Evaluating the Accuracy of Gaze Detection for Moving Character

by

Sanath Narasimhan

THESIS

Submitted in Partial Fulfillment of the Requirements for the Degree of

Master of Science in Computer Science

THE UNIVERSITY OF TEXAS AT ARLINGTON

August 2020

Arlington,Texas

Supervising committee:

Deok Gun Park, Supervising Professor

Won Hwa Kim

Vassilis Athitsos

ABSTRACT

Evaluating the Accuracy of Gaze Detection for Moving Character

Sanath Narasimhan,

The University of Texas at Arlington, 2020

Supervising Professor: Deok Gun Park

Joint attention, where a caregiver and an infant follow each other's eye gaze plays an important role in the learning of language for new-born infants. To study joint attention, it is required to record and analyze the joint attention in a naturalistic environment. For this, we can use the head-mounted eye tracker. However, natural interaction involves the body movement which affects the accuracy of the measurement. In this work, I evaluated the accuracy of the eye-tracking system in the three different scenarios: when the subject is sitting still in front of the target when the subject looks at the target while moving the head sitting at a fixed location when the subject moves freely within the setup while looking at the target. To track the head pose we use an in-built plugin within Pupil Player that detects surface markers (april13-tags) within the world camera feed and generates camera position and orientation in the marker coordinate system. The evaluation shows that the gaze position accuracy and precision are better when the scene is lit under White LED lights compared with Yellow LEDs. The recordings were also replicated in a virtual environment created in Unity3D that are modeled after real-world experimental setup. We expect this evaluation will contribute to the development of the simulated environments for

implementing developmental robotics which can provide simulated experiences such as interactions of a mother and a child during various stages of infant development (from fetus stage to 12 months of age).

TABLE OF CONTENTS

LIST OF ILLUSTRATIONS

CHAPTER 1

Introduction

1.1 Goal

Pupil Labs has a product, the Pupil core, an open-source eye-tracking system with hardware and software. The main goal is to successfully translate the eye position from a user to the character within the virtual environment we have generated allowing us to visualize how the eyes move while the user interacts with an object. The idea is to integrate the data from the eye-tracking device into a live 3D environment to create straight-line pointers from the eyes of a character to the location where the character is supposed to look based on the user's eyes.

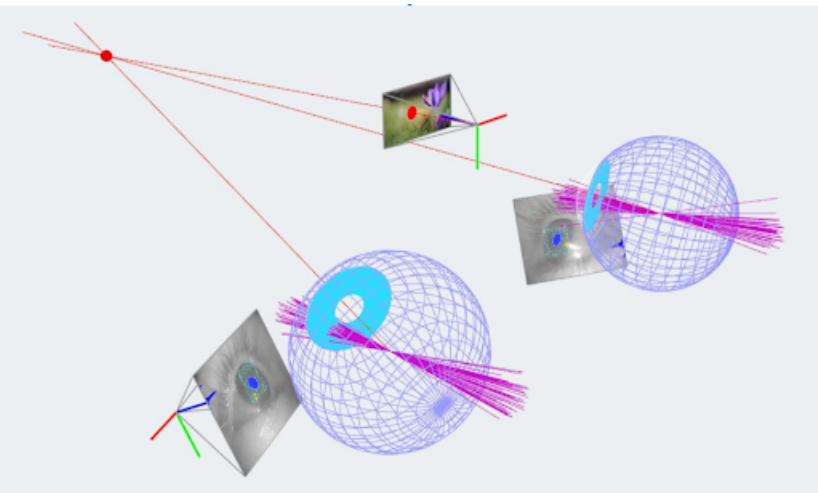

Figure 1.1. Gaze estimation.

This helps capture the exact eye placement when a subject like a child interacts with various objects in an environment for the first time. The challenging part would

be the work-flow to create animations of the eyes. This can be achieved by creating a custom script to invoke the pupil labs API in order to use the raw eye-tracking data within the 3D environment.

1.2 Motivation

The idea of creating a virtual agent that learns to navigate its surroundings by using partial and noise inputs, similar to human beings, has been one of the main driving factors in the field of Reinforcement learning. Countless research papers are available where traditional and modern approaches are applied to control agents in environments like video games. Recent advanced models like the Intrinsic Curiosity Model [14] which allows the agent to try to explore the surrounding environment with sparse external reward because of curiosity and Gated-Attention Architectures for Task-Oriented Language Grounding [7] which uses input in English language to control the agents location in the environment are tested in environments like the VizDoom [15] and Super Mario bros environments.

Figure 1.2. Common Game environments.

Research in the field of Neuroscience has shown that there are nuances to how an infant learns language from its surroundings and the parents. Language acquisition depends on the perception of the objects through various human sensory inputs like observing the general shape and fall of the object, feeling the texture and the weight of the object, and predicting the trajectory an object will take by observing the effects of gravity on it. One of the research [16] concluded that joint attention plays a major role in the accumulation of an infant's vocabulary [17]. Joint attention [4] is a phenomenon where a subject is able to share attention with an interacting agent over an object simply by learning to estimate the location of interest by observing the agent's head pose and gaze direction. An example would be when a parent brings a toy, like a red cube into the line of sight of the child to teach the word "red", the child first notices the parent's face and then tries to estimate where they are looking.

Developmental timescale for joint attention skills (adapted from Kaplan and Hafner 2006b)

Age (months)	Attention detection	Attention manipulation	Social coordination	Intentional understanding
0–3 months	Mutual gaze through eye contact	Mutual gaze through maintaining eye contact	Protoconversation, simple turn taking mediated by caregiver	Early identification with other persons
6 months	Sensitivity only to the left/right side that the caregiver is gazing at	Simple forms of attention manipulation	Shared routines: caregiver-child conversational games	Animate-Inanimate distinction; physical/social causality distinction
9 months	Gaze angle detection, fixation of first salient object	Imperative pointing to ask for object/food	Joint activities, imitative games of caregiver's movement	First goal-directed behavior
12 months	Gaze angle detection, fixation of any salient object	Declarative pointing, draw attention with gestures	Joint activities and imitative games for goal sharing	Goal understanding, behavior understood as goal directed
18 months	Gaze following toward object outside field of view	First predications with words and gestures	Coordination of joint action plans	Intentional understanding, same goal for different action plans

Figure 1.3. Joint attention skill development.

