Tuniklab VR Experiment Suite

Welcome to the Tunik Lab README! Enclosed are a series of brief descriptions of the various components of the lab and the VR Experiment Suite you will be working with.

The main resource for your time here in the Tunik Lab will be the Unity program, currently called the Tuniklab VR Experiment Launcher (VEL for short). It combines information from PPT cameras, a Cyberglove, Intersense trackers, and an Inertia cube to generate a hand that should look and behave similarly to your own. We use this hand to run trials, constructed in the Tuniklab VR Experiment Creator (VEC for short), which use all of the above information to change the virtual reality scene. Pretty cool, right?

A quick note before you delve into the various resources below: Things with heading that mention development are meant to be directed only towards readers who have a programming background and are looking to extend the functionality of the Tuniklab VR Experiment Suite

---

The Tuniklab VR Experiment Creator (VEC)

Config Creation:

The Tuniklab VR Experiment Creator is the tool that you, or other researchers will be using to build the trials that participants will be undertaking. You can use the VEC to set how many fine grained details about the trials, as well as simple things such as how many objects you want in a scene, where you want them, and how you want them to change over the course of the experiment or trials. This is also where you set basic identification information about the experiment, such as the name of the participant and where you want the captured data to be stored when the experiment is completed.

The VEC is split up into 3 tabs:

Peripheral Tab:

This tab allows you to specify device and general configuration settings, such as what devices and triggers will be enabled for any part of the experiment, as well as if you'd like to use specific experiment-wide functionality.

Environment Tab:

This tab allows you to specify parameters about the virtual environment across all trials. It also allows you to define rough details about the unique objects that you will use in the experiment.

Experiment Tab:

This tab allows you to specify the nitty-gritty details of each trial, and all objects, triggers, and interactions within them. See the **Glossary of Experiment Table Fields** below for more information on this tab.

The best way to get familiar with these tabs is to explore the VEC yourself! Most fields and options should have accurate labels.

Following completion of all tabs, you should make sure to save the config in a memorable location so that it can be used in the Tuniklab VR Experiment Launcher to run virtual reality experiments! Currently, the config file is saved as a json file, and can be values can be quick edited from any text editor (as long as you are careful). Alternatively, you can load the file using the VEC and make changes there (making sure to save after all changes).

Glossary of Experiment Table Fields

Break Trial? (bool): Specifies whether or not this trial should be set as a break (no data recorded and limited trial functionality)

Duration (seconds): The time that this trial will run

Condition Name (string): The name of this trial, can be used to specify what condition is being tested as the participant shouldn't be able to see this name

Cue

Text (string): The text that will appear as the cue

Duration (string): The duration that the cue text will appear for

Delay (seconds): The delay from the beginning of the trial before the Cue Text will appear

Finger Angle Gain

Gain (float): [0-inf) The proportion of finger angle change that translates to the VR model. 1 is normal movement.

PPT Gain

Gain (float): [0-inf) The proportion of hand position change that translates to the VR model. 1 is normal movement.

Orientation Gain

Gain (float): [0-inf) The proportion of hand orientation change that translates to the VR model. 1 is normal movement.

Object Grab

Color: The color that an object will change to when grabbed by the CyberGlove

Position Trigger

X (centimeters): The displacement in the x direction that must be reached (in addition to pos y and pos z) to fire the trigger

Y (centimeters): The displacement in the y direction that must be reached (in addition to pos x and pos z) to fire the trigger

Z (centimeters): The displacement in the z direction that must be reached (in addition to pos x and pos y) to fire the trigger

Velocity Trigger

X (centimeters/second): The velocity in the x direction that must be reached (in addition to vel y and vel z) to fire the trigger

Y (centimeters/second): The velocity in the y direction that must be reached (in addition to vel x and vel z) to fire the trigger

Z (centimeters/second): The velocity in the z direction that must be reached (in addition to vel x and vel y) to fire the trigger

