



SCHOOL OF COMPUTING
UNIVERSITY OF TEESSIDE
MIDDLESBROUGH

TS1 3BA

ELI Beamlines
Hand Tracking Module
for Controlling Interactive 3D Spaces
in Virtual Reality

BSc Computing Project

Oleksii Strapchev

(student ID: S6361012)

November 29, 2017

Supervisor: Bohus Ziskal

Second Reader: Dr. Stefano Cavagnetto

Abstract

The aim of the project is to develop module that could be integrated with a Virtual Reality based application, and allow the users to control a 3D environment using hand tracking only. This module would be used for virtual reality 3D applications developed by ELI Beamlines laser research facility to provide researchers with training and prepare them to using real-life equipment. The project addresses the problem of hand tracking and gesture recognition being a complex task that a developer not familiar with the specifics would find very complex.

The research of the topic was undertaken using specialised literature, published academic articles, official documentation for developers, and user community discussions on the Web. The research, together with the communication with the client of the project has led to the formulation of a set of functional and non-functional requirements.

The platform selected for the project implementation is Unity 3D game engine. The virtual reality headset is Oculus Development Kit 2, and the gesture recognition hardware is Leap Motion Development Kit. Official software development kits for both Leap Motion and Oculus DK2 were utilized throughout the project. The programming language selected for the implementation of application logics is C#.

Table of Contents

Introduction	5
Problem Statement.....	5
Solution	5
Rationale	6
Project Deliverables	6
Client Collaboration	6
1 Development Methodology	7
1.1 Software Development Approach	7
1.2 Scrum Methodology.....	8
1.2.1 Stages of Scrum.....	9
2 Research.....	9
2.1 Personal Experience.....	9
2.2 Unity.....	10
2.2.1 Official Documentation	10
2.2.2 Unity Community	10
2.2.3 Core Findings.....	11
2.3 Oculus.....	12
2.3.1 Oculus developer documentation	12
2.3.2 Technical requirements	13
2.3.3 Core Findings.....	13
2.4 Gesture Recognition	14
2.4.1 Impact on the Project	14
2.5 Leap Motion.....	15
2.5.1 Technical Requirements.....	15
2.5.2 Leap Motion SDK.....	15
2.5.3 Official Documentation	15
3 Requirements.....	16
3.1 Functional requirements.....	16
3.2 Non-functional requirements	17
4 Design.....	18
4.1 Overview	18
4.2 Structure	18
4.2.1 Inventory menu.....	18

4.2.2 Object Instantiation	19
4.2.3 Navigation System Using Hand Tracking.....	20
4.2.4 Transformation Tool	21
5 Implementation	21
5.1 Development Environment configuration	21
5.1 C# Programming Language	22
5.2 MonoBehaviour	22
5.3 Core Implementation.....	22
5.3.1 Scene Preparation.....	22
5.3.2 Inventory.....	23
5.3.3 Object Instantiation	24
5.3.4 Navigation	25
5.3.5 Object Manipulation	25
5.4 Development History	26
6 Testing.....	26
6.1 Verification Testing	27
6.2 Validation Testing	28
6.3 Performance Testing.....	28
6.4 User Interface Testing.....	29
7 Project Evaluation	29
7.1 Overall Evaluation	30
7.2 Unity.....	30
7.3 Hand Tracking	30
7.4 Oculus DK2	30
7.5 Future Considerations.....	31
7.5.1 Addressing the Limitations	31
7.5.2 Expanding the Functionality	32
Conclusion.....	33

List of Tables

Table 1: Minimum technical requirements for Oculus	13
Table 2: Minimum technical requirements for Leap Motion	15
Table 3: Functional requirements	17
Table 4: Non-functional requirements	17
Table 5: Development environment configuration	22
Table 6: Requirement verification testing	28

List of Figures

Figure 1: Inventory menu logic	19
Figure 2: Item instantiation.....	20
Figure 3: Gesture-based navigation	20
Figure 4: Inventory structure and setup	23
Figure 5: Item structure	23
Figure 6: Inventory menu in the scene	24
Figure 7: Object instantiation setup	24
Figure 8: Object instantiated on the working surface	25
Figure 9: Transformation tool structure	26
Figure 10: Transformation tool and target object representation in the scene.....	26
Figure 11: Frame rate test.....	29

Introduction

The output of the project is to provide a system that would allow interactions with a 3D environment developed in Unity 3D game engine using gesture recognition with the help of Leap Motion hand tracking hardware. The secondary deliverable is the documentation guide that would provide guidance for the developers on how to integrate the module into their Unity 3D project. The tertiary deliverable is a demonstration application that showcases the core features of the module developed.

The report consists of an introduction, seven sections, and a conclusion section. The first section focuses at the software development methodology utilized within the project. It covers the principles of the methodology selected, and how it has affected the process of project development throughout the project lifecycle. The second section focuses on the research conducted during the preparation to project design and development phases. The third section discusses the requirements to the project outcomes. The fourth and fifth sections discuss the design of the final solution, and its implementation respectively. Sixth chapter discusses the test conducted in order to verify that the solution implemented conforms to the requirements stated. The seventh chapter critically evaluates the solution delivered, and states the direction for future development. Finally, the results section covers the project status, and concludes the report.

Problem Statement

Currently the problem is that there is no module that widely accessible to Unity developers that would allow complex interactions with a 3D environment using hand tracking. A developer that decides to integrate Leap Motion hand tracking technology into their application would need to conduct extensive research on the documentation for Leap Motion and Oculus software development kits, and develop the solution themselves (Wright, 2016). Such situation is problematic since it greatly slows down the development process for complex virtual reality software with hand tracking based controls.

Solution

The problem will be solved by developing a module that would be easy to integrate into an existing project, would be reliable, expandable, and provide a basis for creating virtual reality applications featuring complex environment interactions. The solution will be available in the form of a *Unity package*, which is the default way for importing collections of assets into a Unity project (Unity Technologies, 2017).