In order to implement phenomenons like joint attention we need to develop new methods to help capture them and test them out in virtual environments for helping improve the current state-of-the-art models in mimicking human-like intelligent behaviours.

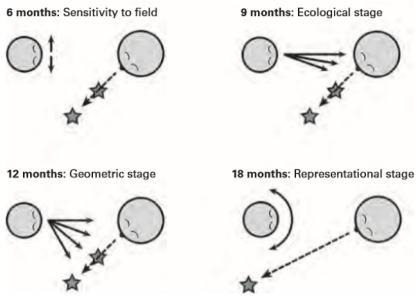

Figure 1.4. Visualization of joint attention.

1.3 Outcome

The eye-tracker is mainly for capturing the infant eye placement in our IDS experiments that we will conduct. One of the key ways we can observe how an infant learns is by looking at how their eyes move when a direction is given to them, for example, if the infant is asked to look at the red cube, the infant will turn towards the object and start scanning the shape and the color of the object. The key is where the infant is looking while observing a new object, this information can give us a good insight into the process of learning and recognition that happens in the brain when a new object is being interacted with. With these set of experiments we can evaluate the pipeline that allows us to capture fixation data from a real world subject and translate it into a virtual agent in a simulation.

Figure 1.5. Pupil core eye-tracker.

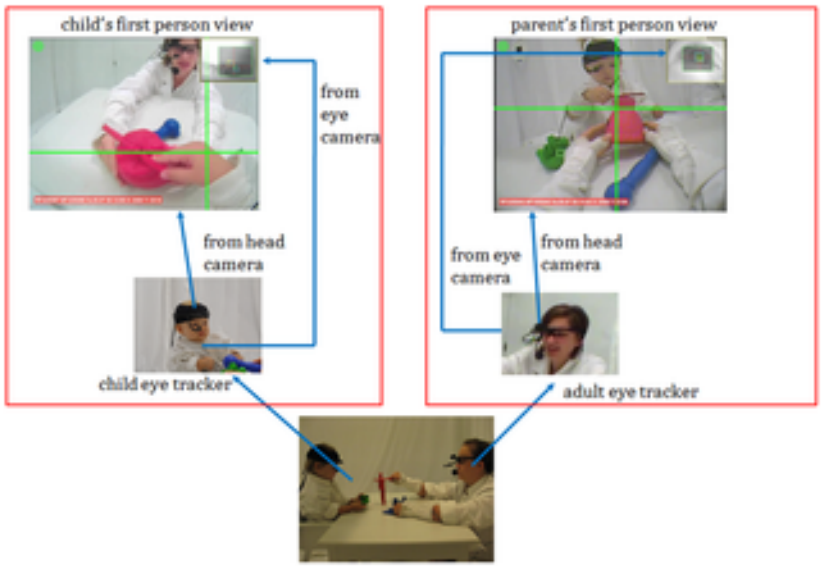

Figure 1.6. Infant sustained attention.

1.4 Background

Evaluation of eye tracking is a well-known problem with some standard definitions of Accuracy and Precision. In this scenario, the Pupil Core's real-world accuracy is defined by the distance error and the precision by the directional (angular) error. The Pupil Core plugins run with OpenCV, hence use the Y-down convention coordinate system for any data generated [12]. The gaze point detection accuracy and precision depend upon the confidence of the pupil detection at every frame. First, a normalized screen coordinate of the gaze is generated within world camera feed pixel coordinates which are then converted to 3D world coordinates. The generated fixation points rely on the gaze data and are in millimeters, with the center of the world camera lens as the origin and are corrected for camera intrinsic. They have been several experiments testing the real-world performance of the Pupil Pro and other similar eye-trackers [9] along with the manufacturer's guarantee that the 0.6 deg of accuracy and 0.08 deg of precision under ideal conditions.

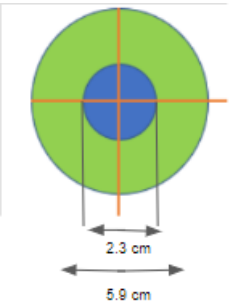

Figure 1.7. A target.

There can be multiple error sources like user error, change is the distance to the target (Parallax error), slippage (slight movement of the headset), and limitations of human fixation (0.5 deg). We need to keep in mind that these errors might get amplified when using in the virtual environment due to a change in convention and scale.

The Head-pose plugin mainly relies on the detection of surface markers and requires that there are sufficient frames with the majority of the markers visible to the world camera. The generated 3D model can be customized by specifying the origin marker within the detected marker set and uses the real-world marker length and the scaling factor. A marker cache is maintained ensuring that the relevant surfaces are tracked with a minimum of 2 markers for each surface. If a surface has 4 or more markers, the probability of it being tracked is higher even when all markers of that surface are not visible in the current frame. The generated rotations from this plugin that uses OpenCV are in Rodrigues axis notation.

Unity3D is a game development engine that consists of various packages and tools useful in the creation of a virtual environment. The scale of Unity is in meters and uses a Y-up convention for the representation of the coordinate system. General rotation notations are in either Quaternion or Euler notation. Unity3D consists of an in-built function SphereCast that can make a point of origin, direction, and a

radius, to cast a cylindrical ray with the thickness of the radius, returning the point of collision of the ray. This will be useful for retrieving the gaze location in the virtual world using the captured real-world data.

In both the virtual and the real-world environments, the targets would lie along the X-Y plane and the distance of the target would be the Z coordinate of a gaze point. The accuracy can be defined in this case as the deviation of the virtual gaze point to real-world point, taking the Euclid distance along the X-Y plane. The precision would be the difference in the viewing angles.

CHAPTER 2

Environment

2.1 Environment Setup

The experiment is conducted in a cubicle with the targets displayed on a 27"; Dell monitor and the subject sitting in front of the monitor on a rolling chair. Targets are designed based on an experiment that evaluates the real-world accuracy of the eye tracker [8] and [12]. Each Target consists of an outer circle, green, 5.9 centimeters in diameter and an inner circle, blue, 2.3 centimeters in diameter with a total of 54 targets displayed over six rows and nine columns evenly spaced with red lines marking the center of individual targets.