Aperture Trigger

finger 1: The first finger to use for aperture triggering

finger 2: The second finger to use for aperture triggering

distance (centimeters): The distance that must be reached between finger 1 and finger 2 to fire the trigger

Finger Angle Trigger

finger: The finger whose joint angle will be measured

joint: The joint whose angle will be measured

angle (degrees): The angle that must be reached to fire the trigger. 0 angles are a flat hand

Orientation Trigger

pitch (degrees): The pitch that must be reached (in addition to roll and yaw) with the orientation sensor on the hand to fire the trigger

roll (degrees): The roll that must be reached (in addition to pitch and yaw) with the orientation sensor on the hand to fire the trigger

yaw (degrees): The yaw that must be reached (in addition to pitch and roll) with the orientation sensor on the hand to fire the trigger

DAQ

React to (trigger): The trigger which should trigger a pulse to be sent to the DAQ. **Note: Can be left blank for no trigger on this trial**

Pulse Delay (seconds): The time after the trigger has occurred before the DAQ should send a pulse

Pulse Channel: The channel of the DAQ that the pulse should be sent to

Object General Parameters

Shape: The shape of this object in the scene

Dimensions: The size of the shape. **Format of this field is dependent on shape:**

- Rectangular Prism: l,w,h
- Sphere: r
- Cylinder: r,h

Color: The color of this object in the scene

Position X (centimeters): The position of the center of the object from the origin in the X direction

Position Y (centimeters): The position of the center of the object from the origin in the Y direction

Position Z (centimeters): The position of the center of the object from the origin in the Z direction

Euler Rotation X (degrees): The rotation of the object in the X direction

Euler Rotation Y (degrees): The rotation of the object in the Y direction

Euler Rotation Z (degrees): The rotation of the object in the Z direction

Object Physics Properties

Mass (float): (0-inf) The mass of the object. Smaller mass equates to a lighter object, and a larger mass equates to a heavier object.

Drag (float): (0-inf) The linear drag coefficient. Higher drag means that the object's movement will 'slow down' faster.

Angular Drag (float): (0-inf) The angular drag coefficient. Higher drag means that the object's rotation will slow down faster.

Static Friction (float): [0-1] The friction coefficient to use when at rest. A value of 0 is like ice, while a value of 1 will make it hard to get the object moving.

Dynamic Friction (float): [0-1] The friction coefficient to use when already moving. A value of 0 is like ice, while a value of 1 will make the object come to rest very quickly without a strong force behind it.

Bounciness (float): [0-1] Determines how bouncy the object is. A value of 0 will not bounce, while a value of 1 will bounce without any loss of energy.

Object Perturbation

Perturbed By (trigger): The trigger by which this object will be perturbed **Note: Can be left blank for no trigger on this trial**

Delay (seconds): The amount of time between when the trigger is activated and when the perturbation will occur

Note: Subsequent object post-perturbation parameters mirror those spec'd out above.

VEC Development:

1. In order to develop you need the following installed:
   - Python 2.7 (https://www.python.org/downloads/release/python-2712/)
   - pyQt4 (https://sourceforge.net/projects/pyqt/files/PyQt4/PyQt-4.11.4/)
   - A python editor (preferably PyCharm Community Edition: https://www.jetbrains.com/pycharm/download/#section=windows)
   - jsonpickle, which can be downloaded from the package manager built into pycharm
2. After installing the above, you should be able to open the Rev1 folder in the repo as a PyCharm project. You may need to configure the python interpreter, but if Python is installed correctly there should be only one option in the interpreter dropdown
3. In order to test the GUI, you should be able to run `Main.py` in PyCharm

Building an executable:

1. In addition to everything in the development section, you will also need the following installed:
   - py2exe (https://sourceforge.net/projects/py2exe/files/py2exe/0.6.9/)
2. If you need to edit build parameters, you will need to edit setup.py
3. You should create a run configuration for building an executable:
   - The script being run should be setup.py
   - The "Script Parameters" field should contain the following: `py2exe`
4. When you want to create a new version executable, you should update the version number in setup.py, as well as move a copy of the contents of the Executable\CurrentBuild folder into a subfolder of the Executable\Old folder
5. Now you should be able to run this configuration any time you want to create an executable. With the current setup configuration, all files that you will need to run the executable (including the executable itself) should show up under the Executable\CurrentBuild folder.

