_coordinate()* (line 263-273). For visualisation on the live video stream with the *"OpenCV"* python package, pixel position as integers was needed. Line 278 to 280 converts the float list to an integer list. A red circle with radius 10pixels was chosen to demonstrate the focus point for the eye prototype. A point 1/3 from the left of the bounding box and 1/3 from the top of the bounding box was chosen as the point where the right eye of faces normally is located and selected as the focus point for the eye prototype.

```
263         box_test = get_eye_focus_coordinate(
264             image,
265             np.squeeze(boxes),
266             np.squeeze(classes).astype(np.int32),
267             np.squeeze(scores),
268             category_index,
269             use_normalized_coordinates=True,
270             max_boxes_to_draw=200,
271             min_score_thresh=.3,
272             agnostic_mode=False)
273         # print(box_test) #example (0.23469042778015137, ...
        0.30845338106155396, 0.7406021952629089, ...
        0.5217226147651672)
274
275
276         if box_test:
277             # print(box_test)
278             box_int_list = [0,0,0,0]
279             for i in range(4):
280                 box_int_list[i] = int(box_test[i])
281
282             # 1/3 from the left of the box
283             x_location = ...
        int(((box_int_list[1]-box_int_list[0])*1/3)+box_int_list[0])
284             #1/3 from the top.
285             y_location = ...
        int(((box_int_list[3]-box_int_list[2])*1/3)+box_int_list[2])
```

The depth sensing capability of the camera is used to derive the location of the detected faces in 3 dimensional space. Line 286 to 295 takes the depth frame from the camera and finds the distance at the X and Y pixel location of the focus point from previous steps in the code. The depth sensor will occasionally return a frame with 0's. Storing the previous distance above 0.01 meter is used so the eyes will not "flicker" between a real focus distance and 0 meter from the camera. If a 0 frame is returned from the depth camera the previous depth measurement will be used. This is handled by line 292-295.

```
286             # get depth from realsense camera
287             depth_location = ...
        depth_frame.get_distance(x_location, y_location) # ...
        depth in xx units
288             depth_location_left = ...
        depth_frame.get_distance(x_location, y_location+10)
289             depth_location_right = ...
        depth_frame.get_distance(x_location, y_location-10)
290             depth_location = ...
        np.mean([depth_location,depth_location_right,depth_location_left])
291             # Write some Text
292             if depth_location < 0.01:
293                 depth_location = depth_previous
294
295             depth_previous = copy.deepcopy(depth_location)
```

Conversion of the the pixel postion x and y and the depth to the position is sent to the *"coordinate_ converter"* class instance in line 316. The converted angle for the eye prototype is then retrieved from the converter class and sent using the *"send_ data_ to_ socket"* class. The sending over network is exception handled with try: except:, so the code does not stop if there is a network problem. There is also a very small sleep delay (line 326 and 330) put in after sending that can be altered to simulate slower detection speed and limit the network usage on detection speeds faster than needed for the eye prototype.

```
316             coordinate_converter.set_xyz(
317                 circle_coordinates[0],
318                 circle_coordinates[1],
319                 depth_location*1000
320             )
321
322             try:
323                 str_data_to_send = ...
        coordinate_converter.get_eye_coordinates()
324                 # print(str_data_to_send)
325                 ...
        send_data_to_socket.send_data(str_data_to_send)
326                 time.sleep(0.05)
327             except Exception:
```

```
                # str_data_to_send = ...
    coordinate_converter.get_eye_coordinates()
                # ...
    send_data_to_socket.send_data(str_data_to_send)
                time.sleep(0.05)
```

3.2.5 Object Detection Coordinate Conversions

Full code can be viewed in this projects repository [12] and appendix B. Main parts and calculations will be described in this sub chapter.

Eye Prototype Angle Calculations

The eye prototype is explained in more detail in section 3.4. Shortly explained it is visualizing 3D object of eyes that it rotates a camera around to angles given to the prototype's code. These angles are what is calculated from the object detection pixel position and depth.

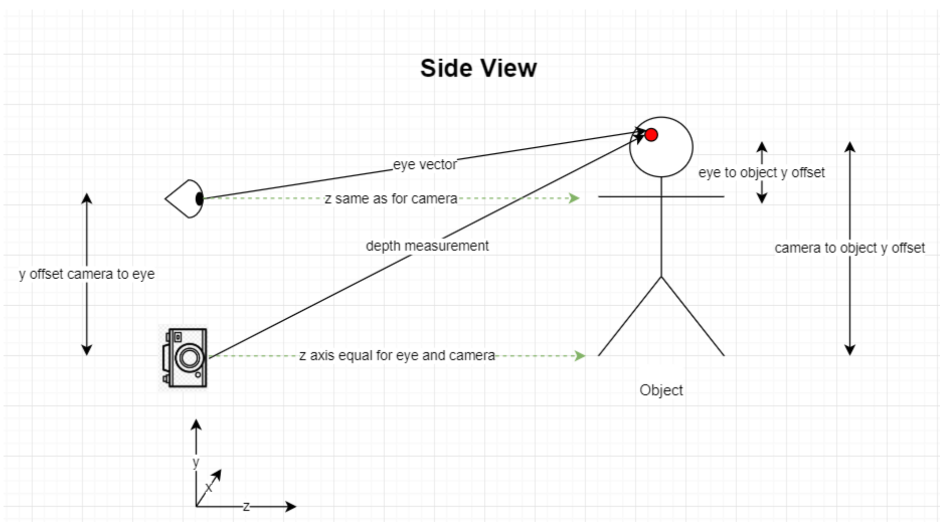

Figure 3.2: Sideview of the measurements and calculated distances used in calculation of eye Y angle

Since the camera has a specific field of view, the distance and pixel location to the detection there is enough information to calculate a three dimensional position vector (X, Y, Z). First the amount of pixels per degree of field of view is calculated. From that variable the degrees from center or edge of image can be calculated. Since the distance is measured it will be the hypotenuse in this trigonometry. The distance from from the coordinate system centers can then be measured in the side view plane (seen in Figure 3.2) and top down view plane. Lines 74-88 in *"coordinate_ converter.py"* calculates the x, y and z displacement of the object from the camera reference.

```
74          pix_per_degree_x = ...
        self.__camera_resolution[0]/self.__fov_x
75          degrees_from_left = x / pix_per_degree_x
76          degrees_from_center = degrees_from_left - ...
        (self.__fov_x/2)
77          x_distance_from_center_mm = \
78              math.sin(math.radians(degrees_from_center)) * ...
        depth
79          z_distance_from_center_mm = \
80              math.cos(math.radians(degrees_from_center))* depth
81          z_distance_from_center_mm = \
82              z_distance_from_center_mm - ...
        self.__camera_to_between_eyes_offset_z
83
84          pix_per_degree_y = ...
        self.__camera_resolution[1]/self.__fov_y
85          degrees_from_top = y / pix_per_degree_y
86          degrees_from_center = degrees_from_top - ...
        (self.__fov_y/2)
87          y_distance_from_center_mm = \
88                              ...
        math.sin(math.radians(degrees_from_center)) * depth
```

When the x, y and z position relative to the camera is found the eye offsets from the camera can be taken into account and results in two sides of the triangle available and the angle can be found in the two planes mentioned above. The z distance in the coordinate system is the same for camera and eyes. If it is not the same a z offset variable can be set in the code. The *"eye to object y offset"* distance in figure 3.2 is the other length needed to find the angle to object relative to the z axis. The y direction angle is the same for both eyes when camera and prototype head is put in the same horizontal orientation. The x (sideways) angles will not be the same and need to be calculated individually. This is done in line 99 to 103.

Individual y angles for left and right eye is calculated in this code. This is done for future applications where camera and head might be positioned in different coordinate systems relative to the camera and the heads straight ahead z axis.

```
91          #x and y coordinates relative to eye positions
92          left_eye_x = x_distance_from_center_mm - ...
        self.__eye_offset_L_x
93          left_eye_y = y_distance_from_center_mm - ...
        self.__eye_offset_L_y
94
95          right_eye_x = x_distance_from_center_mm - ...
        self.__eye_offset_R_x
96          right_eye_y = y_distance_from_center_mm - ...
        self.__eye_offset_R_y
97
98
99          left_eye_x_angle = ...
        math.asin(left_eye_x/z_distance_from_center_mm)
100         left_eye_y_angle = ...
        math.asin(left_eye_y/z_distance_from_center_mm)
101
102         right_eye_x_angle = ...
        math.asin(right_eye_x/z_distance_from_center_mm)
103         right_eye_y_angle = ...
        math.asin(right_eye_y/z_distance_from_center_mm)
```

3.3 Transfer of Detection

The eye prototype uses hardware which is designed for Raspberry Pi 3. This model of Raspberry Pi is not as powerful as the newer version of Raspberry Pi 4 or other computers. A laptop or a Raspberry Pi 4 was used as the computing system to run the object detection. Due to the object detection being run on an external system and the eye prototype not easily ported to another system, code for transferring and receiving data was developed. This section describes the classes for sending and receiving data over cables or wireless network. [6] [12]

One class for sending data and one class for receiving data was developed.

3.3.1 Send Data

The class *"SendData()"* in *"send.py"* sets up a server for sending data. The way of using this class is to initialize it with the built in method *"setup_server_sending"*. The class has some hardcoded defaults for ip and port that was used, but they can be set with the setters; *"set_host_ip('ip address')"* and *"set_port(port number)"*.

```
import socket
import numpy as np
import time


class SendData():
```

```
    def __init__(self) -> None:
        self.__host = '192.168.191.125' # loopback ...
    interface address (localhost)
        self.__port = 65432 # Port to listen on ...
    (non-privileged ports are > 1023)
        self.__socket = socket.socket(socket.AF_INET, ...
    socket.SOCK_STREAM)
        self.__connection = None
        self.__address = None

    def setup_server_sending(self):
        print("Server Started waiting for client to ...
    connect ")
        self.__socket.bind((self.__host, self.__port))
        self.__socket.listen(5)
        self.__connection, self.__address = ...
    self.__socket.accept()
        print('Connected to', self.__address)

    def send_data(self,my_data):
        # my_data = f'{self.__eyeX},{self.__eyeY}'
        # print(my_data)
        my_data_bytes = bytes(my_data, 'utf-8')
        # print('length of bytes: ', len(my_data_bytes))
        self.__connection.send(my_data_bytes)
```

```

    def set_host_ip(self, ip):
        #set host ip as string '192.168.1.1'
        self.__host = ip

    def set_port(self, port):
        #set port as int
        self.__port
```

3.3.2 Receive Data

The class *"RecieveData()"* in *"obj_ detection_ data_ socket.py"* connects to a socket server for receiving data. This class is used on the hardware for the eye prototype for receiving data. Details on the use of the external data can be viewed in section 3.4.

This class uses the python standard library *"socket"*. [17] This module provides access to the BSD socket interface.

```
import socket

class RecieveData():
    """
    Class that starts a socket connection and recieves eye ...
    coordinates
    for eye simulator to use

    """
    def __init__(self):
        self.__host = '192.168.191.125'
        self.__port = 65432
        self.__eyeXR = 30
        self.__eyeYR = 30
        self.__eyeXL = 30
        self.__eyeYL = 30
        self.__socket = socket.socket(socket.AF_INET, ...
    socket.SOCK_STREAM)
        self.__connected_to_socket = False
```

"RecieveData()" is used by initializing it with the *"connect_ to_ server()"* method. The connection is set in an try except clause in case the server

is not set up. The code and eye simulation would terminate and the code would need restart if this was not exception handled.

```
    def connect_to_server(self):
        try:
            self.__socket = socket.socket(socket.AF_INET, ...
    socket.SOCK_STREAM)
            self.__socket.connect((self.__host, self.__port))
            self.__connected_to_socket = True
        except:
            self.__connected_to_socket = False

    def get_data_from_connection(self):
            data = self.__socket.recv(1024).decode('utf-8')
```

The connection can be closed using the *"close_ connection()"* method

```
    def close_socket(self):
        self.__socket.shutdown()
        self.__socket.close()
```

Default class IP address and port number can be overwritten with the setter methods *"set_ host_ ip('enter ip address as string')"* and *"set_ host_ port('set host port as integer')"*.

```
    def set_host_ip(self, host_ip):
        # Set host ip as string: example: '192.168.2.1'
        self.__host = host_ip

    def set_host_port(self, host_port):
        # Set host port as integer: example: 65432
        self.__port = host_port
```

Static IP set on the host and client on the cabled network interfaces creates little need of editing these settings.

