Project Documentation

Group ThaljtDj team members: Lucas Essmann, Jasmin Walter, Dirk Gütlin, Tom Hatton, Hendrik Timm, Anne Lang, Josefine Zerbe, Tilman Kalthoff

Content

What you need to know as an experimenter

Overview

The experiment can be completed using several playmodes. Matching the original experiment, different experimental conditions can be selected. The experimental setup will depend on the selected playmode (VR or fallback mode). An extensive UI menu allows for intuitive selection of the necessary experimental settings. In case of the VR playmode (HMD Use) the settings are only visible for the experimenter.

UI Settings

In the beginning, press ‘Start as Server’ on one machine and ‘Start as Client’ on the other one (upper right corner). Afterwards, it is important to select the required settings (you can access the menu via the cogwheel button on the upper left).
Firstly, click on ‘Settings’. There you can enable the usage of a VR headset by activating ‘HMD Use’ (This is very important because it en-/disables the UI and instructions for the HMD or the 2D Fallback Mode). There you can also en-/disable data capturing and the usage of eye tracking. After setting up everything, again click on ‘Settings’ to return to the menu.
Secondly, click on ‘Sub Settings’. Here you can enter a subject ID (set with Enter) and select an experiment condition. After setting up everything, again click on ‘Sub Settings’ to return to the menu. Hypothetically, you could also click on ‘Calibration’ and ‘Validation’, though they are not yet assigned (see Eye Tracking with Tobii Eye Tracker (Vive Pro Eye) for more details). The ‘pause’ button stops unity time (set to 0). But be very careful with this button, since data capturing is also depending on the Unity time. The trials are time independent, so they could be carried on with stopped time, however, in that case they will not be logged.
The ‘End Experiment’ button ends the experiment and closes the exe!

When you have finished the setup and preparations, you can hit ‘Start Experiment’ to spawn the subject in the experiment room. Remember to make sure that your subjects followed the instructions and turned towards the door with the “ready” sign, otherwise they might not see the wall of fame directly when being teleported to the experiment room.

Note: All these settings, including the Start button, are machine dependent and thus have to be done on both machines.

Movement and Controllers

The experiment can be completed with or without a VR headset. Movement and controllers depend on the selected playmode.

- Playmode: **HMD Use**
    - **Experimental setup:** the participant should be seated on a turntable chair, so that rotation is possible, but the position of the participant is fixed.
    - **Settings:** If a HMD is connected to the computer, the movement and controllers for the VR mode will be automatically selected. Though the correct

playmode “with HMD” should be selected in the settings to activate the correct instructions.
    - **Movement**: the participant will be able to move by moving the head freely and rotate the full body by rotating on the rotatable chair.
    - **Controller:** one controller should be placed in each hand. By pressing the buttons on the controller, it can be verified that the left and right controller are grabbed with the matching hand.
- Playmode: **2D Fallback Mode = without HMD**
    - **Experimental setup:** the participant should be seated in front of a 2D display. A keyboard and mouse should be available.
    - **Settings:** If no HMD can be detected when starting the program, the 2D fallback mode will be automatically activated, including the correct settings.
    - **Movement:** the participants will be able to rotate using the arrow keys (slow rotation) or by clicking the right mouse button and moving the mouse (fast rotation). The location is fixed, so it matches the VR experimental setup with the fixed turnable chair.
    - **Controller:** the participant will be able to respond by clicking “y” or “n” on the keyboard - see instructions for further details.

Gaze Sphere and Shared Gaze Conditions

In the conditions that include shared gaze (SG, SVG), a gaze sphere representing the estimated location of the partners gaze will be visible and displayed in blue colour. The gaze sphere is scaled depending on the distance to the player/participant location, so it appears to stay the same visual size. The calculation method of the estimated gaze depends on the selected playmode.

- Playmode: **HMD Use & eye tracking deactivated**
    - The gaze sphere is estimated according to the nose vector of the VR headset
- Playmode: **HMD Use & eye tracking activated** (not yet implemented)
    - The gaze sphere is estimated according to the eye tracking data of Tobii
    - **Note:** calibration and validation should be completed before starting the experiment
- Playmode: **2D fallback**
    - The gaze sphere is estimated according to the position of the mouse cursor on the computer screen

Note: the data of the gaze spheres of both participants will be saved independent of whether they are visible or not during the experiment. So selecting the correct playmode depending on the available hardware is crucial for data collection.