Rationale

The rationale of the project direction lies within the fact that such module would allow developers to build applications with hand tracking environment interactions more easily. Such benefit would lower the entry barrier for developers inexperienced with hand tracking technologies, and potentially strengthen the community of virtual reality developers.

Project Deliverables

The deliverables that will be created in this project include:

1. Package for integration with Unity projects.
2. Guide for package integration
3. Demonstration application with the delivered features showcased.

Client Collaboration

The project was developed specifically for ELI Beamlines research centre, which is a facility focused at scientific research within the fields of physics, material science, biomedical research and astrophysics. However, the outcomes of the project may also be used in other fields ranging from entertainment to design facilities that utilize virtual reality solutions. The project was developed under the supervision of Jakub Grosz, a virtual reality specialist employed by ELI Beamlights, and with cooperation from other employees in the facility.

1 Development Methodology

A software development model is a representation of the approach to software development process. Choosing a methodology does not restrict the software development process to using this particular methodology exclusively. Rather, different methodologies may be fully or partially adopted for different components or phases of a project, Somerville (2017, p. 30) states.

1.1 Software Development Approach

The choice of a methodology is done according to its benefits within current context. The methodologies selected for the project discussed in this report is a combination of *iterative development* and *reuse-oriented software engineering*. The benefit of such approach over incremental methodologies lies within the fact that the project can react to changing requirements more easily. This ensures that the client does not end up with a system that is not useful due to the fact that the needs have changed while the development was under way.

According to Somerville (2017, p. 31), the core idea behind *iterative development* approach is flexibility. In this methodology, a project is developed in *iterations* that are released in short intervals, and exposed to the client. Each iteration is a fully operational piece of software that however may have limited functionality compared to the final product. Such approach enhances the communication between a developer and a client and, helps to react to requirements that were not strictly defined from the beginning.

Reuse-oriented software development in the classical understanding implies the reliance on a large library of reusable components. This is highly relevant to our project since it allows us to utilize a large amount of objects that have been created by the developers of Unity, Oculus and Leap Motion. Additionally, within the context of the project discussed this approach is also focused at creating these *reusable components*. This is justified by the fact that the module created will be used not only by the author, but by the client's developers, and for applications that are completely independent from the project discussed.

The model for the project development was determined to be the Agile iterative *approach*. The choice of model is justified by the following development conditions:

1. The development team is small.

2. The requirements are not fully defined at the beginning of the project. They will rather evolve to react to the requirements of the client on the basis of communication and feedback.
3. Problems and difficulties are likely to be discovered during the development process due to the complexity of the project. According to Sommerville (pp. 57-58), iterative methods are more suited to finding alternative solutions and workarounds as oppose to iterative approach.

Further analysis of the development conditions led methodology to the choosing Scrum software development methodology.

1.2 Scrum Methodology

Scrum is a type of agile methodology focused at managing iterative software development. According to James (2015), Scrum was developed in as the response to the flawed software development methodologies such as waterfall. It emphasises the importance of project progress analysis, and project direction adaptation during the development process.

Somerville (2011, p. 74), suggests a list of benefits that may be yielded when Scrum methodology is adopted:

1. The whole project may be split into manageable logical parts.
2. The project development is not incapacitated by changing or undefined requirements.
3. Communication within the team is improved.
4. Clients may provide feedback on increments, thus ensuring the development process in conformity with changing business requirements.
5. Trust between the customer and development team is established.

One of the features of the Scrum methodology is the use of daily team meetings, where team members share information about their progress, problems faced and the work planned for the next day. Due to the nature of the project, the fact that the development team has only one member and that there is no place for the developer and client to meet on a daily basis, the daily meetings feature has been omitted.

According to International Scrum Institute (2017), product backlog is a core feature of Scrum. It includes all the tasks that need to be done within the project. Product is a living document, which means that the entries may be added or removed from it. Every entry in the backlog is prioritized, and the effort required for fulfilment is estimated. The backlog for the project discussed in this paper is presented in section 8 Implementation.

1.2.1 Stages of Scrum

Three main stages are discerned within the Scrum methodology:

1. The outline planning phase determines the general objectives of the development process, and outlines the software architecture.
2. Sprints are the key feature of the Scrum methodology; they are the basic building blocks of the development process. A sprint is a planning unit of a fixed length, typically two weeks long. Within a sprint, the work to be done is assessed, planned, implemented, and delivered to the client. During a sprint, the work to be done is selected from a product backlog. The final version of the project is delivered to the customer at the end of the last sprint.
3. Project closure is the final phase of a Scrum process. In this phase, the documentation is finalized and experience gained from project development is assessed.

2 Research

The research phase is a critical part of the project development cycle. It allows identifying the requirements, limitations, and alternative solutions for the overall project. Additionally, the research phase provides the author with an understanding of the environment the project will be deployed into.

The initial environmental requirements as well as the aim of the project were provided by the client, Jakub Grosz, who is the virtual reality specialist at ELI Beamlines. Thus, the hardware and software environment was determined. The platform of choice was decided to be Windows 10, the engine to be Unity 3D, and the virtual reality headset and hand tracking equipment – Oculus DK2 and Leap Motion respectively.

2.1 Personal Experience

Personal experience with the Unity 3D game engine, along with my previous career in game development allowed me to grasp the vast potential of the platform for making computer games. Additionally, the fact that ELI Beamlines along with large companies as NASA, Microsoft, and Lego (Unity Technologies, 2017) are using the platform for the development of non-game related software, suggests that the potential is not limited to game development only.

The personal experience with virtual reality and gesture recognition technologies while much more limited than that with Unity in general, lead to the author's personal opinion that these technologies are promising for creation of interactive and intuitive environments. This also encouraged further research the area, and eventually led to working on this project.

2.2 Unity

According to Unity Technologies (2017), Unity is a game engine that offers a user-friendly interface that can be used for development both basic and complex interactive applications. It provides tools for rendering 2D and 3D objects in the scene, calculating physics, and implementing logics with the use of such programming languages as C# and JavaScript. Additionally, it provides a library of extension tools through the official repository called Unity Asset Store.