Note: The current version of py2exe does not allow bundling all dependencies inside of the executable, so you will need all of the contents of the Executable\CurrentBuild folder to properly run the executable

---

Tuniklab VR Experiment Launcher (VEL)

You will be using a unique program called the Tuniklab VR Experiment Launcher in order to run Virtual Reality experiments.

This program is built using a game engine named "Unity", and it holds all of the models, scripts, and information you need to actually make things run in virtual reality.

When starting up the program, you will be required to select a config file to load. This should be a json file created in the Tuniklab VR Experiment Creator, detailed above.

VEL Calibration

Each time the VEL is run, you must calibrate certain parts of the environment to provide a realistic experience and accurate data.

Generally, you as the experimenter should be following the steps left to right on the screen.

1. Recenter Screen: Have the Patient face straight forward with the headset on, then press this button. This calibrates the orientation of the HMD.
2. Calibrate Flat: Have the Patient place their hand flat on the table with their fingers together, then press this button. This calibrates the 'zero' position of their finger joints.
3. Calibrate Fist: Have the Patient form a tight fist (disregarding thumb), then press this button. This calibrates the '90 degree' position of their finger joints.
4. Calibrate Thumb Roll: Have the Patient stretch their hand as far as possible across their palm, then press this button. This calibrates the '90 degree' position of one of the thumb joints.
5. Calibrate Thumb Bent: Have the Patient bend their thumb inwards as much as possible, then press this button. This calibrates the '90 degree' position of other thumb joints.
6. Calibrate Splayed: Have the Patient place their hand on the table with their fingers spread apart, then press this button. This calibrates the max abduction of the fingers.

After performing all of these calibration steps, a 'START' button should appear in the lower right of the experimenter UI. Pressing this starts the experiment!

VEL Triggers

One of the fundamental functions of the VEL the ability to change the environment and objects based on events that occur. We call these 'triggers'. They should be relatively straightforward to understand once you have gotten some experience using the VEC and VEL. There are, however, some caveats with some of the triggers.

- Velocity Trigger: *Not implemented*
- Aperture Trigger: *Not currently accurate enough to be useful*
- Orientation Trigger: Not sensitive to rotation direction (ie. right or left roll)
- Touch Trigger: *Not implemented*
- Force Trigger: *Not implemented*

VEL Development

Unity Structure Overview

Canvas- World Space

The World Space canvas describes elements of the UI which are meant to exist in the world. It should only hold things like Text or Images that exist inside the Virtual environment.

Canvas - Experimenter Only

The Experimenter Only canvas describes elements of the UI which do not exist in the world and should only be seen by the person running the experiment on the computer, and not visual inside of the headset. It holds any buttons that the experimenter can press, but the functionality for those buttons exists in relevant areas (for example, the “Calibrate Fist/Open” buttons have their functions coded in “Cyber Glove Hands Controller” (Script).

Game Main Control

This empty object holds most of the functionality tied to running the trials. It has the script which parses through the information in the GUI (“Store*GUI*Info”), the script which runs the experiments (“Unity Experiment”), the script which saves an Excel file to the desired location (“Excel_XML”), and the script responsible for triggering events to change the virtual landscape (“Triggers”).

Lab Room

This object holds all of the models for the Lab Room environment. In the case where multiple environments are in the project, you would likely want to store all of the objects that compose the room itself under an empty game object. You also may want to consider a roof, if you’re into that sort of thing.

Player

This holds all the information regarding the player, including the information on the head position and integrating the information from the PPT cameras. The script that gives control of the position of the head to the PPT cameras is “HMD Override.”