3.4 Eye Simulator

As explained in section 2.3.1 Laerdal Medical has a prototype made of eyes using LCD monitors. This section explains in detail the modifications and some general functions of the prototype that is built on the Adafruit LCD monitors and code.[2]

3.4.1 Edge Device

The edge device the eye prototype uses is a Raspberry Pi 3B. [6] This lacks the processing power to run the object detection. It has a memory card with its operating system on. This card can be inserted in a computer and the *"Pi_eyes"* code can be updated there. [3] It is also possible to set up the Raspberry Pi to be accessed via SSH and edits can be done to the code and eye prototype directly while the Raspberry Pi is running.

3.4.2 Modifications on Premade Eye Prototype Code

The code for the eye simulation was originally developed by the company that makes the LCD monitors and then modified by Laerdal Medical's application with a joystick and selection switch and button. [2] [3] The github repository for the eyes [3] includes a couple of modules and eye texture maps that can be modified for preferred look. For this project only the *"eyes.py"* code was modified. The *"obj_detection_data_socket.py"* containing the *"RecieveData()"* class was added to the prototype for receiving eye angles and used in the *"eyes.py"* script .

Modified dependency imports can be seen in the code snippet below:

```
# for object detection use
from obj_detection_data_socket import RecieveData
import threading
import queue
```

Lines 337 to 345 initialize the recieving data class *"RecieveData()"*, the shared queue (*"dnn_ queue"*) between threads that contain the eye angles from the object detection, inital eye angles for second monitor (prototypes left eye) and the previous eye angles written to the monitors. The previous angles are used to keep the eyes at the same position and allowing for the eye animation winking to continue until a new angle is received from the object detection.

```
# initialize socket class, used if option 6 is selected.
eye_coordinate_socket = RecieveData()
dnn_queue = queue.Queue()
curX2 = 20
curY2 = 20
last_x = 0
last_y = 0
last_x2 = 0
last_y2 = 0
```

The function that does the eye position updates *"frame(p)"* uses global variables defined earlier in the script. New global variables where added; *"curX2, curY2"* in line 349 and lines 372 to 377 in snippet below. *"curX, curY, curX2 and curY2"* are the eye angles for right and left eye respectively.

```
347 # Generate one frame of imagery
348 def frame(p):
349     global startX, startY, destX, destY, curX, curY, ...
        curX2, curY2
350     global startXR, startYR, destXR, destYR, curXR, curYR
351     global moveDuration, holdDuration, startTime, isMoving
352     global moveDurationR, holdDurationR, startTimeR, isMovingR
353     global frames
354     global leftIris, rightIris
355     global pupilMinPts, pupilMaxPts, irisPts, irisZ
356     global leftEye, rightEye
357     global leftUpperEyelid, leftLowerEyelid, ...
        rightUpperEyelid, rightLowerEyelid
358     global upperLidOpenPts, upperLidClosedPts, ...
        lowerLidOpenPts, lowerLidClosedPts
359     global upperLidEdgePts, lowerLidEdgePts
360     global prevLeftUpperLidPts, prevLeftLowerLidPts, ...
        prevRightUpperLidPts, prevRightLowerLidPts
361     global leftUpperEyelid, leftLowerEyelid, ...
        rightUpperEyelid, rightLowerEyelid
```

```
    global prevLeftUpperLidWeight, prevLeftLowerLidWeight, ...
    prevRightUpperLidWeight, prevRightLowerLidWeight
    global prevPupilScale
    global irisRegenThreshold, upperLidRegenThreshold, ...
    lowerLidRegenThreshold
    global luRegen, llRegen, ruRegen, rlRegen
    global timeOfLastBlink, timeToNextBlink
    global blinkStateLeft, blinkStateRight
    global blinkDurationLeft, blinkDurationRight
    global blinkStartTimeLeft, blinkStartTimeRight
    global trackingPos
    global trackingPosR
    global eye_coordinate_socket
    global dnn_queue
    global last_x
    global last_y
    global last_x2
    global last_y2
```

In line 603 there is an if statement that will activate if the switch is set into position 6. This is the mode that uses the object detection angles. It uses the same rotation functions in the prototype if the the switch position is set to other positions than 6. When position 6 is set it writes the individual independent positions for the calculated eye angles in line 622 to 638

```
603     if GPIO != 6:
604         convergence = 2.0
605
606         rightIris.rotateToX(curY)
607         rightIris.rotateToY(curX - convergence)
608         rightIris.draw()
609         rightEye.rotateToX(curY)
610         rightEye.rotateToY(curX - convergence)
611         rightEye.draw()
612
613         # Left eye (on screen right)
614
615         leftIris.rotateToX(curY)
616         leftIris.rotateToY(curX + convergence)
617         leftIris.draw()
618         leftEye.rotateToX(curY)
619         leftEye.rotateToY(curX + convergence)
620         leftEye.draw()
621     else:
622         convergence = 0
623
```

```
        rightIris.rotateToX(curY)
        rightIris.rotateToY(curX - convergence)
        rightIris.draw()
        rightEye.rotateToX(curY)
        rightEye.rotateToY(curX - convergence)
        rightEye.draw()

        # Left eye (on screen right)

        leftIris.rotateToX(curY2)
        leftIris.rotateToY(curX2 + convergence)
        leftIris.draw()
        leftEye.rotateToX(curY2)
        leftEye.rotateToY(curX2 + convergence)
        leftEye.draw()
```

A new function was made for the intent of receiving the data and being applicable for use in another thread. The threading was implemented to let the animation of the eyes continue winking instead of appearing frozen waiting for inputs from the object detection.

A global queue (*"dnn_queue"*) is used for holding angles for the eyes. The function *"frame(p)"* uses the same queue for popping out the first(oldest) angles and updating the eye angles. The eye animation is fast enough to pop the angles quickly and no pile ups of data in the queue was experienced in this project.

The function *"fill_queue()"* has a continuous loop running that checks if the switch is set to position 6 (object detection mode). If it is set to that position it will try to setup connection over the network. If it is not successful it will try continuously until it succeeds.

```
def fill_queue():
    global dnn_queue
    global eye_coordinate_socket
    global curX, curY, curX2, curY2

    while True:
        if checkGPIO() == 6:
            #modified for test of eye tracking
            # AUTOBLINK = False #disables blinking
            try:
                if not ...
```

```
    eye_coordinate_socket.get_socket_connected_status():
                    eye_coordinate_socket.connect_to_server()
            except Exception:
                ...
    eye_coordinate_socket.set_socket_connected_status(False)

            try:
                ext_curX, ext_curY, ext_curX2, ext_curY2 = ...
    eye_coordinate_socket.get_eye_coordinates_float()
                dnn_queue.put((ext_curX, ext_curY, ...
    ext_curX2, ext_curY2))

            except Exception as e:
                ...
    eye_coordinate_socket.set_socket_connected_status(False)    ...

                print(f'failed to get datafrom socket and ...
    put to queue: {e}')

        if checkGPIO() != 6 and ...
    eye_coordinate_socket.get_socket_connected_status():
            ...
    eye_coordinate_socket.set_socket_connected_status(False)
            try:
                eye_coordinate_socket.close_socket()
            except Exception:
                pass
            time.sleep(2)
```

Line 709 to 712 sets up the *"fill_queue()"* function for multi-threading. It will run in the background and populate the *"dnn_queue"* queue when it receives new data from over the network from the object detection algorithm.

```
709 #MAKE THREAD FOR EXTERNAL DATA AND START IT.
710 get_data_thread = threading.Thread(target=fill_queue)
711 get_data_thread.deamon = True
712 get_data_thread.start()
```

The main loop of the eye prototype can be seen below on lines 717 to 734. The thread that receives data will run threaded with this loop. The updated drawing of the the eyes on the monitor happens on line 730. When that function is called it checks for the switch position. If it is set to object detection mode 6 it will pop the *"dnn_queue"* for updates to the eye angles to use. If there is no data in the queue it keeps updating with the latest received.

The other parts of this main loop is related to the possibility of having a light sensor that corrects the pupil size. In this implementation without that sensor it will only vary it randomly.

```
# MAIN LOOP -- runs continuously ...
    -------------------------------------------
while True:

    if PUPIL_IN ≥ 0: # Pupil scale from sensor
        v = bonnet.channel[PUPIL_IN].value
        # If you need to calibrate PUPIL_MIN and MAX,
        # add a 'print v' here for testing.
        if   v < PUPIL_MIN: v = PUPIL_MIN
        elif v > PUPIL_MAX: v = PUPIL_MAX
        # Scale to 0.0 to 1.0:
        v = (v - PUPIL_MIN) / (PUPIL_MAX - PUPIL_MIN)
        if PUPIL_SMOOTH > 0:
            v = ((currentPupilScale * (PUPIL_SMOOTH - 1) + ...
    v) /
                PUPIL_SMOOTH)
        frame(v)
    else: # Fractal auto pupil scale
        v = random.random()
        split(currentPupilScale, v, 4.0, 1.0)
    currentPupilScale = v
```

3.5 Object Detection Eye Prototype Files in Project

The files in the project can be seen in the figure 3.3 below. There are additional files in the repository, but they are related to custom training of an object detection network. [12]

```
|   .gitignore
|   README.md
|
+---rasppi3
|       eyes.py
|       obj_detection_data_socket.py
|
\---rasppi4
    |   coordinate_converter.py
    |   inference_usbCam_face.py
    |   send.py
    |
    +---model
    |       frozen_inference_graph_face.pb
    |
    +---protos
    |   |   face_label_map.pbtxt
    |   |   string_int_label_map_pb2.py
    |
    |
    +---utils
    |   |   label_map_util.py
    |   |   visualization_utils_color.py
```

Figure 3.3: File Tree of The Project

3.6 Code Tests

The individual classes and modified scripts were tested with test functions inside the .py files themselves. Unit tests were not set up for this projects as the modifications implemented small parts of the overall existing code and the individual classes and communication was simple to verify. Good practice would be to implement unit tests should this eye prototype be implemented in a larger system in the SimMan Patient Simulator.[15]

Chapter 4

Experiments, Results and Discussion

4.1 Experiments

This sections explains the experiments done to verify functionality of the object detection eye simulator prototype created in this project.

Three experiments were designed to verify the functionality of the object detection eye prototype in this project.

1. Single Person Tracking
2. Single Person Tracking with Multiple People Visible
3. Single Person Tracking Multiple Camera Position

The target person moved to 9 predefined positions (Figure 4.1), a screen capture of the object detection and a photo towards the prototype from the target person was done to confirm if the target was detected and if the eyes was properly angled at the target.

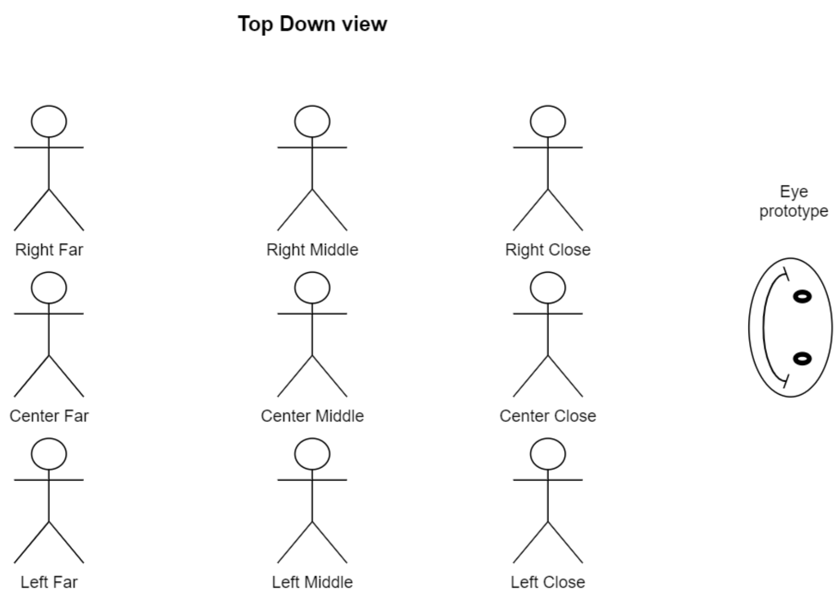

Figure 4.1: Positions For Target Person During All Experiments

Single Person Tracking

In the **Single Person Tracking** experiment the camera was set close under the prototype and a single person was moving around its field of view.