According to Loizos (2017), utilizing Unity game engine significantly reduces the amount of resources needed to create a produce an interactive application. Moreover, about 50% of games for mobile devices released in 2016, and over 70% of all applications using virtual and augmented reality were built using Unity, Wingfield (2016) adds. Such impressive statistics, while not directly pointing at the benefits of Unity, certainly suggests that many development companies find it appealing to base their products on.

2.2.1 Official Documentation

Unity Technologies keeps a vast knowledge base that provides guidance for using both the graphical user interface of the editor, and the documentation for scripting in both C# and JavaScript. The Scripting Reference section covers all of the classes along with their respective methods that are provided by Unity within their *Monobehaviour* framework.

Additionally, the developers of the engine encourage new developers to study and use Unity for their projects by providing a large amount of guided tutorial videos that are available online. These tutorials cover all aspects of the engine, and range from beginner to expert level.

2.2.2 Unity Community

Unity engine has a massive community of professionals and enthusiasts that are eager to share their knowledge. The discussions and best practices are split between multiple platforms online, the most important of which are indubitably Unity Answers and Stack Overflow.

2.2.3 Core Findings

According to the Unity documentation (Unity, 2017), the game engine utilizes a class *Transform* to handle information about the position, rotation, and scale of the object relative to its parent space. Object position is handled with *Vector* class, which in this case represents Cartesian coordinates of the object. Object rotation is handled through *Quaternion* class internally, but may be manipulated with Euler angles for simplicity. These findings later allowed us to form a theoretical basis for some of the more complex features of the project.

2.2.3.1 Vector manipulations

The Unity engine utilizes three-dimensional vectors for the representation of object positions and directions within a scene. Additionally, they may be used for manipulation of objects, e.g., modifying or setting object position or rotation. Vectors may also be used for more complex operations, which is the case with the project in question.

A vector is a geometric object that has a length or magnitude and direction. It can be thought of as a directed line of fixed length. In three-dimensional space vectors are represented with two three-component sets that describe the point of origin and terminal points, i.e., where from and to the vector is being drawn.

A more common way of vector representation is just using one component that corresponds to the coordinates of the terminal point of the vector; the origin point is thus assumed to be in the origin point of the coordinate system. Thus, a vector $A(a, b, c)$ is a representation of a vector that has its origin point at $(0, 0, 0)$, and terminal point at (a, b, c) .

Principles of vector manipulations are used throughout the project to determine relative positions of objects within the 3D space. A large portion of application logics relies on vector calculations.

One of the core features utilizing vectors involves determining the relative orientation of vectors. In order to determine how to calculate this, we will need to review some of the key notions about vectors. Those are vector magnitude, vector normalization, and dot product of two vectors.

According to Lengyel (2012, p. 13), vector **magnitude** represents the length of a vector, i.e., the distance between its origin and terminal points. If for a three dimensional vector X with represented with points $A(x_1, y_1, z_1)$ and $B(x_2, y_2, z_2)$, the magnitude is calculated with the following formula:

$$||A|| = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2 + (z_2 - z_1)^2}$$

Thus, for a vector with its origin point at (0,0,0), the formula will be the following:

$$magnitude = \sqrt{x^2 + y^2 + z^2}$$

A **unit vector** is a vector with magnitude equal to one unit. Vector normalization is the process of calculating a unit vector pointing to the same direction as the original vector. For an origin vector A (a1,a2,a3), the normalized vector A' can be calculated by dividing each of the vector's attributes by its magnitude. The formula looks as follows:

$$A' = \left(\frac{a_1}{||A||}, \frac{a_2}{||A||}, \frac{a_3}{||A||} \right)$$

Vector **dot product** allows calculating the difference between two vectors' directions. According to Lengyel (2012, p. 17), the dot product of two vectors P and Q may be calculated with the following formula:

$$P \cdot Q = ||P|| ||Q|| \cos \alpha,$$

where α is the angle between the two vectors. Hence, if two unit vectors are used for the calculation of the project, the formula would look the following way:

$$P \cdot Q = \cos \alpha$$

This way, given two vectors we can normalise them and calculate the angle between them using dot product.

2.3 Oculus

2.3.1 Oculus developer documentation

Oculus VR (2017) provides extensive documentation for the supported development environments, including Unity game engine. The documentation covers the process of integrating Oculus utilities into a Unity project, as well as providing some advice on how to setup the project correctly to achieve best performance results. Additionally, it provides a scripting guide that covers the classes within the Oculus utility for Unity, some insight on how they are organized, and how to communicate with them in the game engine.

2.3.2 Technical requirements

It is important to mention that virtual reality technologies are quite hardware demanding, and this will limit the amount of devices the project can run on in the future. The minimum hardware and software requirements offered on the official Oculus VR website are presented in the table below. Additionally, Oculus virtual reality headsets require Oculus software to be installed on the user's computer in order to operate.

Graphics Card	NVIDIA GTX 1050Ti / AMD Radeon RX 470 or greater
CPU	Intel i3-6100 / AMD Ryzen 3 1200, FX4350 or greater
Memory	8GB+ RAM
Video Output	Compatible HDMI 1.3 video output
USB Ports	1x USB 3.0 port, plus 2x USB 2.0 ports
OS	Windows 8.1 or newer

Table 1: Minimum technical requirements for Oculus

2.3.3 Core Findings

The research of Oculus documentation resulted in the discovery of a simple way to integrate Oculus technology with a Unity project. Although Unity provides a built in generic support for major virtual headset devices, Oculus offers an optional utility package which can be integrated into a Unity project. The utility provides tools for controlling camera behaviour, rendering, debugging tools, and other pre-built features that are specific to the Oculus headset.

2.4 Gesture Recognition

As part of the preparation for project design and implementation, research of the field of gesture recognition has been conducted. Gesture recognition systems analyse data obtained from tracking devices to recognize patterns that correspond to gestures. The main benefit of using gesture recognition systems is the fact that they can provide an intuitive interface for interaction between a user and a computer, Barros, et al. (2016) suggest.