Cyberglove Input

This object holds the script that converts the information from the Cyberglove application to Unity objects and information (“Cyber Glove Input”). For the Cyberglove to be properly used, the Device Configuration Utility (or DCU) needs to be running and the device needs to be connected.

Intersense Input

This object holds the script that converts Intersense information to Unity objects and information (“I Sense Sample”). No additional programs need to be running for the Intersense to work.

PPT Trackers

This object holds the main scripts for the PPT cameras.

Cyberglove Hands

This is the Game Object that holds all of the scripts and the game objects associated with the hand object. The main script controlling how the hand responds to the inputs from the Cyberglove. There are individual sub-

objects for each of the hand, fingers, the palm, and special objects on the tip of each finger for aperture calculations.

Virtual Reality Devices

Cyberglove

The Cyberglove is a device that is meant to measure all variable joints of the hand to be used by a computer. It uses voltage strips on the back of the glove to calculate the angle of each joint on each finger. To use the glove, you need to turn on the Device Configuration Utility.

The Device Configuration Utility is a tool provided by CyberGlove Systems to both calibrate the Cyberglove, as well as send information to our virtual reality environment. To start using it, make sure the Cyberglove is plugged into the computer and that the battery connected to it has charge. Then, right click on the hand you want working and hit (Re-)connect. It may take a few tries, and you may have to open and close the application.

There is a default configuration file personally adjusted by Tunik Lab employees that is currently working that is named defaultConfig.cal in the _TEST2 project folder. This should have the hand looking and behaving visually well in the DCU. In the case that you need to recalibrate the hand, you can do so by going to the Device menu and then “Advanced Calibration.” There, you can individually change the gains and offsets for each joint to make the hand look and behave as realistically as possible. **Important: These calibration values should only be changed if you are experienced with the Cyberglove, and should not need to be changed often/in most cases**

Using the Cyberglove is relatively simple. The Cyberglove unit comes with an attached pack, on which there are two slots- a battery slot, and a mini-USB slot. Connect a battery to the battery slot (batteries are available charging near the Brainsight computer. Make sure to replace the batteries after taking one so there is always one on the charger) and connect the mini-USB slot to the computer. After that, open up the DCU and go through the steps above.

Development

Cyberglove "Raw" values are pulled through the CyberGloveUnityPlugin, through the get[Left or Right]HandRawData(). The following is the list of indexes:

Index	Joint
0	Thumb Roll Sensor
1	Thumb Inner Joint Sensor
2	Thumb Outer Joint Sensor
3	Thumb-Index Abduction Sensor
4	Index Finger Inner Joint Sensor

5	Index Finger Middle Joint Sensor
6	Index Finger Outer Joint Sensor
7	For some reason, always returns 0
8	Middle Finger Inner Joint Sensor
9	Middle Finger Middle Joint Sensor
10	Middle Finger Outer Joint Sensor
11	Index-Middle Abduction Sensor
12	Ring Finger Inner Joint Sensor
13	Ring Finger Middle Joint Sensor
14	Ring Finger Outer Joint Sensor
15	Middle-Ring Abduction Sensor
16	Pinky Finger Inner Joint Sensor
17	Pinky Finger Middle Joint Sensor
18	Pinky Finger Outer Joint Sensor
19	Ring-Pinky Abduction Sensor
20	Palm Arch Sensor
21	Wrist Flexion Sensor
22	Wrist Abduction Sensor

HMD (Oculus Rift DK2)

The HMD allows a user to see the environment as if they are a person in the environment. It also gives rudimentary head orientation data. **The Oculus app must be running in order for participants to see properly in the VEL** (albeit, it can be minimized).

One thing to ensure before allowing a patient to wear the headset is that the "Health and Safety information" screen, which occasionally pops up, has been dispelled. To do this, put on the headset and point the reticle at the box below the warning for a few seconds.

Because of processing reasons, **the experimenter should not use the Vizard computer for anything other than the VE L when experiments are in progress**, as the VR environment may not render correctly in the

headset if the ___ program is minimized or unfocused. This is doubly important because of the input that the "Go Switch" provides, detailed below.