Eye Tracking with Tobii Eye Tracker (Vive Pro Eye)

The experiment is designed so that eye tracking with an HMD integrated eye tracker is possible. Before starting the experiment, participants are located in a preparation room, so that all necessary settings, including calibration and validation of the eye tracker can be completed before starting the experiment.
Due to the closed VR lab, the implementation of eye tracking with the Tobii eye tracker could not be completed, nevertheless the necessary interface is implemented (menu buttons, gaze

sphere case selection, scene hierarchy references). For more details see section What you need to know as a programmer when modifying or adding to the project.

Trials, Spawning of Objects and Randomization

In the experiment this project is based on, the participants encounter various sets of stimuli. In target-absent trials, those stimuli only contain the Q-shaped distractor objects, whereas in target-present trials one O-shaped target object is included in the set of stimuli. While the original experiment evenly distributed target-absent and target-present trials, our implementation randomly draws the `targetPresent` variable with a probability of 0.5. This probability can be set with the variable `stimulusPresenceRate`. The original experiment was performed with 192 trials. For easier development we set the value of `numberOfTrials` to 20. In case you want to change this, make sure to have it changed in code and the Unity inspector.

In general, our implementation covers various variables which are chosen randomly such as:
- if a trial contains a target object (`targetPresent`, mentioned above)
- the number of stimuli present (`stimuliSize`, either 21 or 35)
- the rotation of each distractor object (`distractorDirection` chosen from `_distractorDirections` with the degrees of 0, 90, 180 and 270 degrees)
- the position of the target object in target-present trials (`_targetIndex`)

To ensure that both players receive the same values for each random variable, all our random variables get picked by a `Random _rnd` Object with a set random seed, which is equal for both participants. The random seed is also generated pseudo-randomly with a different `Random()` object and is sent over the network to the client machine.

Handling of User Responses and Feedback

According to the underlying paper, a trial is terminated as soon as one player gives a response to whether a stimulus set contains an O-shaped target object, no matter whether the answer is correct or incorrect.
Thus, we have to check in each `Update()` loop on each machine if either the local player or the remote player has answered. We, therefore, created both variables `localResponseGiven` and `remoteResponseGiven` in the `ExperimentManager.cs`. The `remoteResponseGiven` variable on each machine gets continuously synchronised over the network with the value of `localResponseGiven` from the other machine.

Thus, as soon as one player gives a response, the possibility of responding gets deactivated for the current trial, the reaction time ('Time.time - stimulusOnsetTime') of the responding player gets logged. This is accomplished in the function `HandleResponse()`. Also, both participants receive visual feedback for their accuracy with a text displaying "Correct!"/"Incorrect" to their response and in target-present trials a visual clue (change of material colour) for the position of the target object, which is handled in the function `GiveTargetFeedback()`.

Saving

Since networking a lot of data negatively affects the performance, we do not network any additional data for saving purposes. However, we can use all the data that will be networked anyway, and save it, for additional insights.
To enable saving the data, you should enable “Data Capture” in the Settings tab. You should also have a Subject ID defined in Sub Setting, in order to save data to unique paths. When two subjects go through the experiment, a saving folder will be created on both computers, named after the respective subject number. The saving folder can be found in the application’s “persistentDataPath/DefaultCompany/Project” (so for Windows, this directory should somehow look like:
“C:\Users\`$USER`\AppData\LocalLow\DefaultCompany\Project”). Each subject folder will be named like “subj_`$SUBJECT_ID`”. Within the subject folder, a JSON File will be stored after the end of every trial. This folder will contain the following information:

- `subID`: The unique ID of the current subject.
- `condition`: The experimental condition currently running.
- `currentTrial`: An index for the currently recorded trial.
- `targetPresent`: A bool that informs whether the target object was actually present in this trial.
- `stimuliSize`: The number of stimuli used in the current trial.
- `targetObjectPos`: The location of the target object if a target object is present. If there is no target object within this trial, this field will be {x=0, y=0, z=0}.
- `answerPresent`: True if the participants answered that the target is present, else False.
- `lastReactionTime`: The time that passed between the start of the trial and the time where a local answer was given. If only the remote subject answered the trial, the local `lastReactionTime` will be 0.
- `index`: A list of indices which enumerate all the samples logged during the trial.
- `remoteGazePos`: The {x, y, z} coordinates of the opposite subject’s camera focus (i.e. the point where the remote nose vector Raycast hits the closest object).
- `localGazePos`: The {x, y, z} coordinates of the local subject’s camera focus (i.e. the point where the local nose vector Raycast hits the closest object).
- `runTime`: The Unity time measured in seconds since the start of the experiment.
- `sysTime`: The computer’s system time at the respective sample.