There are three main types of gesture recognition systems: systems utilizing on external sensors, systems utilizing other input devices, and systems that use visual data from video cameras. The focus of the research was put onto the latter type since it most closely corresponds to the environmental requirements provided by the client. Namely, Leap Motion belongs to this category.

Systems that obtain visual information from video cameras use images captured to extract data about the tracked object, such as position, position change, movement speed, and others. All of those are dynamic and are very unlikely to fully match for any given gesture. Thus, as stated by Barros, et al., (2016), gesture recognition systems learn, classify, and recognize gestures using complex machine learning and analysis algorithms. Izuta et al. (2014) further argue that in order for gesture to be recognized at real time, the system must recognize a gesture before it is finished. Thus, an algorithm for gesture prediction that would make a decision based on incomplete information should be implemented.

The complexity of dynamic gesture recognition is further underlined by the Leap Motion development team through the fact that all of the previously supported gestures have been deprecated due to low reliability of recognition (Ward, 2016).

2.4.1 Impact on the Project

The initial solution included creating a system for runtime gesture creation and recognition. However, due to the findings discussed above, such approach was deemed to be infeasible. Since the actual interest of the client lied within the development of an intuitive user interface, changes to the requirements have been made. It was agreed to remove the feature of dynamic gesture creation and recognition, and rather focus on the development of the interface.

2.5 Leap Motion

Leap Motion is a small hand tracking device that may be installed on top of a virtual reality headset, or simply put on any surface such as a work desk. It provides the interface for interacting with virtual reality environments using hands only with no additional controlling devices. Leap Motion is particularly attractive for developers due to its relatively low price of 80 USD (Leap Motion, 2017) availability and size compared to closest competitor's Myo (Myo, 2017) device priced at 199 USD.

2.5.1 Technical Requirements

Leap Motion as well as Oculus has some minimal technical requirement that should be taken into consideration. These requirements are presented in the table below. However, it is evident that these requirements are lower than that of Oculus, and thus will not affect the required system configuration for the final project output.

Graphics Card	NVIDIA GTX 970 / AMD R9 290 equivalent or greater
CPU	Intel® Core™ i5-4590 equivalent or greater
Memory	8GB+ RAM
Video Output	Compatible HDMI 1.3 video output
USB Ports	3x USB 3.0 port
OS	Windows 7 SP1 64 bit or newer

Table 2: Minimum technical requirements for Leap Motion

2.5.2 Leap Motion SDK

Leap Motion offers a software kit called Leap Motion's Core Assets for integrating Leap Motion into a Unity project. This by itself kit simply allows rendering hand models in a Unity scene. Additionally, it serves as a foundation for other Leap Motion modules that extend the supported functionality. One of these modules is the Interaction Engine, which adds support of Unity physics engine. According to Leap Motion (2017), this module provides the foundation for creating interactions with other object within a scene.

2.5.3 Official Documentation

Leap Motion keeps detailed documentation for both the Core Assets and optional modules. The documentation contains information on all of the classes that comprise the development kit, along with their dependencies, methods, and other information.

3 Requirements

Conducting an in-depth research of the relevant areas, along with requirements elicitation processes allowed us to determine a defined set of requirements and proceed into the design phase for the application.

Project requirements were prioritized according to their importance within the final aim of the project, and assigned priorities *Must Have*, *Should Have* and *Could Have*. The functional requirements were assigned a higher priority and were based mostly on the requirements provided by the client. The non-functional requirements due to the methodology selected were more flexible and concerned the way the functional requirements would be met, as well as requirements to the development environment, user experience, etc.

3.1 Functional requirements

Functional requirements are those directly linked to the functionality provided by the product.

ID	Description	Priority
ELI-FUN-01	Module should be able to work with 3D models supplied externally.	Must Have
ELI-FUN-02	A system for avatar movement within the scene must be developed.	Must Have
ELI-FUN-03	Object position must be modifiable at runtime of the application. Object must be moveable on the horizontal plane (axes X and Z).	Must Have
ELI-FUN-04	Object orientation must be modifiable at runtime of the application. Object rotation around the vertical axis must be modifiable.	Must Have
ELI-FUN-05	An inventory of available objects must be developed. User can place objects from the inventory into the scene, and from the scene back into the inventory.	Must Have
ELI-FUN-06	User must be able to access an inventory at any time.	Could Have
ELI-FUN-07	The module must rely on the hand tracking data only. No additional input devices should be required.	Should Have
ELI-FUN-08	A system for rendering laser beams and tracking their position and points of hitting objects within the scene must be developed.	Could Have
ELI-FUN-09	Objects may only be positioned on surfaces that represent working surfaces.	Should Have

ELI-FUN-10	Object position must be restricted to a grid. The grid is to be rectangular with a side of 10cm.	Should Have
ELI-FUN-11	Object orientation within the scene must be restricted to steps of 5 degrees.	Should Have

Table 3: Functional requirements

3.2 Non-functional requirements

ID	Description	Priority
ELI-NF-01	Module should be integrable with Unity engine versions starting from 5.6.	Must Have
ELI-NF-02	Module should be compatible with Oculus DK2 and newer models.	Must Have
ELI-NF-03	Module should be compatible with Leap Motion Developer Kit.	Must Have
ELI-NF-04	Hardware requirements for running the module must not exceed those for the current version of the Oculus application.	Should Have
ELI-NF-05	Controls developed within the application must be intuitive, and easy to approach for new users.	Must Have
ELI-NF-06	Inventory menu must be dynamically scalable as more objects are added to the inventory. Must support up to 10 items at a given time.	Could Have
ELI-NF-07	The module should showcase performance that would not make user uncomfortable.	Must Have

Table 4: Non-functional requirements

4 Design

The following section covers the design process of the individual components of the application. The design process was done prior to the implementation of these components, and is detached from the actual programming code. However, the capabilities of the development environment were taken into consideration to ensure that the design can in fact be carried onto the implementation.