The person moving around held a camera and took pictures from the different positions it moved to for documentation on how the eyes orient. See Figure 4.1

A laptop ran the object detection with screen recording to document the face tracking.

A table for each experiment was filled out for the different positions a target was in and to record if there was a position that the object detection or eye tracking was less accurate at.

The person that was set as the target objective moved to 9 positions relative to the eye prototypes perspective; Left, Middle and Right at distances Close, Middle and Far. The metrics for these experiments was if the object detection detects the face of the target person and if the target person perceive that the eye prototype has eye contact.

Single Person Tracking with Multiple Persons Visible

The **Single Person Tracking with Multiple Persons Visible** experiment was conducted in the same way as the **Single Person Tracking** experiment. The difference was that there was multiple people visible in the field of view for the camera.

Single Person Tracking Multiple Camera Position

In the **Single Person Tracking Multiple Camera Position** experiment the camera was moved to different positions relative to the prototype head and relative position was updated in the *"coordinate_converter.py"*. The single target person moved to the same relative positions to the camera as in the other two experiments

4.2 Results and Discussion

All positions in the "Position" Column is from the Eye Prototype's perspective looking towards the target person.

The grading for the object detection and eye tracking was set to OK or Not OK. Not OK did not mean that it was very wrong, but there was not an impression of good eye contact. For object detection it was set to OK if the object detection detected and selected the right target face to focus on at the positions for the test.

4.2.1 Single Person Tracking

Table 4.1: Results - Single Person Tracking

Position	Object Detection	Eye Tracking
Left Close	OK	OK
Left Middle	OK	OK
Left Far	OK	OK
Center Close	OK	OK
Center Middle	OK	OK
Center Far	OK	OK
Right Close	OK	Not OK
Right Middle	OK	OK
Right Far	OK	OK

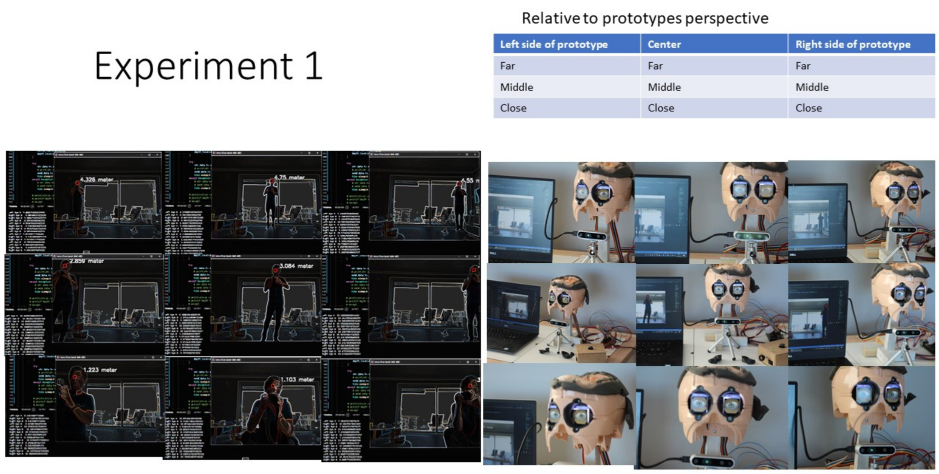

Figure 4.2: Single Person Tracking

Discussion

Tracking a single person in the field of view and directing the eyes towards the person was successful. The object detection had no problem in any of the positions tested. The eye tracking was following the target person well, but looked to the side of the target when the target was close to the right side of the prototypes perspective. See details on the results in Figure 4.2 above. Reasons for the eye tracking not being perfect can be from misalignment and measurement error of the camera position and rotation relative to the eyes.

4.2.2 Single Person Tracking with Multiple Persons Visible

Table 4.2: Results - Single Person Tracking with Multiple Persons Visible

Position	Object Detection	Eye Tracking
Left Close	Not OK	Not OK
Left Middle	Not OK	Not OK
Left Far	Not OK	Not OK
Center Close	OK	OK
Center Middle	Not OK	OK
Center Far	Not OK	OK
Right Close	OK	OK
Right Middle	OK	OK
Right Far	OK	OK

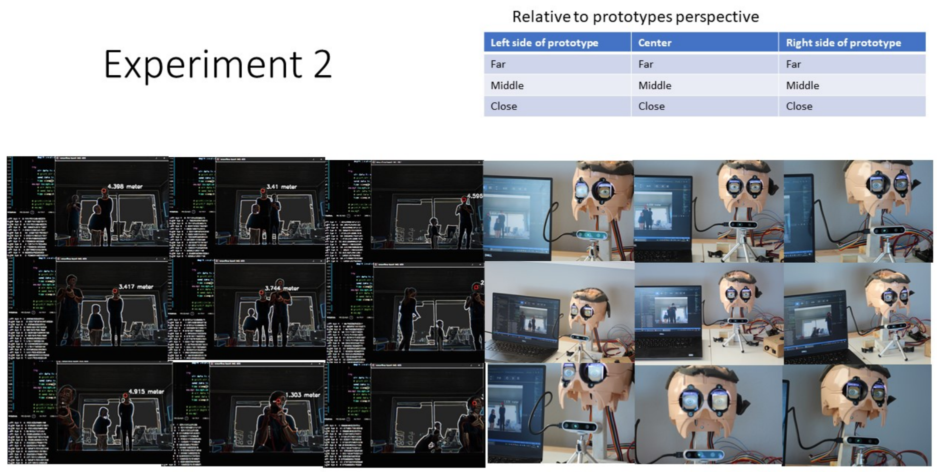

Figure 4.3: Single Person Tracking Multiple Persons Visible

Discussion

Tracking a single target person when there was multiple people in the cameras field of view was not successful. This was as expected as there was not implemented any logic in the code to handle this case. As in the experiment in section 4.1 the object detection works on faces and when the right target face was detected it was able to direct the eyes properly towards the target.

Photos and screen captures from the experiment can be seen in the Figure 4.3 above.

4.2.3 Single Person Tracking Multiple Camera Position

Camera Position 1

First change in camera position was in the same x (lateral sideways) location, but moved further back in the z(lateral backwards) orientation and moved higher in the y (vertical) orientation.

The camera was located directly behind the prototype so no X shift in position. It was 73 cm behind(Z) and 32 cm above(Y) the center of the eyes. See figure 4.4 for an illustration of the position marked by the red arrow.

Table 4.3: Results - Single Person* Tracking Multiple Camera Position 1

Position	Object Detection	Eye Tracking
Left Close	OK	OK
Left Middle	OK	Not OK
Left Far	OK	OK
Center Close	OK	Not OK
Center Middle	OK	Not OK
Center Far	Not OK*	Not OK
Right Close	OK	Not OK
Right Middle	OK	OK
Right Far	OK	OK

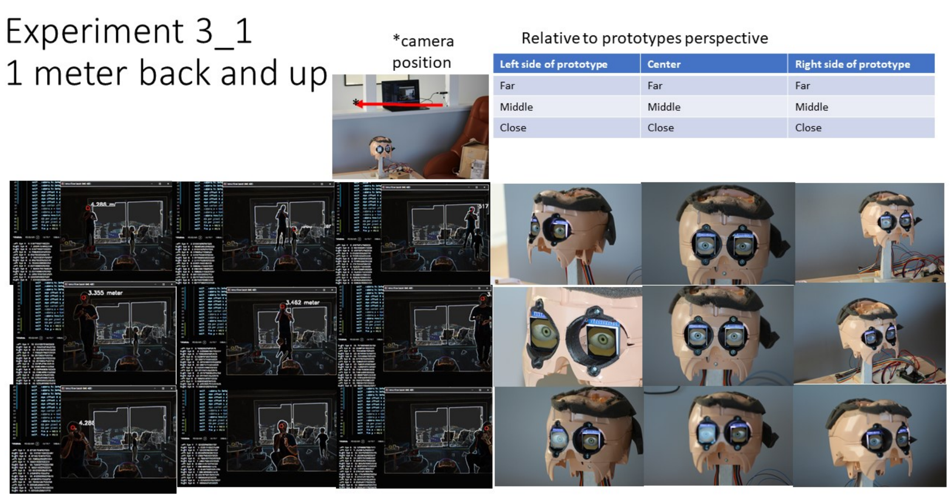

Figure 4.4: Single Person Tracking Multiple Camera Position

* The target person had a small visitor in the field of view that influenced the center far position object detection. Logic for handling multiple faces or objects will be needed in a case like this.

Discussion

The benefit of moving the camera behind the eye prototype is a larger field of vision directly ahead of the eye prototype. For future applications together with the full patient simulator it could be an idea to position the camera higher up and to one end of the room for full overview of people inside it. This will create some challenges in needing good transformations on the detected positions to where the eyes should be angled. If the patient simulator is moved during simulations the transformations will need to be updated. On board sensors for head orientation and potentially using the camera for detecting the patient simulator head position in the room can be sufficient in updating flexible transformations.

Camera Position 2

Second change in camera position was 83 cm to the left of the prototype (lateral sideways) location, 73 cm behind the prototype in z(lateral backwards/forwards) orientation and same y (vertical 32cm above) location as the experiments in section 4.2.3 camera position 1. See figure 4.5 for an image of the position relative to the eye prototype.

Table 4.4: Results - Single Person Tracking Multiple Camera Position 2

Position	Object Detection	Eye Tracking
Left Close	OK	OK
Left Middle	OK	Not OK
Left Far	OK	OK
Center Close	OK	Not OK
Center Middle	OK	Not OK
Center Far	OK	Not OK
Right Close	OK	Not OK
Right Middle	OK	Not OK
Right Far	OK	OK

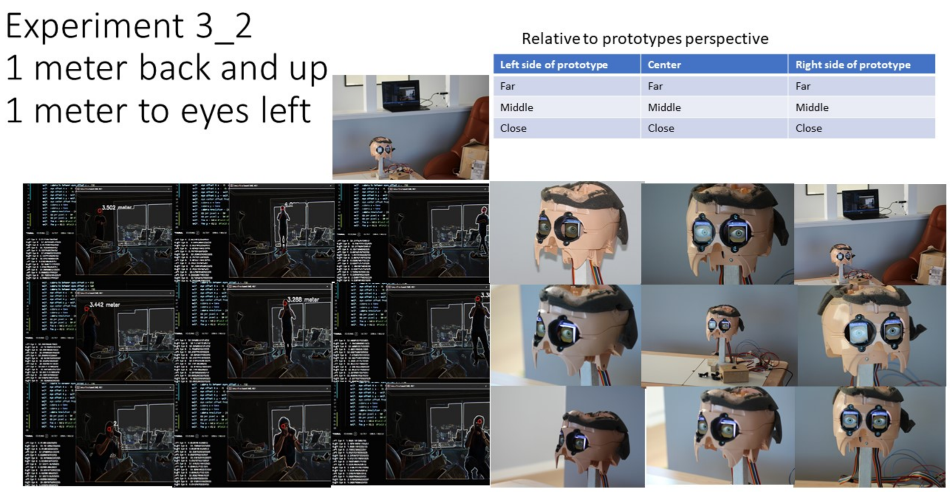

Figure 4.5: Single Person Tracking Multiple Camera Position

Discussion

Object detection worked well in all experiments. It only struggled in some positions when the camera for documenting eye tracking was held up to the face of the target person. Eye tracking on most of the positions in this camera position was not accurate. This can come from angulation and measurement offsets of the camera to between eyes of prototype not being correct. More work and experiments on the camera positions relative to eyes and the calculated angle for the eyes is needed to determine why the tracking failed in most of the predetermined positions for this test.

Chapter 5

Conclusion

This object detection eye prototype worked well on a single person in the field of view. It lacks logic for handling multiple objects. The depth camera and the LCD eyes worked well to make a realistic simulation of eyes keeping eye contact with a target person that moves around to different positions and distances.

The neural networks needed for this application is relatively easy to train and not much novel work needs to be done to train them on different objects. Collecting the datasets and training are the time consuming tasks along with the logic of where the eyes should focus.