Intersense Inertia-cube 4

The Intersense Intertia-Cube 4 is a device that can measure orientation in 3D space along three primary axes - pitch, yaw, and roll. If you are unfamiliar with these axes, there is plenty of documentation about it online. Treat the Intersense like it is an airplane with the "WorldViz" sticker representing the nose of the plane.

Using the Intersense Inertia-Cube4 is very easy. All you have to do is plug it into the computer, and the Unity project should start responding (as long as you start to play after the cube is connected). There is a device utility for the Intersense that's currently only on the Vizard computer. It's in the folder on the desktop marked "Calibration Tools." It gives you an opportunity to see the raw data coming in from the Intersense.

Go Switch

The Go Switch is a Makey Makey that has been outfitted to send a "Tab" signal when a user presses down on the switch. This allows the VR environment to determine when the user has pressed the switch, and is therefore in a suitable location.

It is currently attached to the table in a fixed position, and should function normally without any calibration, as long as it's plugged in.

Note: Because it is programmed to send a "Tab" to the computer whenever a the switch is depressed, it should be unplugged when the VEL is not in use. Users of the Vizard computer should be wary of patients/coworkers pressing the switch when the VEL is not in use if the switch is plugged in.

DAQ Card

The VEL has the ability to command a DAQ unit to output single analog 5V square waves, provided information is properly entered when creating the config file. In order to properly configure the card, it must be plugged into a USB port on the Vizard computer, and the "Analog output" channel on the card that is specified in the VEC must be hooked up to the device which you'd like to receive the signal.

PPT Cameras

The PPT cameras are high intensity cameras set up around the VR workstation. We have six of them, and they capture data at 144 hz, and can capture movement as small as millimeters. To turn on the PPT cameras, plug in the power cable with the tag on it. After that, wait four or five minutes and turn on PPT studio, an application on the main computer. Under "File," click "Load calibration" and look in the folder it takes you to for any file with the word "_LATEST" at the end. This is the current calibration file for the PPT cameras.

Currently, the PPT system is tracking the head and the wrist. It does this through the small white spheres that can be mounted on the headset "horns" or one the wrist-mounted pad. These two devices both need to be charged. We have several of each, so one tracker of both types should always be charging at the charging area.

**Note: Occasionally, there will be an `opticalheading.dll` error which appears in the lower left of PPT Studios N. This seems to appear somewhat randomly. If this appears, please restart the program.

Calibration Details

You may need to calibrate the PPT cameras occasionally, particularly if you are getting wonky behavior in the VR environment, or if you know they have been moved or bumped.

In order to calibrate, you first need to start up "PPT Studio N". With the PPT cameras on (described above) and **ALL TRACKERS AND/OR OBJECTS OFF OF THE EXPERIMENT TABLE**, as well as the lights off to minimize interference, you can begin calibration by clicking the "Calibrate" button, towards the upper left of the main frame in the PPT program.

You should switch on the calibration rig which is usually stored in the right of the lab table, and place it roughly on the center of the table, ensuring that the axes are pointed in the correct direction. First checking to confirm the calibration rig size is set to "Small", you should press the "Start Calibration" button. This should begin tracking the calibration rig with the cameras. If any of the cameras on the right show a red dot, you need to re-position the calibration rig and start over. Positioning of the rig is not an exact science, but generally the rig should be seen towards the center of the each camera.

Once all cameras calibrate to the rig correctly, you need to reset the "zero" point of the cameras back to our accepted zero, which is the front and center divot in the table. **You must do this step in order for the VR program to function normally.** To do this, you should place a hand tracker, which should show as tracker 3, on the table such that the white tracking marker is as close to the zero-divot as possible. You should then record the "X", "Y", and "Z" fields of tracker 3 from the "marker data" section of the program, and change the "Global Offset" values to such that they are negated versions of the values that you just recorded (ex: if tracker 3 shows x,y,z: .5,7,-2.3 when placed in the divot after calibration, the Global Offset should be set to x,y,x: -.5,-7,2.3).