4.1 Overview

It was decided to create the within the Unity editor as interactive scene incorporating all of the required features. In this scene the user would be able to move around, place different objects on a working surface, manipulate these objects' position and rotation, and view how a graphical representation of a laser beam would behave in the setup environment. Objects in this context would be the graphical representations of equipment available at the laboratory, such as laser emitters and lenses. Assets and scripts from the scene would then be compiled into a Unity package for distribution and integration with different projects.

4.2 Structure

The overall structure of the scene will follow the format:

- 3D environment
 - Character controller
 - Oculus camera
 - Leap motion hands
 - Inventory menu
 - Inventory item
 - Working surface
 - Equipment unit
 - Transformation tool

4.2.1 Inventory menu

These vector manipulations discussed in the Research section are used in many components developed for the project. For example, in order to design a menu that would “sit” on the user's palm, and would not get in the way when not needed, we create two vectors: one pointing forward from the position of

the user's eyes determined by Oculus, and the second upwards from the point in the centre of the left palm of the hand model generated by Leap Motion. If the dot product between these two vectors is less than -0.9 (angles between approximately 160 and 200 degrees), the menu would be displayed; otherwise it would be hidden and inactive.

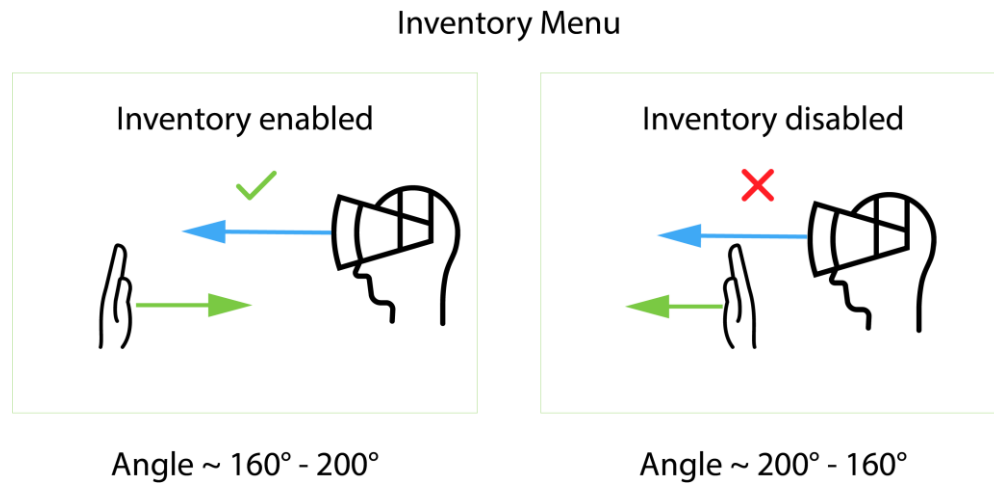


Figure 1: Inventory menu logic

The inventory menu is organised on a grid structure to allow dynamic population of the inventory with items. This approach removes the need to manually position items within the menu, and offers a simple way to modify the menu structure, e.g., changing the number of items in a single row.

4.2.2 Object Instantiation

Object instantiation allows placing items in the scene on runtime. This feature incorporates several manipulations with vectors discussed in the Research section. Every inventory item will have a position assigned to it relative to the inventory. When an item is dragged from its position and released, a script will evaluate the distance from its initial 'anchor' position using the formula for magnitude of the vector discussed in section 3.2.3.1 Vector Manipulations. If the item is released within a certain distance from its 'anchor', the item will return to its original position. If the item is released outside of a certain radius, the module will draw a line from the position of the item downwards, and determine whether an acceptable surface is underneath it. If there is a working surface, the object will be instantiated at the point of contact between the line and the surface.

Item instantiation

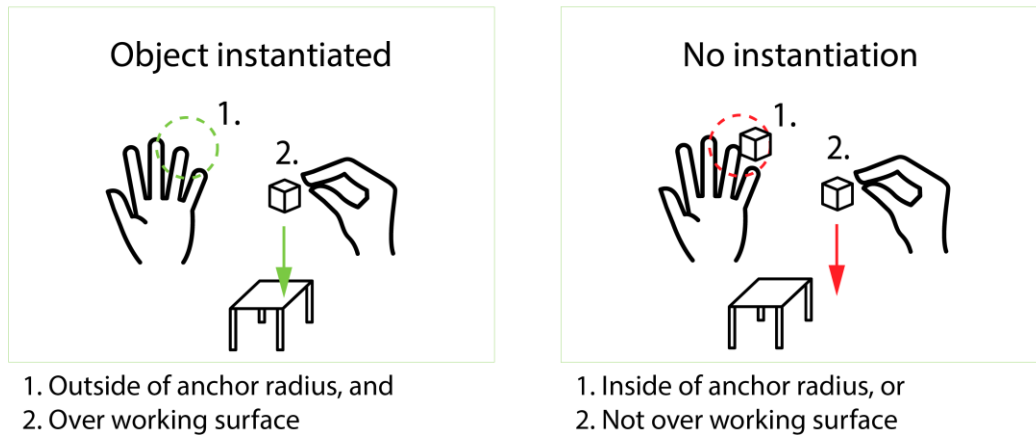


Figure 2: Item instantiation

4.2.3 Navigation System Using Hand Tracking

Another feature designed using the same is controlling the user's avatar position using hand gestures only. However, in this system four vectors had to be used instead of two. Those vectors include: vector pointing forward from the position of the camera, and vectors pointing forward from the position of the tips of index, and middle fingers and thumb.