Smaller optimised networks can be run with good enough performance for smooth eye tracking on edge devices like Raspberry Pi 4 with the calculation assistance of a USB Accelerator.[18] [5]

5.1 Further Work

To further develop this prototype or for implementation into the SimMan Patient Simulator the list below can be used as a starting point. The items listed are in no particular order.

- Allow for camera positions where camera and head does not point in the same direction
- Object detection networks that detect other items than faces
- Logic to handle multiple different objects detected
- Realistic eye focusing logic
- Optimised network to perform on edge device.
- Upgrade Edge device with TPU device like Coral USB Accelerator. [18]
- Implement eye movement behaviour related to medical symptoms the SimMan Patient Simulator is simulating. [15]
- Work on lenses for the monitor eyes that does not distort the eyes in the monitor they way the current ones do.

Bibliography

[1] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viégas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, and X. Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. Software available from tensorflow.org.

[2] Adafruit. Adafruit animated snake eyes bonnet for raspberry pi. `https://learn.adafruit.com/animated-snake-eyes-bonnet-for-raspberry-pi`. Accessed: 2021-04-16.

[3] Adafruit. Pi_eyes (python code for adafruits lcd eye monitors). `https://github.com/adafruit/Pi_Eyes/`, 2020.

[4] S. Bozinovski. Reminder of the first paper on transfer learning in neural networks, 1976. *Informatica 44*, 291–302, 2020.

[5] E. Electronics. Raspberry pi 3 and 4 performance comparrison. `https://www.youtube.com/watch?v=TiOKvOrYNII&t=216s`. Accessed: 2021-04-23.

[6] R. P. Foundation. Rasperry pi 3 mode b+. `https://www.raspberrypi.org/products/raspberry-pi-3-model-b-plus/`. Accessed: 2021-04-16.

[7] K. Fukushima. Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position. *Biological cybernetics*, 36(4):193–202, 1980.

[8] S. S. Haykin. *Neural networks and learning machines*. Pearson Education, Upper Saddle River, NJ, third edition, 2009.

[9] A. G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, and H. Adam. Mobilenets: Efficient convolutional neural networks for mobile vision applications. *CoRR*, abs/1704.04861, 2017.

[10] Intel. Intel realsense depth camera d435. `https://www.intelrealsense.com/depth-camera-d435/`. Accessed: 2021-04-16.

[11] Intel. Python wrapper for intel realsense sdk 2.0. `https://pypi.org/project/pyrealsense2/`. Accessed: 2021-05-06.

[12] J. Kunnas. person_event-detect-recognition. `https://github.com/jkunnas58/person_event-detect-recognition`, 2021.

[13] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[14] W. Liu, D. Anguelov, D. Erhan, C. Szegedy, S. Reed, C.-Y. Fu, and A. C. Berg. Ssd: Single shot multibox detector. *Lecture Notes in Computer Science*, page 21–37, 2016.

[15] L. Medical. Laerdal's patient simulator simman 3g plus. `https://laerdal.com/products/simulation-training/emergency-care-trauma/simman-3g/`. Accessed: 2021-04-16.

[16] U. Michelucci. *Applied Deep Learning: A Case-Based Approach to Understanding Deep Neural Networks*. Apress, USA, 1st edition, 2018.

[17] Python. Socket - python standard library. `https://docs.python.org/3/library/socket.html`. Accessed: 2021-04-22.

[18] G. Research. Coral tpu usb accelerator. `https://coral.ai/products/accelerator`. Accessed: 2021-04-23.

[19] L. Vladimirov. Tensorflow object detection api tutorial. `https://tensorflow-object-detection-api-tutorial.readthedocs.io/en/latest/index.html#`. Accessed: 2021-04-18.

[20] S. Yang, P. Luo, C. C. Loy, and X. Tang. Wider face: A face detection benchmark. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016.

[21] yeephycho. tensorflow-face-detection. `https://github.com/yeephycho/tensorflow-face-detection`, 2020.

Appendix A

Object Detection Code

```
#!/usr/bin/python
# -*- coding: utf-8 -*-
# pylint: disable=C0103
# pylint: disable=E1101

from os import X_OK
import sys
import time
import numpy as np
import tensorflow as tf
import cv2
import collections
import six
import PIL.Image as Image
import pyrealsense2 as rs
from utils import label_map_util
from utils import visualization_utils_color as vis_util
from send import SendData
from coordinate_converter import ConvertCoordinates
import copy


# Path to frozen detection graph. This is the actual model ...
    that is used for the object detection.
PATH_TO_CKPT = './model/frozen_inference_graph_face.pb'
PATH_TO_CKPT = ...
    r"C:\dev\tensorflow\workspace\tensorflow-face-detection-master\model\frozen_inferenc

```

```
# List of the strings that is used to add correct label ...
    for each box.
PATH_TO_LABELS = './protos/face_label_map.pbtxt'
PATH_TO_LABELS = ...
    r"C:\dev\tensorflow\workspace\tensorflow-face-detection-master\protos\face_label_map

NUM_CLASSES = 2

label_map = label_map_util.load_labelmap(PATH_TO_LABELS)
categories = ...
    label_map_util.convert_label_map_to_categories(label_map, ...
    max_num_classes=NUM_CLASSES, use_display_name=True)
category_index = ...
    label_map_util.create_category_index(categories)


def get_eye_focus_coordinate(
    image,
    boxes,
    classes,
    scores,
    category_index,
    instance_masks=None,
    instance_boundaries=None,
    keypoints=None,
    keypoint_scores=None,
    keypoint_edges=None,
    track_ids=None,
    use_normalized_coordinates=False,
    max_boxes_to_draw=20,
    min_score_thresh=.5,
    agnostic_mode=False,
    line_thickness=4,
    mask_alpha=.4,
    groundtruth_box_visualization_color='black',
    skip_boxes=False,
    skip_scores=False,
    skip_labels=False,
    skip_track_ids=False):
    """

    """
      # Create a display string (and color) for every box ...
    location, group any boxes
  # that correspond to the same location.
    box_to_display_str_map = collections.defaultdict(list)
    box_to_color_map = collections.defaultdict(str)
    box_to_instance_masks_map = {}
    box_to_keypoints_map = collections.defaultdict(list)
```

```
    if not max_boxes_to_draw:
        max_boxes_to_draw = boxes.shape[0]
    for i in range(min(max_boxes_to_draw, boxes.shape[0])):
        if scores is None or scores[i] > min_score_thresh:
            box = tuple(boxes[i].tolist())
            if instance_masks is not None:
                box_to_instance_masks_map[box] = ...
instance_masks[i]
            if keypoints is not None:
                box_to_keypoints_map[box].extend(keypoints[i])
            if scores is None:
                box_to_color_map[box] = 'black'
            else:
                if not agnostic_mode:
                    if classes[i] in category_index.keys():
                        class_name = ...
category_index[classes[i]]['name']
                    else:
                        class_name = 'N/A'
                    display_str = '{}: {}%'.format(
                        class_name,
                        int(100*scores[i]))
                else:
                    display_str = 'score: ...
{}%'.format(int(100 * scores[i]))
                ...
box_to_display_str_map[box].append(display_str)
                if agnostic_mode:
                    box_to_color_map[box] = 'DarkOrange'
                else:
                    box_to_color_map[box] = ...
groundtruth_box_visualization_color
            #Export location of box in relative or ...
absolute coordinates
                image_for_size = ...
Image.fromarray(np.uint8(image)).convert('RGB')
                im_width, im_height = image_for_size.size
                ymin, xmin, ymax, xmax = box
                if use_normalized_coordinates:
                    (left, right, top, bottom) = (xmin * ...
im_width, xmax * im_width,
                                            ymin * ...
im_height, ymax * im_height)
                else:
                    (left, right, top, bottom) = (xmin, ...
xmax, ymin, ymax)
                return (left, right, top, bottom)
            return False

```

```

class TensoflowFaceDector(object):
    def __init__(self, PATH_TO_CKPT):
        """Tensorflow detector
        """

        self.detection_graph = tf.Graph()
        with self.detection_graph.as_default():
            od_graph_def = tf.compat.v1.GraphDef()
            with tf.io.gfile.GFile(PATH_TO_CKPT, 'rb') as fid:
                serialized_graph = fid.read()
                od_graph_def.ParseFromString(serialized_graph)
                tf.import_graph_def(od_graph_def, name='')


        with self.detection_graph.as_default():
            config = tf.compat.v1.ConfigProto()
            config.gpu_options.allow_growth = True
            self.sess = ...
    tf.compat.v1.Session(graph=self.detection_graph, ...
    config=config)
            self.windowNotSet = True


    def run(self, image):
        """image: bgr image
        return (boxes, scores, classes, num_detections)
        """

        image_np = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)

        # the array based representation of the image will ...
    be used later in order to prepare the
        # result image with boxes and labels on it.
        # Expand dimensions since the model expects images ...
    to have shape: [1, None, None, 3]
        image_np_expanded = np.expand_dims(image_np, axis=0)
        image_tensor = ...
    self.detection_graph.get_tensor_by_name('image_tensor:0')
        # Each box represents a part of the image where a ...
    particular object was detected.
        boxes = ...
    self.detection_graph.get_tensor_by_name('detection_boxes:0')
        # Each score represent how level of confidence for ...
    each of the objects.
        # Score is shown on the result image, together ...
    with the class label.
        scores = ...
    self.detection_graph.get_tensor_by_name('detection_scores:0')
```

```
        classes = ...
    self.detection_graph.get_tensor_by_name('detection_classes:0')
        num_detections = ...
    self.detection_graph.get_tensor_by_name('num_detections:0')
        # Actual detection.
        start_time = time.time()
        (boxes, scores, classes, num_detections) = ...
    self.sess.run(
            [boxes, scores, classes, num_detections],
            feed_dict={image_tensor: image_np_expanded})
        elapsed_time = time.time() - start_time
        # print('inference time cost: ...
    {}'.format(elapsed_time))

        return (boxes, scores, classes, num_detections)




if __name__ == "__main__":
#     import sys
#     if len(sys.argv) != 2:
#         print ("usage:%s (cameraID | filename) Detect faces\
#  in the video example:%s 0"%(sys.argv[0], sys.argv[0]))
#         exit(1)

#     try:
#       camID = int(sys.argv[1])
#     except:
#       camID = sys.argv[1]



    camID = 0
    # Configure depth and color streams
    pipeline = rs.pipeline()
    config = rs.config()

    # Get device product line for setting a supporting ...
    resolution
    pipeline_wrapper = rs.pipeline_wrapper(pipeline)
    pipeline_profile = config.resolve(pipeline_wrapper)
    device = pipeline_profile.get_device()
    device_product_line = ...
    str(device.get_info(rs.camera_info.product_line))

    config.enable_stream(rs.stream.depth, 640, 480, ...
    rs.format.z16, 30)