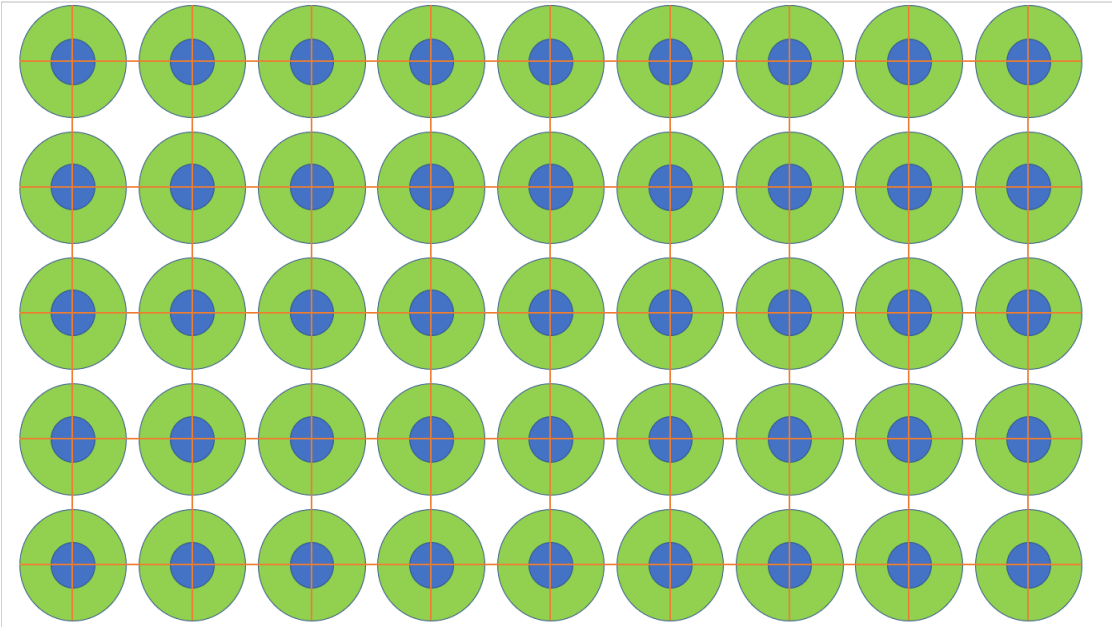

Figure 2.1. A set of 5X9 targets evenly spaced.

There are 12 markers (each with a unique ID) placed in the cubical, 4 on each corner of the Targets display monitor, 4 on each corner of the laptop display, and 4 on objects within the cubical. These are april36-tags, used to detect objects within a camera feed using OpenCV and the Pupil Capture software's Head pose tracker plugin uses this feature to detect the camera's position with respect to a currently visible marker as the origin (the scale of the position is the side length of the marker in the real world). We have kept the bottom right corner of the Targets display as the fixed origin (marker number 4). In order to help with the visibility of the markers an LED light array is used which has a set of yellow lights and white lights, interchangeable. The subject can move around in the cubical and uses the Pupil Capture software to record the fixation points over the targets.

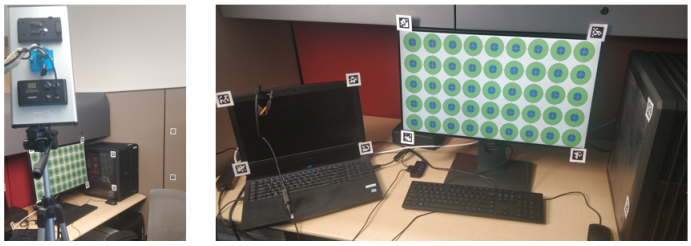

Figure 2.2. The experiment setup with apriltag markers, and LED lights to illumination.

2.2 3D Environment

The Unity3D environment consists of an identical camera to that of the world camera of the eye-tracker whose location will be determined based off of the head pose tracking data. The image of the targets is added as a texture to a 3D plane GameObject with a smaller plane representing the bottom right corner marker. A simplified version of the cubical is recreated. The gaze visualizer has two GameObjects, Raycast hit

marker (black) sphere that is used for evaluating distance error, and a Raw Gaze Direction Marker (green) disk that is used for evaluating the direction error.

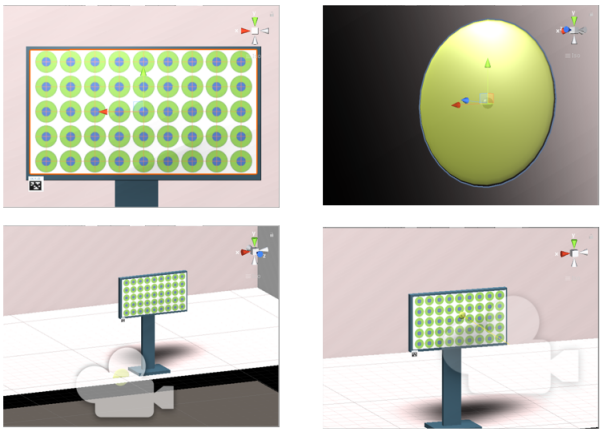

Figure 2.3. The Unity3D setup with origin marker, and the gaze visualization markers .

There are two scripts along with the virtual setup:

Data.cs

This script takes in the preprocessed, raw gaze and camera location data from the CSV file for and loads it up for visualization. The readCSV method reads the combined data from the preprocessed file and stores it in a temporary variable within the object class

```
1  void readCSV()
2          {
3              StreamReader inpstr = new StreamReader(filepath+cdata);
4              fixHeaders = inpstr.ReadLine().Split(',');
5
6              while (!inpstr.EndOfStream)
7              {
8                  string inpline = inpstr.ReadLine();
```

```
                inputList.Add(inpline);
            }
            inpstr.Close();
        }
```

Listing 2.1. readCSV method

The parselist method is used to return a row from the combined data in the correct format, taking the index as input.

```
public double[] parseList(int Index)
        {
            Debug.Log("frow " + Index);
            string[] temp = inputList[Index].Split(',');

            parsedRow[0] = double.Parse(temp[0]);

            parsedRow[1] = double.Parse(temp[1]);
            parsedRow[2] = double.Parse(temp[2]);
            parsedRow[3] = double.Parse(temp[3]);

            parsedRow[4] = double.Parse(temp[4]) ;
            parsedRow[5] = double.Parse(temp[5]) ;
            parsedRow[6] = double.Parse(temp[6]) ;

            parsedRow[7] = double.Parse(temp[7]);
            parsedRow[8] = double.Parse(temp[8]);
            parsedRow[9] = double.Parse(temp[9]);

            parsedRow[10] = double.Parse(temp[10]);
```

```
21
22              Debug.Log(" fixationinx " + parsedRow[10]);
23
24              return parsedRow;
25          }
```