Gesture-based Scene Navigation

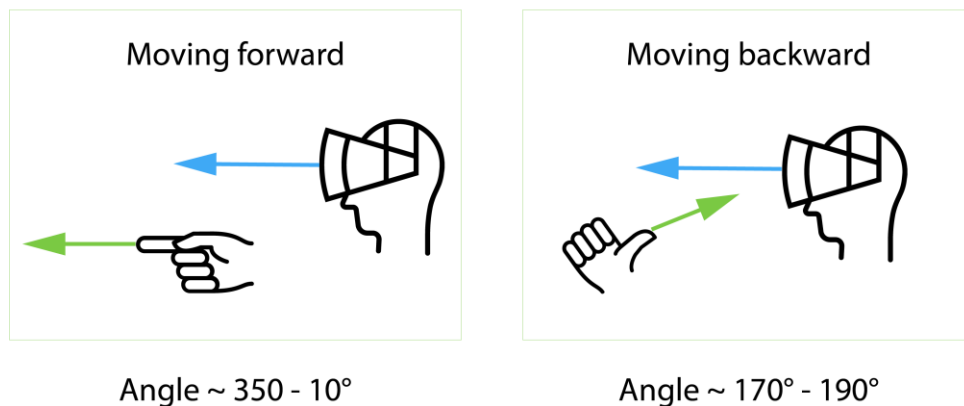


Figure 3: Gesture-based navigation

The user can move forward using his index finger, and move backwards using the thumb. For the forward movement, it is calculated whether the angle between camera and index finger vectors is close to 0 degrees, and the angle between index finger and middle finger vectors is close to 180 degrees. This way we can recognize that the user is pointing forward as oppose to just having all fingers extended and

pointing away from the camera. To trigger the backwards movement the user has to point the thumb towards the camera. No other conditions are applied as it is anatomically awkward to point away from the camera with the index finger, while pointing towards the camera with the thumb. Thus, no conflict of movement direction arises.

4.2.4 Transformation Tool

The transformation tool is the class that allows manipulating object position and rotation. The tool also utilizes Vector and Quaternion classes that Unity game engine uses for setting position and rotation of objects. It is a complex tool that is comprised of the tool itself, and hierarchically organized handles.

The tool is attached to the object it handles the position of, and manipulates that object's position and rotation upon receiving a notification from one of its handles.

The handles are designed to add the information about the change of their position or rotation to a buffer. When a handle is released, it notifies the transformation tool about the total change of position or orientation.

5 Implementation

The following section describes the process of design implementation. The implementation process follows the Scrum software development methodology discussed in Section 3. All of the tools and technologies used will be also listed and evaluated here.

5.1 Development Environment configuration

The software solution was developed using a computer with the following configuration:

Hardware configuration	
Central processing unit	Intel i5-4470
Graphics processing unit	NVIDIA GTX 970
Memory	8GB of RAM
Hand tracking hardware	Leap Motion Development Kit
Virtual reality hardware	Oculus Development Kit 2
Software configuration	
Operating system	Windows 10 Pro
Development environment	Unity version 5.6.1f1

Leap Motion core assets	Version 3.2.0
Leap Motion Interaction Engine	Version 1.0
Oculus development tools	Version 1.16.0

Table 5: Development environment configuration

The configuration roughly matches the minimal technical requirements for Unity, Oculus, and Leap Motion discussed in Subsections 4.2.2, 4.3.2, and 4.5.2 respectively.

5.1 C# Programming Language

Programming of individual components within the scene was done using C# programming language. According to Microsoft (2017) C# is a type-safe object-oriented programming language. Detailed discussion of the benefits of C#, type-safe, or object-oriented programming languages is outside of scope of this report. However, it is worth mentioning that using an object-oriented programming language reduces the amount of code needed to be written, allows using pre-created components, and is generally considered to be more suitable for large projects as oppose to other programming paradigms (Obbayi, 2010).

5.2 MonoBehaviour

MonoBehaviour is the base class for all classes created within Unity (Unity Technologies, 2017). Using MonoBehaviour allows inheriting a large amount of pre-created methods through the principles of inheritance and polymorphism, and significantly reduces the workload on a developer. As a basic example of two such methods, *Start()* allows setting variable values and executing code on the very first frame within the scene, while *Update()* allows running some code every frame.

5.3 Core Implementation

This section will cover the concept of how various features within the final system were implemented. Additionally, a brief description of the implementation within Unity game engine using the C# programming language will be presented for some of the core features. Please refer to Appendix A: Code Examples for the source code of the features discussed.

5.3.1 Scene Preparation

A scene was created in the Unity Editor to incorporate all of the features within the project. Oculus for Unity and Leap Motion Core Assets and Interaction Engine modules discussed in sections 3.3.4 Core Findings and 3.5.2 Leap Motion SDK respectively were downloaded from official websites and imported

into the scene. Similar scenes were later added to the project for the purpose of prototyping individual components independently.

5.3.2 Inventory

Inventory was created starting from the hierarchical structure. The menu itself is nested on the palm of the left hand generated by Leap Motion. The menu contains the mark-up for placing the inventory items on a rectangular grid.

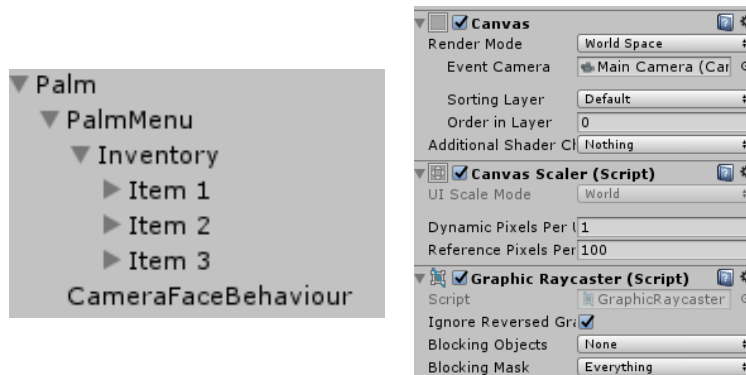


Figure 4: Inventory structure and setup

Items are placed inside the menu. Each item follows the same structure and is instantiated from the same item class, ensuring simple addition of new items to the scene.