```

```
    if device_product_line == 'L500':
        config.enable_stream(rs.stream.color, 960, 540, ...
    rs.format.bgr8, 30)
    else:
        config.enable_stream(rs.stream.color, 640, 480, ...
    rs.format.bgr8, 30)

    # Start streaming
    pipeline.start(config)
    tDetector = TensoflowFaceDector(PATH_TO_CKPT)

    cap = cv2.VideoCapture(camID)
    windowNotSet = True

    #socket sending
    send_data_to_socket = SendData()
    send_data_to_socket.setup_server_sending()

    #Converterclass
    coordinate_converter = ConvertCoordinates()
    coordinate_converter.set_camera_resolution((640,480)) ...
    #camera resolution
    ...
    coordinate_converter.set_eye_center_offset_from_screen(-10) ...
    # distance to fictive eye center behind monitor
    coordinate_converter.set_mode('3D')
    # coordinate_converter.set_xyz(50,50,1000) #default ...
    point to look at top left looking at head
    depth_previous = 0.8

    while True:
        # Wait for a coherent pair of frames: depth and color
        frames = pipeline.wait_for_frames()
        depth_frame = frames.get_depth_frame()
        color_frame = frames.get_color_frame()
        if not depth_frame or not color_frame:
            continue

        # Convert images to numpy arrays
        depth_image = np.asanyarray(depth_frame.get_data())
        color_image = np.asanyarray(color_frame.get_data())

        # Apply colormap on depth image (image must be ...
    converted to 8-bit per pixel first)
        # depth_colormap = ...
    cv2.applyColorMap(cv2.convertScaleAbs(depth_image, ...
    alpha=0.03), cv2.COLORMAP_JET)

        # depth_colormap_dim = depth_colormap.shape
```

```
        # color_colormap_dim = color_image.shape

        # # If depth and color resolutions are different, ...
    resize color image to match depth image for display
        # if depth_colormap_dim != color_colormap_dim:
        #     resized_color_image = ...
    cv2.resize(color_image, dsize=(depth_colormap_dim[1], ...
    depth_colormap_dim[0]), interpolation=cv2.INTER_AREA)
        #     images = np.hstack((resized_color_image, ...
    depth_colormap))
        # else:
        #     images = np.hstack((color_image, ...
    depth_colormap))

        image = color_image
        image_depth = depth_image

        # ret, image = cap.read()
        # if ret == 0:
        #     break

        [h, w] = image.shape[:2]
        # print (h, w)
        # image = cv2.flip(image, 1)
        # print(image.shape)
        (boxes, scores, classes, num_detections) = ...
    tDetector.run(image)

        # vis_util.visualize_boxes_and_labels_on_image_array(
        #     image,
        #     np.squeeze(boxes),
        #     np.squeeze(classes).astype(np.int32),
        #     np.squeeze(scores),
        #     category_index,
        #     use_normalized_coordinates=True,
        #     line_thickness=4)



        box_test = get_eye_focus_coordinate(
            image,
            np.squeeze(boxes),
            np.squeeze(classes).astype(np.int32),
            np.squeeze(scores),
            category_index,
            use_normalized_coordinates=True,
            max_boxes_to_draw=200,
            min_score_thresh=.3,
            agnostic_mode=False)
```

```
        # print(box_test) #example (0.23469042778015137, ...
    0.30845338106155396, 0.7406021952629089, ...
    0.5217226147651672)


        if box_test:
            # print(box_test)
            box_int_list = [0,0,0,0]
            for i in range(4):
                box_int_list[i] = int(box_test[i])

            # 1/3 from the left of the box
            x_location = ...
    int(((box_int_list[1]-box_int_list[0])*1/3)+box_int_list[0])
            #1/3 from the top.
            y_location = ...
    int(((box_int_list[3]-box_int_list[2])*1/3)+box_int_list[2])
            # get depth from realsense camera
            depth_location = ...
    depth_frame.get_distance(x_location, y_location) # ...
    depth in xx units
            depth_location_left = ...
    depth_frame.get_distance(x_location, y_location+10)
            depth_location_right = ...
    depth_frame.get_distance(x_location, y_location-10)
            depth_location = ...
    np.mean([depth_location,depth_location_right,depth_location_left])
            # Write some Text
            if depth_location < 0.01:
                depth_location = depth_previous

            depth_previous = copy.deepcopy(depth_location)

            font                   = cv2.FONT_HERSHEY_SIMPLEX
            bottomLeftCornerOfText = ...
    (box_int_list[1],box_int_list[2])
            fontScale              = 1
            fontColor              = (255,255,255)
            lineType               = 2

            cv2.putText(image,f'{round(depth_location,3)} ...
    meter',
                bottomLeftCornerOfText,
                font,
                fontScale,
                fontColor,
                lineType)


```

```
            circle_radius = 10
            circle_color = (0,0,255)
            circle_coordinates = (x_location,y_location)
            ...
cv2.circle(image,circle_coordinates,circle_radius, ...
circle_color, thickness=-1 )

            coordinate_converter.set_xyz(
                circle_coordinates[0],
                circle_coordinates[1],
                depth_location*1000
            )

            try:
                str_data_to_send = ...
coordinate_converter.get_eye_coordinates()
                # print(str_data_to_send)
                ...
send_data_to_socket.send_data(str_data_to_send)
                time.sleep(0.05)
            except Exception:
                # str_data_to_send = ...
coordinate_converter.get_eye_coordinates()
                # ...
send_data_to_socket.send_data(str_data_to_send)
                time.sleep(0.05)

            # print(circle_coordinates[0], ...
circle_coordinates[1])
            # print(f'depth to focus point {depth_location}')
            # except:
            #     print('passed')
        else:
            # print('no output')
            pass


        if windowNotSet is True:
            cv2.namedWindow("tensorflow based (%d, %d)" % ...
(w, h), cv2.WINDOW_NORMAL)
            windowNotSet = False

        cv2.imshow("tensorflow based (%d, %d)" % (w, h), ...
image)
        k = cv2.waitKey(1) & 0xff
        if k == ord('q') or k == 27:
            break

    cap.release()
```

Appendix B

Coordinate Converter Code

```
import math
import numpy as np


class ConvertCoordinates():
    """
    Class that recieves pixel X Y and depth from camera, ...
    Camera to eye offsets.
    Converts detected focus point to eye simulator eye X ...
    and Y position
    for left and right eye individually in 3D mode.

    Different options of complexity of data conversion ...
    available:

    2D_simple: uses pixel location as fraction of ...
    resolution and puts that
    fraction between -30 and 30 degree eye rotation for x ...
    and y

    2D: uses pixel location as fraction of resolution and ...
    puts that
    fraction between -30 and 30 degree eye rotation for x ...
    and y PLUS takes
    into account depth for prominence calculation

    3D: calculate individual eye rotation to be properly ...
    oriented to detected
```

```
    object


    coordinate system has xyz 0 at camera.
    positive directions references from doll heads perspective
     x+ right, y+up. z+ away from head in eye direction

     Distances in mm millimeter

     center of "fake eye ball" is set at 10mm behind screen.
     can be changed with __eye_center_offset_from_screen ...
    variable
    """

    def __init__(self) -> None:
        self.__eyeXR = None
        self.__eyeYR = None
        self.__eyeXL = None
        self.__eyeYL = None
        self.__mode = ''
        self.__camera_to_between_eyes_offset_x = 180
        self.__camera_to_between_eyes_offset_y = -100
        self.__camera_to_between_eyes_offset_z = -70
        self.__eye_offset_R_x = 31  + ...
    self.__camera_to_between_eyes_offset_x
        self.__eye_offset_L_x = -31 + ...
    self.__camera_to_between_eyes_offset_x
        self.__eye_offset_R_y = ...
    self.__camera_to_between_eyes_offset_y
        self.__eye_offset_L_y = ...
    self.__camera_to_between_eyes_offset_y
        self.__eye_center_offset_from_screen = -10
        self.__camera_resolution = (640,480)
        #field of view[degree] horisontal axis ( x) 64 ...
    from datasheet
        #field of view[degree] of view vertical axis ( y) ...
    41 from datasheet
        self.__fov_x = 64/2
        self.__fov_y = 41/2


    def __calc_eye_coordinates(self,x,y,z):
        if self.__mode == '3D':
            self.__calc_3D(x,y,z)
        elif self.__mode == '2D':
            self.__calc_2D(x,y,z)
        else:
            self.__calc_2D_simple(x,y,z)

```

```
    def __calc_3D(self,x,y,depth):
        """
        X pixel for detection object to focus on
        Y pixel for detection object to focus on
        depth mm to detected object to focus on

        takes in pixel coordinates and depth and sets eye X,Y
        rotation (degrees from 0,0 straight ahead)
        """

        #Calculate X,Y angle relative to eye references
        pix_per_degree_x = ...
    self.__camera_resolution[0]/self.__fov_x
        degrees_from_left = x / pix_per_degree_x
        degrees_from_center = degrees_from_left - ...
    (self.__fov_x/2)
        x_distance_from_center_mm = \
            math.sin(math.radians(degrees_from_center)) * ...
    depth
        z_distance_from_center_mm = \
            math.cos(math.radians(degrees_from_center))* depth
        z_distance_from_center_mm = \
            z_distance_from_center_mm - ...
    self.__camera_to_between_eyes_offset_z

        pix_per_degree_y = ...
    self.__camera_resolution[1]/self.__fov_y
        degrees_from_top = y / pix_per_degree_y
        degrees_from_center = degrees_from_top - ...
    (self.__fov_y/2)
        y_distance_from_center_mm = \
                        ...
    math.sin(math.radians(degrees_from_center)) * depth


        #x and y coordinates relative to eye positions
        left_eye_x = x_distance_from_center_mm - ...
    self.__eye_offset_L_x
        left_eye_y = y_distance_from_center_mm - ...
    self.__eye_offset_L_y

        right_eye_x = x_distance_from_center_mm - ...
    self.__eye_offset_R_x
        right_eye_y = y_distance_from_center_mm - ...
    self.__eye_offset_R_y


        left_eye_x_angle = ...
    math.asin(left_eye_x/z_distance_from_center_mm)
```

```
        left_eye_y_angle = ...
    math.asin(left_eye_y/z_distance_from_center_mm)

        right_eye_x_angle = ...
    math.asin(right_eye_x/z_distance_from_center_mm)
        right_eye_y_angle = ...
    math.asin(right_eye_y/z_distance_from_center_mm)


        self.__eyeXL = math.degrees(left_eye_x_angle)
        self.__eyeYL = -math.degrees(left_eye_y_angle)
        self.__eyeXR = math.degrees(right_eye_x_angle)
        self.__eyeYR = -math.degrees(right_eye_y_angle)

    def print_current_eye_angles(self):
        print(f'Left Eye X: {self.__eyeXL}')
        print(f'Left Eye Y: {self.__eyeYL}')
        print(f'Right Eye X: {self.__eyeXR}')
        print(f'Right Eye X: {self.__eyeYR}')


    def __calc_2D(self, x,y,z):
        self.__eyeXR = round(-30 + (x / ...
    self.__camera_resolution[0]) * 60, 2)
        self.__eyeYR = round(-30 + (1-(y / ...
    self.__camera_resolution[1])) * 60, 2)
        self.__eyeXL = self.__eyeXR
        self.__eyeYL = self.__eyeYR

        # adjust eyes X location when close to head
        # prominence is the amount of degree you
        # rotate the eyes towards each other
        prominence = 10 - ((z/1000)*10)
        if prominence < 0:
            prominence = 0
        if prominence > 10:
            prominence = 10
        self.__eyeXR -= prominence
        self.__eyeXL += prominence

    def __calc_2D_simple(self,x,y,z):
        self.__eyeXR = round(-30 + (x / ...
    self.__camera_resolution[0]) * 60, 2)
        self.__eyeYR = round(-30 + (1-(y / ...
    self.__camera_resolution[1])) * 60, 2)
        self.__eyeXL = self.__eyeXR
        self.__eyeYL = self.__eyeYR

    def get_eye_coordinates(self):
```

```
        # returns a ...
    string:'eye_right_x,eye_right_y,e_left_x,e_left_y'
        #  with coordinates for eyes
        return ...
    f'{self.__eyeXR},{self.__eyeYR},{self.__eyeXL},{self.__eyeYL}'

    def set_xyz(self, x , y , z = 1000):
        self.__calc_eye_coordinates(x,y,z)

    def set_mode(self,mode_str):
        self.__mode = mode_str

    def set_eye_center_offset_from_screen(self, distance_z):
        self.__eye_center_offset_from_screen = distance_z

    def set_camera_resolution(self,resolution_tuple):
        # takes in a tuple in this format (x,y) , (640,480)
        self.__camera_resolution = resolution_tuple


def main():
    print('test')

if __name__ == '__main__':
    main()
```