Listing 2.2. parseList method

CustomViz.cs

This script retrieves the data using parseList frame by frame and updates the position of the RayCast Hit marker and Directional Hit marker. The raw data first is converted to unity coordinate and scale done by the RecieveGaze method.

```
1  void ReceiveGaze(int ind)
2          {
3              Vector3 resultG = Vector3.zero;
4              Vector3 resultC = Vector3.zero;
5              Vector3 resultR = Vector3.zero;
6
7              curGazept = Data.parseList(fixationInd);
8              lastConfidence = curGazept[0];
9              //Debug.Log("confidence for the current fixation gaze
     data" + lastConfidence);
10
11             resultG.x = (float)curGazept[1];
12             resultG.y = (float)curGazept[2];
13             resultG.z = (float)curGazept[3];
14
15             resultC.x = (float)curGazept[4];
16             resultC.y = (float)curGazept[5];
```

```
            resultC.z = (float)curGazept[6];

            resultR.x = (float)curGazept[7] * -1f;
            resultR.y = (float)curGazept[8] ;
            resultR.z = (float)curGazept[9] * -1f;

            cfixationInd = (int)curGazept[10];

            if (lastConfidence < confidenceThreshold)
            {
                return;
            }

            resultG = ReComputePoint(resultG, true);

            fixationPoint_p = resultG;
            //Debug.Log("Gaze point pupil after changing to Unity
    Coordinate System (Y-up) " + fixationPoint_p);

            //Debug.Log("Camera location befor changing Coordinate
    System" + cameraPos);

            cameraPos = ReComputePoint(resultC, false);
            //Debug.Log("Camera location after changing to Unity
    Coordinate System (Y-up)" + cameraPos);

            cameraRot = ReCompRot(resultR);

            localGazeDirection = resultG.normalized;
            gazeDistance = resultG.magnitude;
```

```
44          }
```

Listing 2.3. ReceiveGaze method

The showProjected method uses the modified data to update the location of the MainCamera first and then that of the RayCast Hit marker and Directional Hit marker. The camera position is determined with respect to the origin marker in the virtual environment. This method uses a Sphere caster from the Physics library of Unity3D which takes the location of the gaze from the raw data in local Unity3D camera coordinates, a sphere radius (0.005m by default), and the origin, a location of the camera. The location of the collision of the ray is used as the location of the RayCast Hit marker. The distance of the location of the gaze from the camera is used to determine the location of the Directional Hit marker with an error correction on the local scale.

```
1  void ShowProjected()
2          {
3              gazeDirectionMarker.localScale = origMarkerScale;
4
5              Vector3 corigin = headOrigin.position;
6              Vector3 camerawc = headOrigin.TransformPoint(cameraPos);
7
8              cameraH.transform.position = camerawc;
9              cameraH.transform.rotation = cameraRot;
10
11             //Debug.Log("Cam wc  " + camerawc);
12
13             Transform gazeOrigin = cameraH.transform;
```

```
14
15
16          Vector3 origin = gazeOrigin.position;
17          Vector3 direction = gazeOrigin.TransformDirection(
    localGazeDirection);
18
19          if (Physics.SphereCast(origin, sphereCastRadius,
    direction, out RaycastHit hit, Mathf.Infinity))
20          {
21              Debug.DrawRay(origin, direction * hit.distance,
    Color.yellow);
22
23              projectionMarker.position = hit.point;
24
25              gazeDirectionMarker.position = origin + direction *
    hit.distance;
26              gazeDirectionMarker.LookAt(origin);
27
28              if (errorAngleBasedMarkerRadius)
29              {
30                  gazeDirectionMarker.localScale =
    GetErrorAngleBasedScale(origMarkerScale, hit.distance,
    angleErrorEstimate);
31              }
32          }
33          else
34          {
35              Debug.DrawRay(origin, direction * 10, Color.white);
36          }
```

```
37          }
```

Listing 2.4. ShowProjected method

2.3 Eye-tracking

The Pupil core is an open-source eye-tracking device created by Pupil Labs. They provide three software for capturing, playing, and streaming the data collected by the device. The device consists of two small cameras oriented at each of the user's eyes, and a front camera that provides a First Person View Perspective of the user. The eye-tracker needs to be calibrated for recording the data, the calibration provides flexible control over-focusing on the pupil placement and focuses on the input image. There are a couple of inbuilt plugins that allow gaze-detection, surface mapping, head pose with respect to surfaces, and fixation detection. The application also allows us to record the streaming data. The raw data can be extracted from the recording with another application, Pupil Player, also designed to access the recorded format and to perform offline the same things as the Pupil Capture.

Pupil core specifications

World Camera:
30-120Hz, 90° FOV, 1920 X 1080 resolution.
Eye Cameras:
200Hz, 60° FOV

Recording the Experiment:

The subject is seated in front of the targets on a rolling chair and is asked to fixate over the display at different points wearing the Pupil Core tracker. The Pupil Capture software allows us to calibrate the pupil and gaze detection with inbuilt markers and we use a 5 point calibration method, ensuring that the confidence of pupil detection from each eye camera is above 0.8 for stable detection.

Note :

Ensure that the subject focuses on a thin object like the tip of a pencil and rotates their head while fixating on the object for good pupil detection and to check if the tracker is placed correctly.

It is then used to record the gaze points throughout the session which utilizes the feed from the two eye cameras to determine a gaze location within the world camera feed, converts this into a 3D point with the world camera lens center as the origin. Along with this, the world camera picks up the surface markers that help is tracking the world camera located within the environment. To ensure that the markers are visible we use an LED array light pointing at the markers as ambient light within the cabin was not enough for a stable head-pose tracking. The experiment is conducted in three different scenarios under both White LEDs and Yellow LEDs:

- With the head at a fixed distance from the targets, the subject sits still on the rolling chair in front of the monitor and records the gaze points over the targets.

- While moving the head at a fixed distance from the targets, the subject sits on the rolling chair in front of the monitor and records the gaze points over the targets.
- While moving around in the cubical, seated on the rolling chair along with the body and head movements while looking at the targets.