After calibrating and resetting the zeros, you should save the calibration as the date with the string "_LATEST" appended to the end. You should also rename the old calibration to no longer have the "_LATEST" string appended to it.

Congrats, if you did all of this correctly, the PPT camera systems should be calibrated and functioning correctly!

ADDING EXTRA TRACKERS

There is rudimentary support for adding more than the necessary trackers to the VEL, purely for data capturing purposes. These will not show in experiments. To add them, you need to configure the 'Marker Id' module on the left side of 'PPT Studios N' to include the extra markers. **Note: After configuring extra markers, you should always restart PPT Studios N before running any experiments. Also, Virtual IDs 1, 2, and 3 are reserved to be used as the two HMD markers and the wrist marker respectively, so please do not remove or change these markers when adding extra.**

Miscellaneous Notes for Development

Version Control

One of the hardest parts about working in software development are the pesky other developers. While you're making changes to the code, they're often changing the same files, maybe even the same lines as you are! How should the computer know which lines of code to run? If people are working are different files, how do we make sure we have the right version of each file in the "master" program that will be run?

If you've never heard of it before, this is where I introduce version control. Version control is the best solution we have to this issue, and the specific provider of our version control is Github. Github is an online service that lets multiple people work on the same project. It also allows a single user to work on multiple versions of the same project, in case he or she wanted to test different possibilities without having to keep track of changes.

The tool we use to interact with git is called Git Bash. It simulates a BASH environment, and you interact with it through commands. There are also GUI based tools that you could use if this interface proves too daunting, but your humble README writer had no git BASH experience coming into the project and now can use it well enough.

There are plenty of great resources to learn about git on the Internet. I won't tell you everything there is to know about git, but I will give you some basics to look deeper into:

- **Branches** - a branch is a single avenue of your code. You can have as many branches as you want.
- **Master** - the "master" branch is the main branch. Code on master should be well-tested and integrated such that it won't cause an issue with other code in the project.
- **Commit** - when you commit your changes, you are saying that it forms the new standard for this branch.
- **Checkout** - checkout is the command you use to switch branches.
- **Merge** - When you merge two branches, you combine them. As you might expect, there may be conflicts- if one branch says a variable should be 5, and another branch says 6, which do you trust? In these cases, called **merge conflicts**, you'll have to view the file and individually resolve each by choosing which code segments you want and which you don't. You'll know if you resolved all merge conflicts if the Unity project no longer throws errors and runs as expected.

This has just been a shallow dive into version control, but you should really check it out if you don't know about it already. It'll be useful throughout the rest of your career.

Tunik Board

If you've ever worked in software development before, then you'll know how useful it is to have an issue board for keeping track of issues that arise during development that you don't have time to fix now. For this, we have what we like to call the "Tunik board," but what you may have heard of as a scrum board or a task manager.

The url to access our board is complex and full of vowels in an order that changes when you're not looking, so I personally recommend you use the following link. We currently have four categories: Backlog, Todo, In Progress, and Complete. Backlog refers to any problem that you don't have to think about right now, but you don't want to forget to do. Todo refers to problems that we want to have fixed in the near future. In progress is what you are currently working on or currently testing. Complete refers to anything that is happily complete and has been merged to the master branch.

You will need to be added to the Tunik board. Once you are, you can create new tickets using the easy to use interface. You can tag a problem with any of the tags we've created or add new tags as necessary. You can also assign a problem to a user, either to yourself if you want to take care of it or to another developer if you want to be a bit passive-aggressive (kidding, that's totally allowed).

FOR ANY QUESTIONS NOT ANSWERED BY GENE OR OTHER PEOPLE IN THE LAB, YOU CAN CONTACT THE CREATORS OF THE VEC AND VEL, AS WELL AS THIS README AT Huntoon@live.com AND Berin.s@husky.neu.edu