Appendix C

Send Data Code

```
import socket
import numpy as np
import time


class SendData():
    """
    Send data class. This class sets up a sending server ...
    waiting for clients to
    connect.

    class has a class method that sets up the server in ...
    setup_server_sending()

    The function send_data(data_to_send) sends the data as ...
    string in utf-8
    encoding

    """

    def __init__(self) -> None:
        self.__host = '192.168.191.125' # loopback ...
    interface address (localhost)
        self.__port = 65432 # Port to listen on ...
    (non-privileged ports are > 1023)
        self.__socket = socket.socket(socket.AF_INET, ...
    socket.SOCK_STREAM)
        self.__connection = None
```

```
        self.__address = None

    def setup_server_sending(self):
        print("Server Started waiting for client to ...
    connect ")
        self.__socket.bind((self.__host, self.__port))
        self.__socket.listen(5)
        self.__connection, self.__address = ...
    self.__socket.accept()
        print('Connected to', self.__address)

    def send_data(self,my_data):
        # my_data = f'{self.__eyeX},{self.__eyeY}'
        # print(my_data)
        my_data_bytes = bytes(my_data, 'utf-8')
        # print('length of bytes: ', len(my_data_bytes))
        self.__connection.send(my_data_bytes)

    def set_host_ip(self, ip):
        #set host ip as string '192.168.1.1'
        self.__host = ip

    def set_port(self, port):
        #set port as int
        self.__port


class RandomData():

    def __init__(self) -> None:
        self.oldtime = time.time()
        self.x1 = np.random.randint(-30, 30, None)
        self.y1 = np.random.randint(-30, 30, None)

    def random_data(self):
        if time.time() - self.oldtime > 2:
            x1 = np.random.randint(-30, 30, None)         ...
    # Dummy eye x
            y1 = np.random.randint(-30, 30, None)        # ...
    Dummy dummy eye y
        else:
            x1 = self.x1
            y1 = self.y1

        return x1, y1

def main():
    random_data = RandomData()
    send_data = SendData()
```

```
    send_data.setup_server_sending()

    while True:
        # eye_x = 0   # 30 left to -30 right looking at ...
    the dool
        # eye_y = 0 # -30 looking down 30 looking up
        eye_x, eye_y = random_data.random_data()
        ...
    send_data.send_data(f'{eye_x},{eye_y},{eye_x},{eye_y}')
        time.sleep(0.5)


if __name__ == '__main__':
    main()
```

Appendix D

Receive Data Code

```
import socket

class RecieveData():
    """
    Class that starts a socket connection and recieves eye ...
    coordinates
    for eye simulator to use

    """
    def __init__(self):
        self.__host = '192.168.191.125'
        self.__port = 65432
        self.__eyeXR = 30
        self.__eyeYR = 30
        self.__eyeXL = 30
        self.__eyeYL = 30
        self.__socket = socket.socket(socket.AF_INET, ...
    socket.SOCK_STREAM)
        self.__connected_to_socket = False

    def process_data_from_server(self,x):
        self.__eyeXR , self.__eyeYR , self.__eyeXL , ...
    self.__eyeYL = x.split(",")


    def connect_to_server(self):
        try:
```

```
            self.__socket = socket.socket(socket.AF_INET, ...
        socket.SOCK_STREAM)
            self.__socket.connect((self.__host, self.__port))
            self.__connected_to_socket = True
        except:
            self.__connected_to_socket = False

    def get_data_from_connection(self):
            data = self.__socket.recv(1024).decode('utf-8')
            self.process_data_from_server(data)


    def get_eye_coordinates_str(self):
        self.get_data_from_connection()
        return  self.__eyeXR,\
                self.__eyeYR,\
                self.__eyeXL,\
                self.__eyeYL

    def get_eye_coordinates_float(self):
        self.get_data_from_connection()
        return  float(self.__eyeXR),\
                float(self.__eyeYR),\
                float(self.__eyeXL),\
                float(self.__eyeYL)

    def close_socket(self):
        self.__socket.shutdown()
        self.__socket.close()

    def get_socket_connected_status(self):
        return self.__connected_to_socket

    def set_socket_connected_status(self, bool):
        self.__connected_to_socket = bool

    def set_host_ip(self, host_ip):
        # Set host ip as string: example: '192.168.2.1'
        self.__host = host_ip

    def set_host_port(self, host_port):
        # Set host port as integer: example: 65432
        self.__port = host_port

def main():
    eye_coordinates = RecieveData()
    eye_coordinates.connect_to_server()
    while True:
```

```
        eyex,eyey,eyex2,eyey2 = ...
    eye_coordinates.get_eye_coordinates_str()
        print(f'EyeX: {eyex} , EyeY: {eyey}, EyeX2: ...
    {eyex2} , EyeY2: {eyey2}')

if __name__ == "__main__":
    while True:
        main()
```

Appendix E

Eye Prototype Code

```
#!/usr/bin/python

# Code originates from
# ...
   https://learn.adafruit.com/animated-snake-eyes-bonnet-for-raspberry-pi/ ...
   ...
#  ... software-installation
# https://github.com/adafruit/Pi_Eyes/
# Expanded to be controled by joystick and switch
# Start with python eyes.py --radius 200 or any other ...
   number to change eye size
# Set AUTOBLINK to False to disable eyelids
# The joystick and selection switch are connected to the ...
   bonnet extention card
# On pin 0-1 (joystick) and 22,23,24 & 27 (selection switch)
# The configuration below (INPUT CONFIG and INIT GLOBALS) ...
   can be changed to
# enable and disable inputs/switches, add buttons, and ...
   configure the
# movement speed, duration of movement and movement of eye lid
#--------
import argparse
import math
import pi3d
import random
import threading
import time
import RPi.GPIO as GPIO
```

```
from svg.path import Path, parse_path
from xml.dom.minidom import parse
from gfxutil import *
from snake_eyes_bonnet import SnakeEyesBonnet

# for object detection use
from obj_detection_data_socket import RecieveData
import threading
import queue


# INPUT CONFIG for eye motion ----------------------------------------------
# Configuration of the inputs
JOYSTICK_X_IN   = 0    # Analog input for eye horiz pos (-1 = auto)
JOYSTICK_Y_IN   = 1    # Analog input for eye vert position (")
PUPIL_IN        = -1    # Analog input for pupil control (-1 = auto)
JOYSTICK_X_FLIP = False # If True, reverse stick X axis
JOYSTICK_Y_FLIP = False # If True, reverse stick Y axis
PUPIL_IN_FLIP   = False # If True, reverse reading from PUPIL_IN
TRACKING        = True  # If True, eyelid tracks pupil
PUPIL_SMOOTH    = 16    # If > 0, filter input from PUPIL_IN
PUPIL_MIN       = 0.0   # Lower analog range from PUPIL_IN
PUPIL_MAX       = 1.0   # Upper --"--
SW_PIN1     = 22        # 22 Inputs from the switch
SW_PIN2     = 23        # 23   on pin 22,23,24,27
SW_PIN3     = 24        # 24   set to -1 to disable
SW_PIN4     = 27        # 27
AUTOBLINK       = True  # If True, eyes blink autonomously


# GPIO initialization ------------------------------------------------------
# Only initialize if they are defined.
GPIO.setmode(GPIO.BCM)
if SW_PIN1 ≥ 0:
     GPIO.setup(SW_PIN1, GPIO.IN, pull_up_down=GPIO.PUD_UP)
if SW_PIN2  ≥ 0:
     GPIO.setup(SW_PIN2 , GPIO.IN, pull_up_down=GPIO.PUD_UP)
if SW_PIN3 ≥ 0:
     GPIO.setup(SW_PIN3, GPIO.IN, pull_up_down=GPIO.PUD_UP)
if SW_PIN4 ≥ 0:
     GPIO.setup(SW_PIN4, GPIO.IN, pull_up_down=GPIO.PUD_UP)
```

```
def checkGPIO():
    """
    Used to check the status of the input switch.
    Returns (int) 1-6 depending on the program selected.
    Program 1: Random movement, normal speed
    Program 2: Random movement, slow speed
    Program 3: Random movement, fast speed
    Program 4: Joystick Control, manual control
    Program 5: Random movement, x-axis (horizontal random ...
    movement)
    Program 6: Eyelids closed
    """
    program = 1
    if GPIO.input(SW_PIN1) == GPIO.LOW:
        program = 1
        if GPIO.input(SW_PIN4) == GPIO.LOW:
            program = 4
    elif GPIO.input(SW_PIN2) == GPIO.LOW:
        program = 2
        if GPIO.input(SW_PIN4) == GPIO.LOW:
            program = 5
    elif GPIO.input(SW_PIN3) == GPIO.LOW:
        program = 3
        if GPIO.input(SW_PIN4) == GPIO.LOW:
            program = 6
    return program

# ADC stuff ...
    ----------------------------------------------------------------
# The ADC is used to read the joystick position
# ADC channels are read and stored in a separate thread to ...
    avoid slowdown
# from blocking operations. The animation loop can read at ...
    its leisure.

if JOYSTICK_X_IN ≥ 0 or JOYSTICK_Y_IN ≥ 0 or PUPIL_IN ≥ 0:
    bonnet = SnakeEyesBonnet(daemon=True)
    bonnet.setup_channel(JOYSTICK_X_IN, ...
    reverse=JOYSTICK_X_FLIP)
    bonnet.setup_channel(JOYSTICK_Y_IN, ...
    reverse=JOYSTICK_Y_FLIP)
    bonnet.setup_channel(PUPIL_IN, reverse=PUPIL_IN_FLIP)
    bonnet.start()
```

```
# Load SVG file, extract paths & convert to point lists ...
    ---------------------
dom               = parse("graphics/eye.svg")
vb                = get_view_box(dom)
pupilMinPts       = get_points(dom, "pupilMin"      , 32, ...
    True , True )
pupilMaxPts       = get_points(dom, "pupilMax"      , 32, ...
    True , True )
irisPts           = get_points(dom, "iris"          , 32, ...
    True , True )
scleraFrontPts    = get_points(dom, "scleraFront"   ,  0, ...
    False, False)
scleraBackPts     = get_points(dom, "scleraBack"    ,  0, ...
    False, False)
upperLidClosedPts = get_points(dom, "upperLidClosed", 33, ...
    False, True )
upperLidOpenPts   = get_points(dom, "upperLidOpen"  , 33, ...
    False, True )
upperLidEdgePts   = get_points(dom, "upperLidEdge"  , 33, ...
    False, False)
lowerLidClosedPts = get_points(dom, "lowerLidClosed", 33, ...
    False, False)
lowerLidOpenPts   = get_points(dom, "lowerLidOpen"  , 33, ...
    False, False)
lowerLidEdgePts   = get_points(dom, "lowerLidEdge"  , 33, ...
    False, False)


# Set up display and initialize pi3d ...
    ---------------------------------------
DISPLAY = pi3d.Display.create(samples=4)
DISPLAY.set_background(0, 0, 0, 1) # r,g,b,alpha
# eyeRadius is the size, in pixels, at which the whole eye ...
    will be rendered
# onscreen.  eyePosition, also pixels, is the offset (left ...
    or right) from
# the center point of the screen to the center of each ...
    eye.  This geometry
# is explained more in-depth in fbx2.c.
eyePosition = DISPLAY.width / 4
eyeRadius   = 100  # 128  # Default; use 240 for IPS screens


# Argument to change the size of the entire eye on startup
parser = argparse.ArgumentParser()
parser.add_argument("--radius", type=int)
args = parser.parse_args()
if args.radius:
    eyeRadius = args.radius
```

```
eyeRadius = 240

# A 2D camera is used, mostly to allow for pixel-accurate ... eye placement,
# but also because perspective isn't really helpful or ... needed here, and
# also this allows eyelids to be handled somewhat easily ... as 2D planes.
# Line of sight is down Z axis, allowing conventional X/Y ... cartesion
# coords for 2D positions.
cam    = pi3d.Camera(is_3d=False, at=(0,0,0), eye=(0,0,-1000))
shader = pi3d.Shader("uv_light")
light  = pi3d.Light(lightpos=(0, -500, -500), ... lightamb=(0.2, 0.2, 0.2))


# Load texture maps ... --------------------------------------------------------
irisMap   = pi3d.Texture("graphics/iris.jpg"  , mipmap=False,
              filter=pi3d.GL_LINEAR)
scleraMap = pi3d.Texture("graphics/sclera.png", mipmap=False,
              filter=pi3d.GL_LINEAR, blend=True)
lidMap    = pi3d.Texture("graphics/lid.png"   , mipmap=False,
              filter=pi3d.GL_LINEAR, blend=True)
# U/V map may be useful for debugging texture placement; ... not normally used
#uvMap     = pi3d.Texture("graphics/uv.png"    , mipmap=False,
#              filter=pi3d.GL_LINEAR, blend=False, ... m_repeat=True)


# Initialize static geometry ... -----------------------------------------------
# Transform point lists to eye dimensions
scale_points(pupilMinPts      , vb, eyeRadius)
scale_points(pupilMaxPts      , vb, eyeRadius)
scale_points(irisPts          , vb, eyeRadius)
scale_points(scleraFrontPts   , vb, eyeRadius)
scale_points(scleraBackPts    , vb, eyeRadius)
scale_points(upperLidClosedPts, vb, eyeRadius)
scale_points(upperLidOpenPts  , vb, eyeRadius)
scale_points(upperLidEdgePts  , vb, eyeRadius)
scale_points(lowerLidClosedPts, vb, eyeRadius)
scale_points(lowerLidOpenPts  , vb, eyeRadius)
scale_points(lowerLidEdgePts  , vb, eyeRadius)

# Regenerating flexible object geometry (such as eyelids ... during blinks, or
```

```
# iris during pupil dilation) is CPU intensive, can ...
    noticably slow things
# down, especially on single-core boards.  To reduce this ...
    load somewhat,
# determine a size change threshold below which ...
    regeneration will not occur;
# roughly equal to 1/4 pixel, since 4x4 area sampling is used.

# Determine change in pupil size to trigger iris geometry ...
    regen
irisRegenThreshold = 0.0
a = points_bounds(pupilMinPts) # Bounds of pupil at min ...
    size (in pixels)
b = points_bounds(pupilMaxPts) # " at max size
maxDist = max(abs(a[0] - b[0]), abs(a[1] - b[1]), # ...
    Determine distance of max
              abs(a[2] - b[2]), abs(a[3] - b[3])) # ...
    variance around each edge
# maxDist is motion range in pixels as pupil scales ...
    between 0.0 and 1.0.
# 1.0 / maxDist is one pixel's worth of scale range.  Need ...
    1/4 that...
if maxDist > 0: irisRegenThreshold = 0.25 / maxDist

# Determine change in eyelid values needed to trigger ...
    geometry regen.
# This is done a little differently than the ...
    pupils...instead of bounds,
# the distance between the middle points of the open and ...
    closed eyelid
# paths is evaluated, then similar 1/4 pixel threshold is ...
    determined.
upperLidRegenThreshold = 0.0
lowerLidRegenThreshold = 0.0
p1 = upperLidOpenPts[len(upperLidOpenPts) // 2]
p2 = upperLidClosedPts[len(upperLidClosedPts) // 2]
dx = p2[0] - p1[0]
dy = p2[1] - p1[1]
d  = dx * dx + dy * dy
if d > 0: upperLidRegenThreshold = 0.25 / math.sqrt(d)
p1 = lowerLidOpenPts[len(lowerLidOpenPts) // 2]
p2 = lowerLidClosedPts[len(lowerLidClosedPts) // 2]
dx = p2[0] - p1[0]
dy = p2[1] - p1[1]
d  = dx * dx + dy * dy
if d > 0: lowerLidRegenThreshold = 0.25 / math.sqrt(d)

# Generate initial iris meshes; vertex elements will get ...
    replaced on
```

```
# a per-frame basis in the main loop, this just sets up ...
    textures, etc.
rightIris = mesh_init((32, 4), (0, 0.5 / irisMap.iy), ...
    True, False)
rightIris.set_textures([irisMap])
rightIris.set_shader(shader)
# Left iris map U value is offset by 0.5; effectively a ...
    180 degree
# rotation, so it's less obvious that the same texture is ...
    in use on both.
leftIris = mesh_init((32, 4), (0.5, 0.5 / irisMap.iy), ...
    True, False)
leftIris.set_textures([irisMap])
leftIris.set_shader(shader)
irisZ = zangle(irisPts, eyeRadius)[0] * 0.99 # Get iris Z ...
    depth, for later

# Eyelid meshes are likewise temporary; texture ...
    coordinates are
# assigned here but geometry is dynamically regenerated in ...
    main loop.
leftUpperEyelid = mesh_init((33, 5), (0, 0.5 / lidMap.iy), ...
    False, True)
leftUpperEyelid.set_textures([lidMap])
leftUpperEyelid.set_shader(shader)
leftLowerEyelid = mesh_init((33, 5), (0, 0.5 / lidMap.iy), ...
    False, True)
leftLowerEyelid.set_textures([lidMap])
leftLowerEyelid.set_shader(shader)

rightUpperEyelid = mesh_init((33, 5), (0, 0.5 / ...
    lidMap.iy), False, True)
rightUpperEyelid.set_textures([lidMap])
rightUpperEyelid.set_shader(shader)
rightLowerEyelid = mesh_init((33, 5), (0, 0.5 / ...
    lidMap.iy), False, True)
rightLowerEyelid.set_textures([lidMap])
rightLowerEyelid.set_shader(shader)

# Generate scleras for each eye...start with a 2D shape ...
    for lathing...
angle1 = zangle(scleraFrontPts, eyeRadius)[1] # Sclera ...
    front angle
angle2 = zangle(scleraBackPts , eyeRadius)[1] # " back angle
aRange = 180 - angle1 - angle2
pts    = []
for i in range(24):
    ca, sa = pi3d.Utility.from_polar((90 - angle1) - ...
    aRange * i / 23)
```

```
    pts.append((ca * eyeRadius, sa * eyeRadius))

# Scleras are generated independently (object isn't re-used) so each
# may have a different image map (heterochromia, corneal scar, or the
# same image map can be offset on one so the repetition isn't obvious).
leftEye = pi3d.Lathe(path=pts, sides=64)
leftEye.set_textures([scleraMap])
leftEye.set_shader(shader)
re_axis(leftEye, 0)
rightEye = pi3d.Lathe(path=pts, sides=64)
rightEye.set_textures([scleraMap])
rightEye.set_shader(shader)
re_axis(rightEye, 0.5) # Image map offset = 180 degree rotation


# INIT GLOBALS --------------------------------------------------------
mykeys = pi3d.Keyboard() # For capturing key presses
startX       = random.uniform(-30.0, 30.0)
n            = math.sqrt(900.0 - startX * startX)
startY       = random.uniform(-n, n)
destX        = startX
destY        = startY
curX         = startX
curY         = startY
moveDuration = random.uniform(0.075, 0.175)
holdDuration = random.uniform(0.1, 1.1)
startTime    = 0.0
isMoving     = False

startXR      = random.uniform(-30.0, 30.0)
n            = math.sqrt(900.0 - startX * startX)
startYR      = random.uniform(-n, n)
destXR       = startXR
destYR       = startYR
curXR        = startXR
curYR        = startYR
moveDurationR = random.uniform(0.075, 0.175)
holdDurationR = random.uniform(0.1, 1.1)
startTimeR    = 0.0
isMovingR     = False

frames        = 0
beginningTime = time.time()