Each experiment is recorded on Pupil capture after calibration is run from the start position, storing the recorded data (NYP format) in a default folder. The subject fixates on random targets (on an image) within the display. Pupil Player is used for detecting fixations and head poses to save the recorded data in an accessible format. A fixation is defined as a collection of gaze points within a depression of 1.5° between 600ms and 100ms and for every detected fixation, the timestamp, the start frame, end frame, duration and a list consisting of base data, i.e timestamps of depended gaze points are recorded by the fixation detection plugin. In Head pose tracker plugin, marker cache is maintained to keep track of the detected markers in each frame using which a 3D model of the surrounding is created in marker coordinates with the vertex 0 of the origin marker (can be set manually) as the model origin. Using OpenCV the plugin determines the camera's location and orientation at each frame where enough markers are visible. The Pupil Player allows us to dump the raw data from the recording into CSV files.

The three main files that we will use are:

id	start_timestamp	duration	start_frame_index	end_frame_index	norm_pos_x	norm_pos_y	dispersion	confidence	method	gaze_point_3d_x	gaze_point_3d_y	gaze_point_3d_z	base_data
0	342814.1555	600.5965	13	16	0.3878293	0.528021	1.4576652	0.7730723	3d gaze	-33.9877396	-8.12068235	161.7328485	342814.1555325 342814.159453 342814.1634125

Figure 2.4. fixations.csv.

id: fixation ID within the recording

start_timestamp: The time stamp at which the fixation was first detected

duration: The duration of the fixation in milliseconds

start_frame_index: The start frame number in the recording

end_frame_index: The end frame number in the recording

norm_pos_x: The x coordinate of the fixation point in the pixel coordinates of the world camera recording

norm_pos_y: The y coordinate of the fixation point in the pixel coordinates of the world camera recording

dispersion: The depression angle of the fixation point

confidence: The confidence in the pupil detection at current frame

method: Algorithm used for Pupil detection

gaze_point_3d_x: The x coordinate of the fixation point in the 3D world camera coordinates

gaze_point_3d_y: The y coordinate of the fixation point in the 3D world camera coordinates

gaze_point_3d_z: The z coordinate of the fixation point in the 3D world camera coordinates

base_data: A list of timestamps of the gaze positions used to detect the fixation point

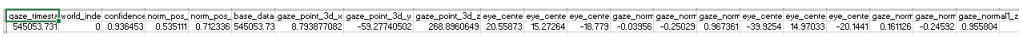

Figure 2.5. gaze_positions.csv.

gaze_timestamp: The time stamp at which the gaze point was detected

world_index: Index of the frame in the world camera recording

confidence: The confidence in the pupil detection at current frame

norm_pos_x: The x coordinate of the gaze point in the pixel coordinates of the world camera recording

norm_pos_y: The y coordinate of the gaze point in the pixel coordinates of the world camera recording

base_data: A list of timestamps of the pupil data used to detect the gaze point

gaze_point_3d_x: The x coordinate of the gaze point in the 3D world camera coordinates

gaze_point_3d_y: The y coordinate of the gaze point in the 3D world camera coordinates

gaze_point_3d_z: The z coordinate of the gaze point in the 3D world camera coordinates

eye_center0_3d_x: The x coordinate of the gaze point from left eye camera data in the 3D world camera coordinates

eye_center0_3d_y: The y coordinate of the gaze point from left eye camera data in the 3D world camera coordinates

eye_center0_3d_z: The z coordinate of the gaze point from left eye camera data in the 3D world camera coordinates

gaze_normal0_x: The x coordinate of the gaze point in the pixel coordinates of the world camera recording based on left eye camera data

gaze_normal0_y: The y coordinate of the gaze point in the pixel coordinates of the world camera recording based on left eye camera data

gaze_normal0_z: The estimation of distance of the gaze point from the world camera based on left eye camera data

eye_center1_3d_x: The x coordinate of the gaze point from right eye camera data in the 3D world camera coordinates

eye_center1_3d_y: The y coordinate of the gaze point from right eye camera

data in the 3D world camera coordinates

eye_center1_3d_z: The z coordinate of the gaze point from right eye camera data in the 3D world camera coordinates

gaze_normal1_x: The x coordinate of the gaze point in the pixel coordinates of the world camera recording based on right eye camera data

gaze_normal1_y: The x coordinate of the gaze point in the pixel coordinates of the world camera recording based on right eye camera data

gaze_normal1_z: The estimation of distance of the gaze point from the world camera based on right eye camera data

timestamp	rotation_x	rotation_y	rotation_z	translation_x	translation_y	translation_z
545053.728	0.030084746	-3.095520258	-0.070064686	-8.781165123	-6.645788193	17.82928658

Figure 2.6. head_pose_tracker_poses.csv.

timestamp: The time stamp of the world camera recording at which the camera position and orientation is detected

rotation_x: The rotation angle along x axis of the camera from marker positions in the world camera data in the 3D marker coordinates

rotation_y: The rotation angle along y axis of the camera from marker positions in the world camera data in the 3D marker coordinates

rotation_z: The rotation angle along z axis of the camera from marker positions in the world camera data in the 3D marker coordinates

translation_x: The x coordinate of the camera from origin marker positions in the world camera data in the 3D marker coordinates

translation_y: The y coordinate of the camera from origin marker positions in the

world camera data in the 3D marker coordinates

translation_z: The z coordinate of the camera from origin marker positions in the world camera data in the 3D marker coordinates

CHAPTER 3

Integration

3.1 Combining the Recorded data

Before using the raw data for visualization in the virtual environment we need to make a consolidated version of it. We use python for this purpose. First, we collect all the base data timestamp lists for each fixation point in an experiment and create a list of new fixation Point timestamps from the gaze data. For each fixation point, we retrieve the fixation index, the list of closest timestamps from gaze data, and corresponding timestamps from the head pose data, creating a dictionary for easy access. Using this dictionary, a table containing the gaze position and confidence along with the camera position and orientation at corresponding timestamp closest to the gaze timestamp is created which acts as the combined input data for the Unity3D environment (saved as CSV).

confidence	gaze_x	gaze_y	gaze_x	camera_x	camera_y	camera_z	camera_rx	camera_ry	camera_rz	fixInx
0.473908655	-0.004440446	-0.039096814	0.380413102	-9.951049805	-7.047941208	19.90202904	-0.021529021	3.063387632	0.164290488	0