Example data and an example script to analyze them are given in the repository under: DyadicInteractions/ExampleData

What you need to know as a programmer when modifying or adding to the project

In General

Most of the project functions are managed via different manager scripts. In the scene hierarchy you can find a game object “Experiment Manager”. To the experiment manager other game objects with the most important project functions are attached. The scripts are then attached to the matching game objects. Alternatively, all our scripts can be found in the Scripts folder of the project, note however, that some scripts were integrated from third parties and are still located in their original third party folders.

Settings

Most of the settings are managed via the “OverlayMenuUI” script attached to the UIManager game object.

Playmodes

The steam VR prefab we used for our player already automatically searches for an attached VR headset. If no HMD can be recognized, it automatically goes into the 2D fallback mode and activates the “NoSteamVRFallbackObjects” of the PlayerSteam game object. Here player height and movement control can be modified to match the VR conditions. The player movement control in the 2D fallback mode can be modified in the “FallbackCameraController” script attached to the FallbackObjects.
Note, that the instructions are not activated automatically but need to be set in the UI settings.

Gaze Spheres and Eye Tracking

The location of the local gaze spheres is calculated depending on the playmodes. The remote gaze sphere is only moved based on the data coming out of the networking.

2D Fallback Mode
In the 2D fallback mode, the local gaze sphere location is calculated in “FallbackCameraController” script attached to the “FallbackObjects” of the PlayerSTEAM game object.

VR mode (HMD Use)
In the VR mode (HMD Use**)** it depends whether the eye tracking is activated in the settings. If eye tracking is deactivated in the settings (which is the default setting at the moment), the local gaze sphere location is calculated in the “VRCameraTracking” script attached to the VRCamera game object of the SteamVRObjects of the PlayerSTEAM game object. Note that the bool “withEyeTracking” in the script must be set to false for the position calculation to be active. The bool “withEyeTracking” is set in the “OverlayMenuUI” script over the settings buttons.

Eye Tracking with Tobii

At the moment the eye tracking with Tobii is not yet implemented. However, the project has already implemented most of the necessary interface for the eye tracking function. The UI has the respective buttons to activate the eye tracking function as well as buttons to start the calibration and validation process (OverlayMenuUI script). The experiment manager game object also has an eye tracking manager attached, so that all eye tracking functions can be added here. Note, that with our programming logic, the eye tracking data itself will not be networked. The gaze sphere data will be networked independently whether eye tracking is activated or not. Only how the gaze sphere location is calculated depends on the game mode and eye tracking settings. In case the eye tracking function is activated, instead of using a raycast based on the nose vector or mouse cursor, the estimated gaze by Tobii can be applied to the raycast and therefore used to calculate the local gaze sphere location. Note, that the calibration, validation and saving will be done on both computers independently. Consequently, our implementation would also allow for one person to use an HMD and eye tracking, while the other person uses a different technical setup and game mode.

Spawning and Experimental procedure

The experimental procedure is implemented to a great extent in the `SpawningManager.cs`. In general, this file handles the spawning of objects, the randomization of trials, the handling of user responses and the creation of user feedback.

Networking

The project uses lightnet for the networking. Here mainly three scripts are relevant in case of modification. The basic idea is, that the networking manager contains the methods that broadcast the data into the network. To broadcast, the data needs to be serialized which is specified in the Serializable script. The experiment manager then handles the received data. If modifications are implemented, it is important that all 3 scripts match each other in the respective methods and variables.

1. The Experiment Manager script
    a. In the experiment manager script, the methods that receive the network data need to be adjusted according to the needs of the networking. This includes:
        i. `ReceivedUserStateUpdate`
        ii. `ReceivedRandomStateUpdate`
        iii. `ReceivedResponseStateUpdate`
        iv. `SetExperimentStatus`

2. The Networking Manager script
    a. In the networking manager script the methods that send out the data are implemented. This includes:
        i. `BroadCastExperimentStatusUpdate`
        ii. `BroadcastExperimentState`
        iii. `BroadCastRandomState`
        iv. `BroadCastResponseState`

3. The serializable script (can be found in the lightnet folder in the scripts folder of the project). The network data needs to match the variables that are broadcasted in the broadcasting methods (network manager) and received in the receiving methods (experiment manager). Here the Serialize and Deserialize methods need to be adjusted according to what needs to be send/received - see comments in the script for more details.

Saving

Most of the saving is handled in the “SavingManager” script. Note that the networking and spawning manager are closely linked to the saving procedure. The saving process is activated in the OverlayMenuUI script.

Licenses:

Lean GUI Extension Asset Licence

https://assetstore.unity.com/packages/tools/gui/lean-gui-72138