```

```
rightEye.positionX(-eyePosition)
rightIris.positionX(-eyePosition)
rightUpperEyelid.positionX(-eyePosition)
rightUpperEyelid.positionZ(-eyeRadius - 42)
rightLowerEyelid.positionX(-eyePosition)
rightLowerEyelid.positionZ(-eyeRadius - 42)

leftEye.positionX(eyePosition)
leftIris.positionX(eyePosition)
leftUpperEyelid.positionX(eyePosition)
leftUpperEyelid.positionZ(-eyeRadius - 42)
leftLowerEyelid.positionX(eyePosition)
leftLowerEyelid.positionZ(-eyeRadius - 42)

currentPupilScale       =  0.5
prevPupilScale          = -1.0 # Force regen on first frame
prevLeftUpperLidWeight  = 0.5
prevLeftLowerLidWeight  = 0.5
prevRightUpperLidWeight = 0.5
prevRightLowerLidWeight = 0.5
prevLeftUpperLidPts  = points_interp(upperLidOpenPts, ...
    upperLidClosedPts, 0.5)
prevLeftLowerLidPts  = points_interp(lowerLidOpenPts, ...
    lowerLidClosedPts, 0.5)
prevRightUpperLidPts = points_interp(upperLidOpenPts, ...
    upperLidClosedPts, 0.5)
prevRightLowerLidPts = points_interp(lowerLidOpenPts, ...
    lowerLidClosedPts, 0.5)

luRegen = True
llRegen = True
ruRegen = True
rlRegen = True

timeOfLastBlink = 0.0
timeToNextBlink = 1.0
# These are per-eye (left, right) to allow winking:
blinkStateLeft      = 0 # NOBLINK
blinkStateRight     = 0
blinkDurationLeft   = 0.1
blinkDurationRight  = 0.1
blinkStartTimeLeft  = 0
blinkStartTimeRight = 0

trackingPos = 0.3
trackingPosR = 0.3

# initialize socket class, used if option 6 is selected.
eye_coordinate_socket = RecieveData()
```

```
dnn_queue = queue.Queue()
curX2 = 20
curY2 = 20
last_x = 0
last_y = 0
last_x2 = 0
last_y2 = 0

# Generate one frame of imagery
def frame(p):
    global startX, startY, destX, destY, curX, curY, ...
    curX2, curY2
    global startXR, startYR, destXR, destYR, curXR, curYR
    global moveDuration, holdDuration, startTime, isMoving
    global moveDurationR, holdDurationR, startTimeR, isMovingR
    global frames
    global leftIris, rightIris
    global pupilMinPts, pupilMaxPts, irisPts, irisZ
    global leftEye, rightEye
    global leftUpperEyelid, leftLowerEyelid, ...
    rightUpperEyelid, rightLowerEyelid
    global upperLidOpenPts, upperLidClosedPts, ...
    lowerLidOpenPts, lowerLidClosedPts
    global upperLidEdgePts, lowerLidEdgePts
    global prevLeftUpperLidPts, prevLeftLowerLidPts, ...
    prevRightUpperLidPts, prevRightLowerLidPts
    global leftUpperEyelid, leftLowerEyelid, ...
    rightUpperEyelid, rightLowerEyelid
    global prevLeftUpperLidWeight, prevLeftLowerLidWeight, ...
    prevRightUpperLidWeight, prevRightLowerLidWeight
    global prevPupilScale
    global irisRegenThreshold, upperLidRegenThreshold, ...
    lowerLidRegenThreshold
    global luRegen, llRegen, ruRegen, rlRegen
    global timeOfLastBlink, timeToNextBlink
    global blinkStateLeft, blinkStateRight
    global blinkDurationLeft, blinkDurationRight
    global blinkStartTimeLeft, blinkStartTimeRight
    global trackingPos
    global trackingPosR
    global eye_coordinate_socket
    global dnn_queue
    global last_x
    global last_y
    global last_x2
    global last_y2

    DISPLAY.loop_running()