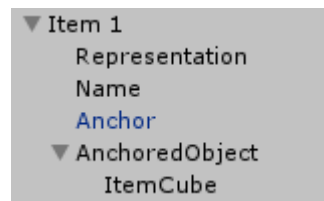


Figure 5: Item structure

An item contains the following components: *Interaction Behaviour* is a component supplied by Leap Motion that allows the interaction between the item and the hands generated by Leap Motion. *Release Behaviour* is a custom component that allows running code upon reaching some conditions, i.e., instantiating objects when the item is released outside of a certain radius from its initial position. Object instantiation is discussed in the next subsection. Additionally, items have a physical representation that includes a cube for user to grasp, and the item name that ensures that the user is selecting the correct object to instantiate.

The item structure together with the inventory layout allows creating a menu that is scalable, and items in which are organized with neat and equal intervals.

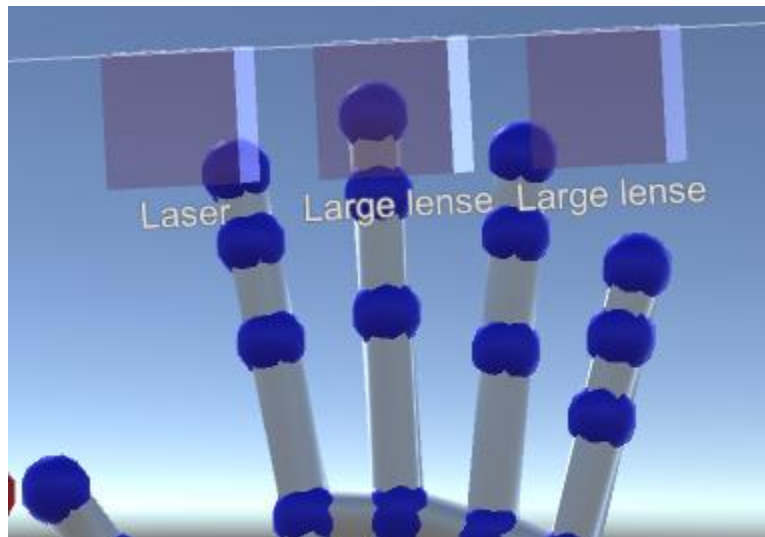


Figure 6: Inventory menu in the scene

5.3.3 Object Instantiation

Object instantiation is done according to the design discussed in section 5.2.2 Object Instantiation. The object that will be instantiated is set up for each individual inventory item through in the Unity Editor.

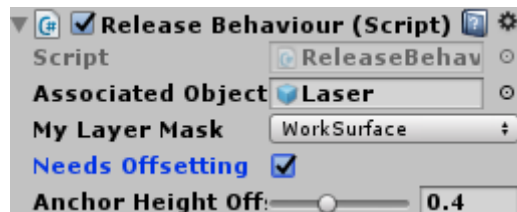


Figure 7: Object instantiation setup

An object is instantiated directly below the position of the inventory item upon release outside of the radius of 0.3 units from the initial position of the item in the menu.

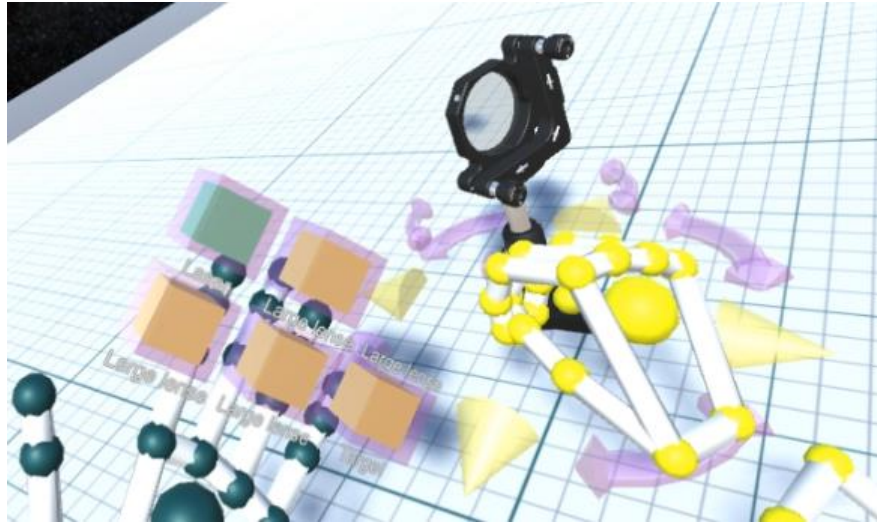
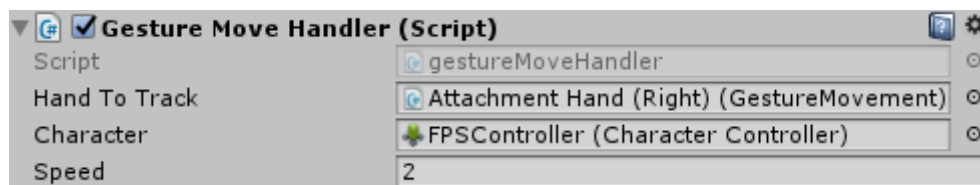


Figure 8: Object instantiated on the working surface

5.3.4 Navigation

Hand tracking based navigation is realised according to the design presented in 5.2.3 Movement System Based on Hand Tracking. The hand to track in this case is setup in the editor so that an interface for left-handed users may easily be set up.



5.3.5 Object Manipulation

Object manipulation is realised through creation of the *TransformTool* and *Handle* classes. The implementation fully follows the design described in section 5.4.2 Transformation Tool.

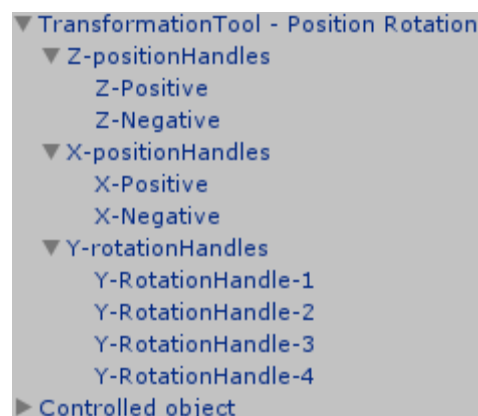


Figure 9: Transformation tool structure

Target object to manipulate may be modified inside the editor, but is setup by default to the object the tool is attached to.