Figure 3.1. combine.csv.

confidence: The confidence in the pupil detection for current gaze point of the fixation

gaze_x: The x coordinate of the gaze point from the closest time stamp in gaze data, in the 3D world camera coordinates

gaze_y: The t coordinate of the gaze point from the closest time stamp in gaze data, in the 3D world camera coordinates

gaze_x: The z coordinate of the gaze point from the closest time stamp in gaze data, in the 3D world camera coordinates

camera_x: The x coordinate of the camera from origin marker positions in the world camera data in the 3D marker coordinates

camera_y: The y coordinate of the camera from origin marker positions in the world camera data in the 3D marker coordinates

camera_z: The z coordinate of the camera from origin marker positions in the world camera data in the 3D marker coordinates

camera_rx: The rotation angle along x axis of the camera from marker positions in the world camera data in the 3D marker coordinates

camera_ry: The rotation angle along y axis of the camera from marker positions in the world camera data in the 3D marker coordinates

camera_rz: The rotation angle along z axis of the camera from marker positions in the world camera data in the 3D marker coordinates

fixInx: The fixation index of the gaze point

3.2 Recording Animation in the 3D Environment

The Unity 3D script *Data.cs* is used to read the generated combined data CSV file into the virtual environment when the Play button is pressed. For each row in the input file the *CustomViz.cs* script first retrieves the camera position and orientation, converts it to Unity coordinate system, and updates the Main camera's location. Then the corresponding gaze point location is obtained from the normalized pupil data and converted to unity coordinates, then a ray with the thickness of 0.005m is cast from the center of the camera and the location of the collision of the ray in the environment acts as the virtual gaze point location used as the location of RayCast

Hit Marker (black sphere). The magnitude of gaze point is used by the Direction marker (green disk) for visualizing the direction error. For each fixation point, we record the location of the center of the RayCast Hit Marker as a virtual gaze position and the distance of the Direction marker from the main camera, along with the respective Pupil Capture data. This is stored in a CSV file named *Output.csv*.
fixationInx: The fixation index of the gaze point

	fixationInx	pupil_f_x	pupil_f_y	pupil_f_z	unity_f_x	unity_f_y	unity_f_z	unity_d	pupil_d	confidence
0	0	-0.026921	0.011330	0.279736	-0.076077	0.032515	0.795801	0.281257	0.721013	0.803369

Figure 3.2. Output.csv.

pupil_f_x: The x coordinate of gaze point from Pupil capture data

pupil_f_y: The y coordinate of gaze point from Pupil capture data

pupil_f_z: The z coordinate of gaze point from Pupil capture data

unity_f_x: The x coordinate of gaze point from Unity3D data

unity_f_y: The y coordinate of gaze point from Unity3D data

unity_f_z: The z coordinate of gaze point from Unity3D data

pupil_d: The distance of the gaze point from Pupil capture data

unity_d: The distance of the gaze point from Unity3D data

confidence: The confidence of pupil detection for gaze point in Pupil capture data

3.3 Evaluation of 3D Environment data

There are two errors that we calculate to evaluate the performance of the experiments:

Directional error: The directional error is defined as the euclidean distance between the Pupil capture gaze point and the Unity3D gaze point along the X-Y plane. This is converted to angular representation using the absolute difference between the z coordinates of the actual and virtual gaze points.

$$dirE_i = \arcsin \frac{\sqrt{({p^x}_i - {u^x}_i)^2 + ({p^y}_i - {u^y}_i)^2}}{|{p^z}_i - {u^z}_i|} \tag{3.1}$$

Where,

${p^x}_i$ is The real-world gaze point's x coordinate

${p^y}_i$ is The real-world gaze point's y coordinate

${p^z}_i$ is The real-world gaze point's z coordinate

${u^x}_i$ is The virtual world gaze point's x coordinate

${u^y}_i$ is The virtual world gaze point's y coordinate

${u^z}_i$ is The virtual world gaze point's z coordinate

Distance error: The distance error is defined as the difference between the viewing angles of the actual and virtual gaze point. The viewing angle is defined using the distance of the gaze and the z coordinate. In case of value errors the y coordinate is used.

$$disE_i = |\arccos \frac{{p^z}_i}{{p^d}_i} - \arccos \frac{{u^z}_i}{{u^d}_i}| \tag{3.2}$$

or

$$disE_i = |\arcsin \frac{{p^y}_i}{{p^d}_i} - \arccos \frac{{u^y}_i}{{u^d}_i}| \tag{3.3}$$

Where,

${p^d}_i$ is The distance of the real world gaze point from the world camera

${u^d}_i$ is The distance of the virtual world gaze point from the virtual camera

For each experiment we then iterate over every fixation index to calculate the average fixation point accuracy and precision.

$$adirE_{f_j} = \frac{\sum_{i=f[0]}^{f[n]} dirE_i}{n} \tag{3.4}$$

$$adisE_{f_j} = \frac{\sum_{i=f[0]}^{f[n]} disE_i}{n} \tag{3.5}$$

Where,

f is the list of gaze point indices for a fixation point,

n is the number of gaze points associated with the fixation point

Result graphs:

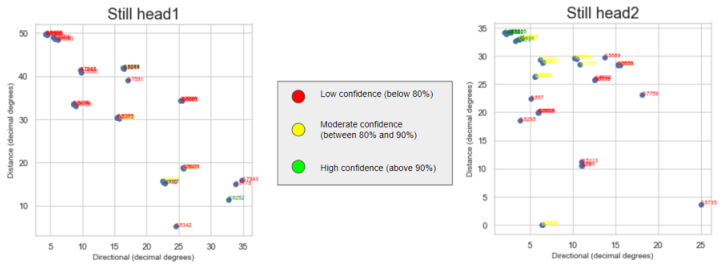

Figure 3.3. Still Head Experiments.

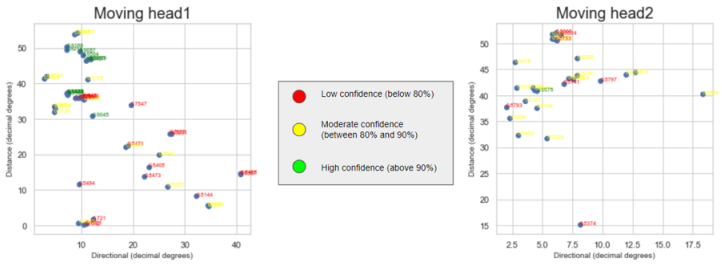

Figure 3.4. Moving head Experiments.