```

```
    now = time.time()
    dt  = now - startTime
    dtR  = now - startTimeR

    frames += 1
#   if(now > beginningTime):
#       print(frames/(now-beginningTime))

    if checkGPIO() == 4: # Joystick control / manual movement
        curX = bonnet.channel[JOYSTICK_X_IN].value
        curY = bonnet.channel[JOYSTICK_Y_IN].value
        curX = -30.0 + curX * 60.0
        curY = -30.0 + curY * 60.0
    else :
        # Autonomous eye position
        if isMoving == True:
            if dt <= moveDuration:
                scale        = (now - startTime) / ...
    moveDuration
                # Ease in/out curve: 3*t^2-2*t^3
                scale = 3.0 * scale * scale - 2.0 * scale ...
    * scale * scale
                curX         = startX + (destX - startX) * ...
    scale
                curY         = startY + (destY - startY) * ...
    scale
            else:
                startX       = destX
                startY       = destY
                curX         = destX
                curY         = destY
                if checkGPIO() == 2: # Random movement ...
    slow speed (3-8 sec between movement)
                    holdDuration = random.uniform(3, 8)
                elif checkGPIO() == 3: # Random movement ...
    fast speed (0.5-1 sec between movement)
                    holdDuration = random.uniform(0.5, 1)
                else:
                    holdDuration = random.uniform(0.5, 6) ...
    # Default movement speed (checkGPIO==1) (0.5-6 sec)
                #holdDuration = random.uniform(0.1, 1.1)
                startTime    = now
                isMoving     = False
        else:
            if dt >= holdDuration:
                destX        = random.uniform(-30.0, 30.0)
                n            = math.sqrt(225.0 - (destX/2) ...
    * (destX/2))
                # MOVE Y AXIS
```

```
            if checkGPIO() == 5: # Random movement ...
only in X-axis (y is set to 0)
                destY = 0  # random.uniform(0,0)
            elif checkGPIO() == 3: # Fast speed and ...
full movement in y-axis
                destY = random.uniform(-n, n)
            elif checkGPIO() == 2: # Slow speed, half ...
movement in y-axis
                destY = random.uniform(-n/2, n/2)
            else:
                destY = random.uniform(-n, n)

            moveDuration = random.uniform(0.075, 0.175)
            startTime    = now
            isMoving     = True

# Regenerate iris geometry only if size changed by ≥ ...
1/4 pixel
if abs(p - prevPupilScale) ≥ irisRegenThreshold:
    # Interpolate points between min and max pupil sizes
    interPupil = points_interp(pupilMinPts, ...
pupilMaxPts, p)
    # Generate mesh between interpolated pupil and ...
iris bounds
    mesh = points_mesh((None, interPupil, irisPts), 4, ...
-irisZ, True)
    # Assign to both eyes
    leftIris.re_init(pts=mesh)
    rightIris.re_init(pts=mesh)
    prevPupilScale = p

# Eyelid WIP
if AUTOBLINK and (now - timeOfLastBlink) ≥ ...
timeToNextBlink:
    timeOfLastBlink = now
    duration        = random.uniform(0.035, 0.06)
    if blinkStateLeft != 1:
        blinkStateLeft     = 1 # ENBLINK
        blinkStartTimeLeft = now
        blinkDurationLeft  = duration
    if blinkStateRight != 1:
        blinkStateRight     = 1 # ENBLINK
        blinkStartTimeRight = now
        blinkDurationRight  = duration
    timeToNextBlink = duration * 3 + ...
random.uniform(1.0, 4.0)

if blinkStateLeft: # Left eye currently winking/blinking?
    # Check if blink time has elapsed...
```

```
        if (now - blinkStartTimeLeft) ≥ blinkDurationLeft:
            blinkStateLeft += 1
            if blinkStateLeft > 2:
                blinkStateLeft = 0 # NOBLINK
            else:
                blinkDurationLeft *= 2.0
                blinkStartTimeLeft = now

    if blinkStateRight: # Right eye currently ...
    winking/blinking?
        # Check if blink time has elapsed...
        if (now - blinkStartTimeRight) ≥ blinkDurationRight:
            blinkStateRight += 1
            if blinkStateRight > 2:
                blinkStateRight = 0 # NOBLINK
            else:
                blinkDurationRight *= 2.0
                blinkStartTimeRight = now

    if checkGPIO() == 6:
        #hacked for test of eye tracking
        # AUTOBLINK = False #disables blinking
        try:
            if not dnn_queue.empty():
                first_out = dnn_queue.get()
                curX = first_out[0]
                curY = first_out[1]
                curX2 = first_out[2]
                curY2 = first_out[3]
                last_x = curX
                last_y = curY
                last_x2 = curX2
                last_y2 = curY2
            else:
                curX = last_x
                curY = last_y
                curX2 = last_x2
                curY2 = last_y2

        except Exception as e:
            print(f'assigning queue items failed: {e}')



    if TRACKING:
        n = 0.4 - curY / 60.0
        if   n < 0.0: n = 0.0
        elif n > 1.0: n = 1.0
        trackingPos = (trackingPos * 3.0 + n) * 0.25
```

```


    if blinkStateLeft:
        n = (now - blinkStartTimeLeft) / blinkDurationLeft
        if n > 1.0: n = 1.0
        if blinkStateLeft == 2: n = 1.0 - n
    else:
        n = 0.0
    newLeftUpperLidWeight = trackingPos + (n * (1.0 - ...
    trackingPos))
    newLeftLowerLidWeight = (1.0 - trackingPos) + (n * ...
    trackingPos)

    if blinkStateRight:
        n = (now - blinkStartTimeRight) / blinkDurationRight
        if n > 1.0: n = 1.0
        if blinkStateRight == 2: n = 1.0 - n
    else:
        n = 0.0

    newRightUpperLidWeight = trackingPos + (n * (1.0 - ...
    trackingPos))
    newRightLowerLidWeight = (1.0 - trackingPos) + (n * ...
    trackingPos)

    if (luRegen or (abs(newLeftUpperLidWeight - ...
    prevLeftUpperLidWeight) ≥
      upperLidRegenThreshold)):
        newLeftUpperLidPts = points_interp(upperLidOpenPts,
          upperLidClosedPts, newLeftUpperLidWeight)
        if newLeftUpperLidWeight > prevLeftUpperLidWeight:
            leftUpperEyelid.re_init(pts=points_mesh(
              (upperLidEdgePts, prevLeftUpperLidPts,
              newLeftUpperLidPts), 5, 0, False))
        else:
            leftUpperEyelid.re_init(pts=points_mesh(
              (upperLidEdgePts, newLeftUpperLidPts,
              prevLeftUpperLidPts), 5, 0, False))
        prevLeftUpperLidPts    = newLeftUpperLidPts
        prevLeftUpperLidWeight = newLeftUpperLidWeight
        luRegen = True
    else:
        luRegen = False

    if (llRegen or (abs(newLeftLowerLidWeight - ...
    prevLeftLowerLidWeight) ≥
      lowerLidRegenThreshold)):
        newLeftLowerLidPts = points_interp(lowerLidOpenPts,
          lowerLidClosedPts, newLeftLowerLidWeight)
```

```
        if newLeftLowerLidWeight > prevLeftLowerLidWeight:
            leftLowerEyelid.re_init(pts=points_mesh(
              (lowerLidEdgePts, prevLeftLowerLidPts,
              newLeftLowerLidPts), 5, 0, False))
        else:
            leftLowerEyelid.re_init(pts=points_mesh(
              (lowerLidEdgePts, newLeftLowerLidPts,
              prevLeftLowerLidPts), 5, 0, False))
        prevLeftLowerLidWeight = newLeftLowerLidWeight
        prevLeftLowerLidPts    = newLeftLowerLidPts
        llRegen = True
    else:
        llRegen = False

    if (ruRegen or (abs(newRightUpperLidWeight - ...
    prevRightUpperLidWeight) ≥
      upperLidRegenThreshold)):
        newRightUpperLidPts = points_interp(upperLidOpenPts,
          upperLidClosedPts, newRightUpperLidWeight)
        if newRightUpperLidWeight > prevRightUpperLidWeight:
            rightUpperEyelid.re_init(pts=points_mesh(
              (upperLidEdgePts, prevRightUpperLidPts,
              newRightUpperLidPts), 5, 0, True))
        else:
            rightUpperEyelid.re_init(pts=points_mesh(
              (upperLidEdgePts, newRightUpperLidPts,
              prevRightUpperLidPts), 5, 0, True))
        prevRightUpperLidWeight = newRightUpperLidWeight
        prevRightUpperLidPts    = newRightUpperLidPts
        ruRegen = True
    else:
        ruRegen = False

    if (rlRegen or (abs(newRightLowerLidWeight - ...
    prevRightLowerLidWeight) ≥
      lowerLidRegenThreshold)):
        newRightLowerLidPts = points_interp(lowerLidOpenPts,
          lowerLidClosedPts, newRightLowerLidWeight)
        if newRightLowerLidWeight > prevRightLowerLidWeight:
            rightLowerEyelid.re_init(pts=points_mesh(
              (lowerLidEdgePts, prevRightLowerLidPts,
              newRightLowerLidPts), 5, 0, True))
        else:
            rightLowerEyelid.re_init(pts=points_mesh(
              (lowerLidEdgePts, newRightLowerLidPts,
              prevRightLowerLidPts), 5, 0, True))
        prevRightLowerLidWeight = newRightLowerLidWeight
        prevRightLowerLidPts    = newRightLowerLidPts
        rlRegen = True
```

```
    else:
        rlRegen = False

    if GPIO != 6:
        convergence = 2.0

        rightIris.rotateToX(curY)
        rightIris.rotateToY(curX - convergence)
        rightIris.draw()
        rightEye.rotateToX(curY)
        rightEye.rotateToY(curX - convergence)
        rightEye.draw()

        # Left eye (on screen right)

        leftIris.rotateToX(curY)
        leftIris.rotateToY(curX + convergence)
        leftIris.draw()
        leftEye.rotateToX(curY)
        leftEye.rotateToY(curX + convergence)
        leftEye.draw()
    else:
        convergence = 0

        rightIris.rotateToX(curY)
        rightIris.rotateToY(curX - convergence)
        rightIris.draw()
        rightEye.rotateToX(curY)
        rightEye.rotateToY(curX - convergence)
        rightEye.draw()

        # Left eye (on screen right)

        leftIris.rotateToX(curY2)
        leftIris.rotateToY(curX2 + convergence)
        leftIris.draw()
        leftEye.rotateToX(curY2)
        leftEye.rotateToY(curX2 + convergence)
        leftEye.draw()

    leftUpperEyelid.draw()
    leftLowerEyelid.draw()
    rightUpperEyelid.draw()
    rightLowerEyelid.draw()

    k = mykeys.read()
    if k==1:
        mykeys.close()
        DISPLAY.stop()
```

```
        exit(0)


def fill_queue():
    global dnn_queue
    global eye_coordinate_socket
    global curX, curY, curX2, curY2

    while True:
        if checkGPIO() == 6:
            #modified for test of eye tracking
            # AUTOBLINK = False #disables blinking
            try:
                if not ...
    eye_coordinate_socket.get_socket_connected_status():
                    eye_coordinate_socket.connect_to_server()
            except Exception:
                ...
    eye_coordinate_socket.set_socket_connected_status(False)

            try:
                ext_curX, ext_curY, ext_curX2, ext_curY2 = ...
    eye_coordinate_socket.get_eye_coordinates_float()
                dnn_queue.put((ext_curX, ext_curY, ...
    ext_curX2, ext_curY2))

            except Exception as e:
                ...
    eye_coordinate_socket.set_socket_connected_status(False)    ...

                print(f'failed to get datafrom socket and ...
    put to queue: {e}')

        if checkGPIO() != 6 and ...
    eye_coordinate_socket.get_socket_connected_status():
            ...
    eye_coordinate_socket.set_socket_connected_status(False)
            try:
                eye_coordinate_socket.close_socket()
            except Exception:
                pass
            time.sleep(2)



def split( # Recursive simulated pupil response when no ...
    analog sensor
  startValue, # Pupil scale starting value (0.0 to 1.0)
  endValue,   # Pupil scale ending value (")
```

```
  duration,    # Start-to-end time, floating-point seconds
  range):      # +/- random pupil scale at midpoint
    startTime = time.time()
    if range ≥ 0.125: # Limit subdvision count, because ...
    recursion
        duration *= 0.5 # Split time & range in half for ...
    subdivision,
        range    *= 0.5 # then pick random center point ...
    within range:
        midValue  = ((startValue + endValue - range) * 0.5 +
                     random.uniform(0.0, range))
        split(startValue, midValue, duration, range)
        split(midValue  , endValue, duration, range)
    else: # No more subdivisons, do iris motion...
        dv = endValue - startValue
        while True:
            dt = time.time() - startTime
            if dt ≥ duration: break
            v = startValue + dv * dt / duration
            if   v < PUPIL_MIN: v = PUPIL_MIN
            elif v > PUPIL_MAX: v = PUPIL_MAX
            frame(v) # Draw frame w/interim pupil scale value


#MAKE THREAD FOR EXTERNAL DATA AND START IT.
get_data_thread = threading.Thread(target=fill_queue)
get_data_thread.deamon = True
get_data_thread.start()



# MAIN LOOP -- runs continuously ...
    -------------------------------------------
while True:

    if PUPIL_IN ≥ 0: # Pupil scale from sensor
        v = bonnet.channel[PUPIL_IN].value
        # If you need to calibrate PUPIL_MIN and MAX,
        # add a 'print v' here for testing.
        if   v < PUPIL_MIN: v = PUPIL_MIN
        elif v > PUPIL_MAX: v = PUPIL_MAX
        # Scale to 0.0 to 1.0:
        v = (v - PUPIL_MIN) / (PUPIL_MAX - PUPIL_MIN)
        if PUPIL_SMOOTH > 0:
            v = ((currentPupilScale * (PUPIL_SMOOTH - 1) + ...
    v) /
                PUPIL_SMOOTH)
        frame(v)
    else: # Fractal auto pupil scale
```

```
        v = random.random()
        split(currentPupilScale, v, 4.0, 1.0)
    currentPupilScale = v
```