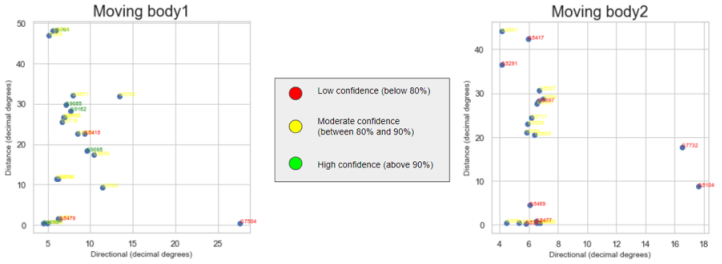

Figure 3.5. Moving body Experiments.

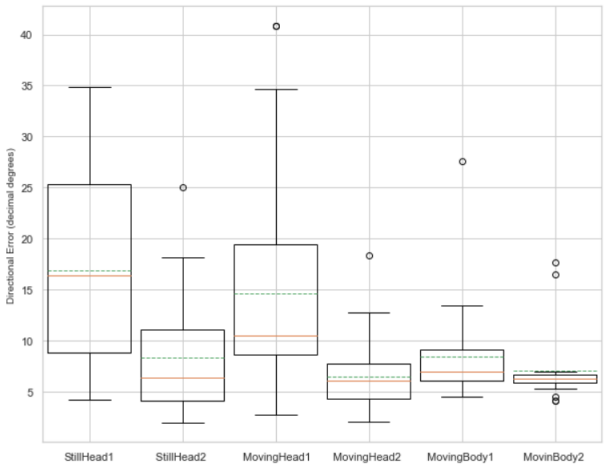

Figure 3.6. Directional Error over all Experiments.

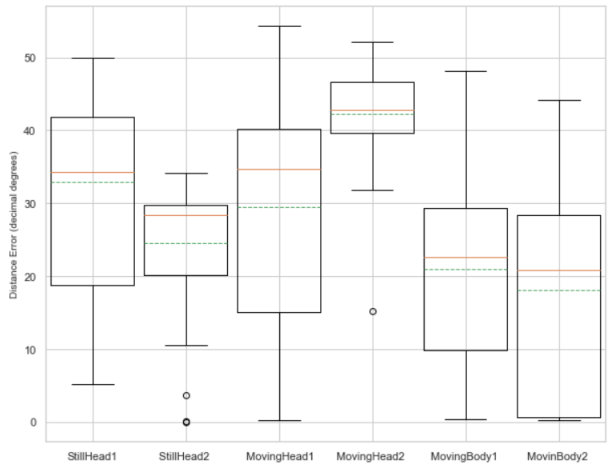

Figure 3.7. Distance Error over all Experiments.

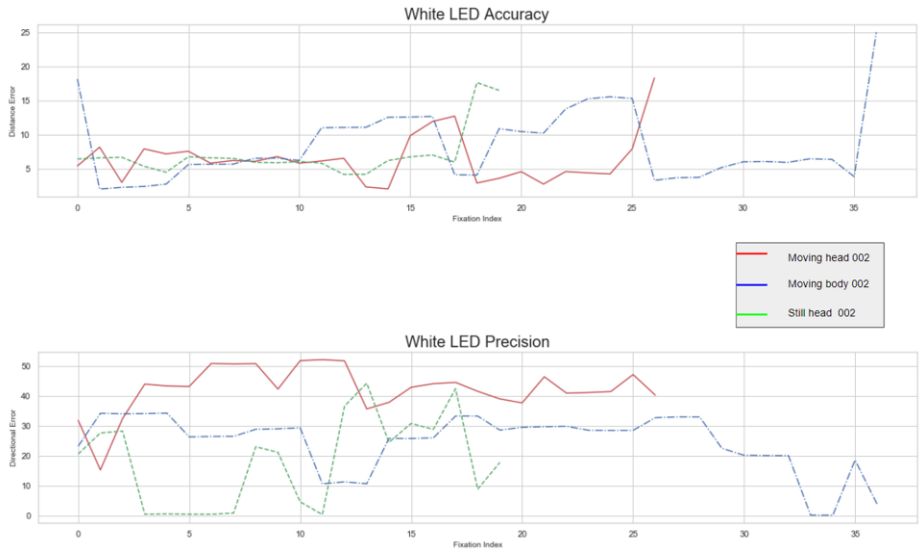

Figure 3.8. Performance under White LED lights.

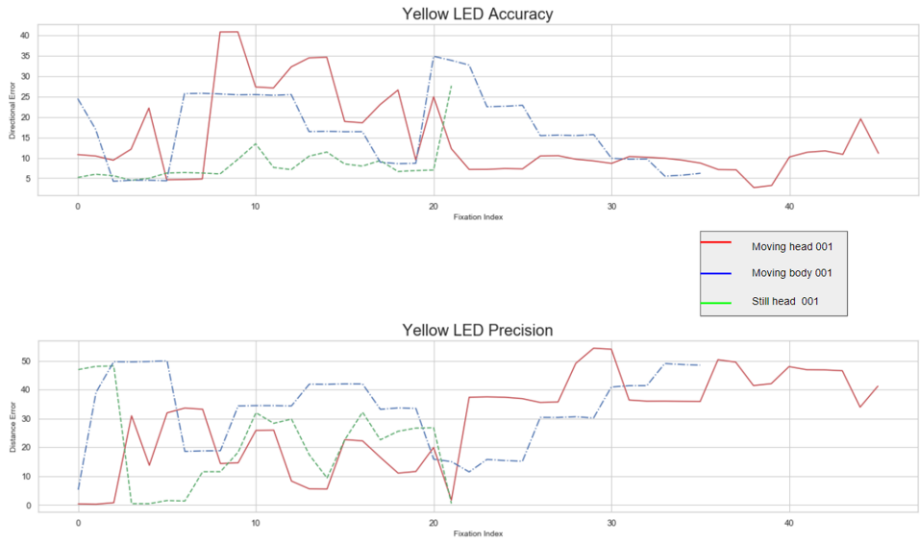

Figure 3.9. Performance under Yellow LED lights.

CHAPTER 4

Conclusion

From the following graphs we can conclude that Pupil core eye-tracker can be reliably used for recording fixation data which can be used in a simulation. The still head experiment results according to fig 3.3, the white LEDs only help improve slightly the accuracy and very small change in precision compared to yellow LEDs. By observing to fig 3.4 we see that for moving head experiments the white LEDs help improve significantly the accuracy and almost has no change in precision compared to yellow LEDs. Finally fig 3.5 shows that for moving body the white LEDs help improve slightly the accuracy and very small change in precision compared to yellow LEDs. From fig 3.6 , and 3.8 we can conclude that in all scenarios lit with white LEDs help improve the accuracy of simulation and fig 3.7 , and 3.9 show that even precision is slightly better than under yellow LEDs.

Experiment Name	Total rows	Average Directional error (decimal degrees)	Average Distance error (decimal degrees)
DMH2	5440	6.4809232744490215	42.647877350160684
DMH1	9150	14.352453439040488	30.86266116401923
DMB2	3900	6.747919494364545	19.054756501034568
DMB1	3989	8.228726117778766	21.75002445888956
DSH2	8664	8.220661393018473	24.56927407224852
DSH1	8075	16.284118516394397	33.52183441037246

CHAPTER 5

Future work

The next step would be combining the Pupil core eye-tracker data with Optitrack motion capture data and to evaluate the overall performance. The april36tags can be used to gain ground truth data for head pose tracking, comparing the OpenCV data with the Optitrack one. This will help us further establish a concrete way of collecting data for the develop of the Simulated Environment for Developmental Robotics (SEDRo), a virtual environment meant to help cut cost in AI research by providing a Real-world like immersion and input within a simulated environment which can be used for training AI to learn intelligent, human-like behaviour.

References

[1] William Agnew and Pedro Domingos. *Self-Supervised Object-Level Deep Reinforcement Learning*. Mar. 2020.

[2] Richard N. Aslin. "Infant Eyes: A Window on Cognitive Development". In: *The Official Journal of the International Society on Infant Studies* 17.1 (2010), 126–140–126–140.

[3] Mohammad Gheshlaghi Azar et al. *World Discovery Models*. 2019. arXiv: 1902.07685 [cs.AI].

[4] Angelo Cangelosi and Matthew Schlesinger. *Developmental Robotics: From Babies to Robots*. The MIT Press, 2014. ISBN: 0262028018.

[5] Angelo Cangelosi and Matthew Schlesinger. "From Babies to Robots: The Contribution of Developmental Robotics to Developmental Psychology". In: *Child Development Perspectives* 12.3 (2018), pp. 183–188. DOI: 10.1111/cdep.12282. eprint: https://srcd.onlinelibrary.wiley.com/doi/pdf/10.1111/cdep.12282. URL: https://srcd.onlinelibrary.wiley.com/doi/abs/10.1111/cdep.12282.

[6] Devendra Singh Chaplot et al. "Gated-Attention Architectures for Task-Oriented Language Grounding". In: *arXiv preprint arXiv:1706.07230* (2017).

[7] Devendra Singh Chaplot et al. "Gated-Attention Architectures for Task-Oriented Language Grounding". In: *arXiv preprint arXiv:1706.07230* (2017).

[8] Kirsten A. Dalrymple et al. "An Examination of Recording Accuracy and Precision From Eye Tracking Data From Toddlerhood to Adulthood". In: *Frontiers in Psychology* 9 (2018). DOI: 10.3389/fpsyg.2018.00803.

[9] Anna Maria Feit et al. "Toward Everyday Gaze Input: Accuracy and Precision of Eye Tracking and Implications for Design". In: *Proceedings of the 2017 CHI Conference on Human Factors in Computing Systems* (2017).

[10] Ana Alexandra Caldas Osório1 Gisane Novaes Balam. "Use of eye tracking technology in infancy research". In: *Revista Psicologia: Teoria e Prática* 20.1 (2020), pp. 179–188. ISSN: 1980-6906. DOI: 10.5935/1980-6906/psicologia.v20n1p179-188.

[11] Moritz Philipp Kassner and William Rhoades Patera. "PUPIL: Constructing the Space of Visual Attention". MA thesis. Massachusetts Institute of Technology, 2012. URL: http://hdl.handle.net/1721.1/72626.

[12] Moritz Kassner, William Patera, and Andreas Bulling. "Pupil: An Open Source Platform for Pervasive Eye Tracking and Mobile Gaze-Based Interaction". In: *Proceedings of the 2014 ACM International Joint Conference on Pervasive and Ubiquitous Computing: Adjunct Publication*. UbiComp '14 Adjunct. Seattle, Washington: Association for Computing Machinery, 2014, pp. 1151–1160. ISBN: 9781450330473. DOI: 10.1145/2638728.2641695. URL: https://doi.org/10.1145/2638728.2641695.

[13] Cynthia Matuszek. "Grounded Language Learning: Where Robotics and NLP Meet". In: *Proceedings of the Twenty-Seventh International Joint Conference on Artificial Intelligence, IJCAI-18*. International Joint Conferences on Artificial Intelligence Organization, July 2018, pp. 5687–5691. DOI: 10.24963/ijcai.2018/810. URL: https://doi.org/10.24963/ijcai.2018/810.

[14] Deepak Pathak et al. "Curiosity-driven Exploration by Self-supervised Prediction". In: *ICML*. 2017.

[15] Marek Wydmuch, Michal Kempka, and Wojciech Jaśkowski. "ViZDoom Competitions: Playing Doom from Pixels". In: *IEEE Transactions on Games* PP (Oct. 2018), pp. 1–1. DOI: 10.1109/TG.2018.2877047.

[16] Chen Yu and Linda B. Smith. "Joint Attention without Gaze Following: Human Infants and Their Parents Coordinate Visual Attention to Objects through Eye-Hand Coordination". In: *PLoS ONE* 8 (2013).

[17] Chen Yu, Sumarga H. Suanda, and Linda B. Smith. "Infant sustained attention but not joint attention to objects at 9 months predicts vocabulary at 12 and 15 months". In: *Developmental Science* 22.1 (), e12735. DOI: 10.1111/desc.12735. eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1111/desc.12735. URL: https://onlinelibrary.wiley.com/doi/abs/10.1111/desc.12735.