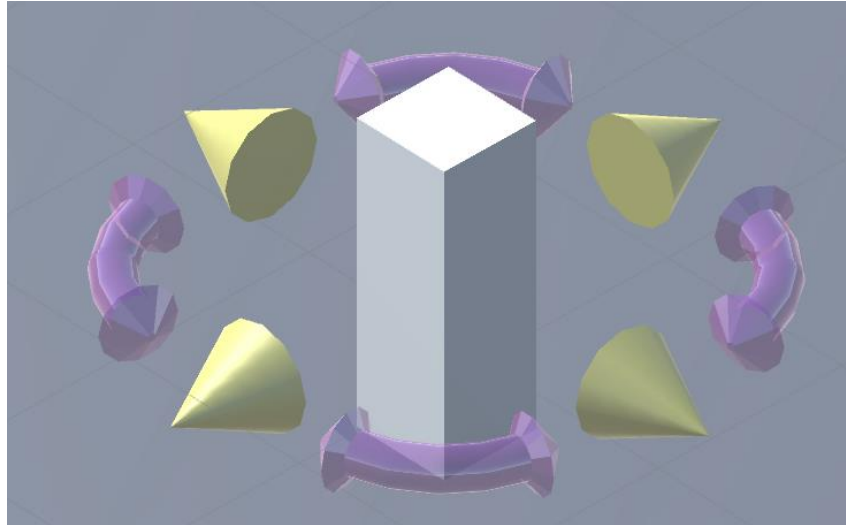


Figure 10: Transformation tool and target object representation in the scene

5.4 Development History

Project backlog, as discussed in the Section 3, contains all of the tasks that have to be done on the project. Please refer to Appendix D: Project Backlog for the full version of the project backlog.

6 Testing

The following section addresses the testing of the implementation, and provides results for the tests conducted. Additionally, it covers specifically performance and user interface testing, as these were the most challenging tests to assess.

Testing is a crucial part of software development. According to Sommerville (2011, pp. 206-208), the purpose of testing is to detect undesirable behaviour of program operation while running it in a controlled environment. Correctly undertaken testing also ensures that software is providing the functionality required, and that defects are removed before product goes live.

Tests of individual components were conducted prior to the end of each sprint to identify and remove defects from the software. Additionally, relationships between components were tested upon integration to ensure that multiple components do not conflict with each other. At the end of the development process the whole project was tested as a whole to confirm that it works as intended, and meets the client's expectations.

6.1 Verification Testing

Verification testing's concern is whether the product meets all of its functional and non-functional requirements. All of the individual requirements were tested and evaluated. In order to do so, expected results of each individual test were compared to the actual results. A test is considered to be successful if the actual results correspond to the expected ones.

Test ID	Expected result	Outcome
Test-01	Module supports 3D models created outside of Unity.	Passed
Test-02	User can move the scene using gesture based navigation.	Passed
Test-03	Intractable objects can be moveable on a horizontal plane.	Passed
Test-04	Intractable objects can be rotated around vertical axis.	Passed
Test-05	Intractable objects can be instantiated from the inventory menu.	Passed
Test-06	Inventory is accessible from anywhere in the scene.	Passed
Test-07	Module is fully controllable by gestures.	Passed
Test-08	Beams are rendered in the scene correctly, can be blocked or reflected by other objects.	Passed
Test-09	Objects cannot be placed outside of specified areas, i.e. working table.	Passed
Test-10	Objects cannot assume positions other than those allowed by snapping mechanism.	Passed
Test-11	Objects cannot assume orientations other than those allowed by snapping mechanism.	Passed
Test-12	Module is supported in Unity 5.6 and higher.	Passed
Test-13	Module is compatible with Oculus Development Kit 2.	Passed
Test-14	Module is compatible with Leap Motion Developer Kit.	Passed
Test-15	Module can run on a system with minimal technical specifications	Passed

	for Oculus software.	
Test-16	Users familiar with virtual reality technologies are able to navigate the scene and perform all operations within 10 minutes from introduction to the module.	Passed
Test-17	10 items can be placed in the inventory menu at the same time.	Passed
Test-18	Frame rate does not drop below 60 frames per second under normal operating conditions.	Passed

Table 6: Requirement verification testing

6.2 Validation Testing

Validation testing involves checking whether software meets the specifications agreed on with the customer. Since the developer's tests are inherently artificial, additional tests in a close to operating requirement are required. This is achieved through user testing. Namely, the customer and other users were invited to test application and share their experience while being monitored.

Acceptance testing is the formal testing conducted by the customer that serves to decide whether the system is ready to be accepted. It was agreed that the acceptance criteria would include all of the required features to be implemented. In addition to the fact that these features were tested and accepted after each sprint discussed in Section 7, the system as a whole was tested in the end of the development process. It was negotiated that all of the required features are implemented, and are working under the agreed standards (please refer to Appendix B: Solution Evaluation by the Client for further details). Thus, acceptance testing was passed successfully.

6.3 Performance Testing

In the context of virtual reality applications, the importance of application performance cannot be overestimated. According to Oculus (2017), an application that runs at frame rates lower than 60 frames per second may cause users to feel discomfort and dizziness. Such situation is extremely undesirable; hence extra attention was given to the performance of the module.

In order to do so, a special scene has been setup with a number of objects close to the expected maximum that would be used within the application in the initial steps of integration with ELI Beamline's application in the future. The expected number of intractable objects was set at 20 units. Then a scene was tested on a computer with specifications with close minimal technical requirements, and the frame

rate was measured. The test was specifically addressing non-technical requirement *ELI-NF-07* and test *Test-18*.

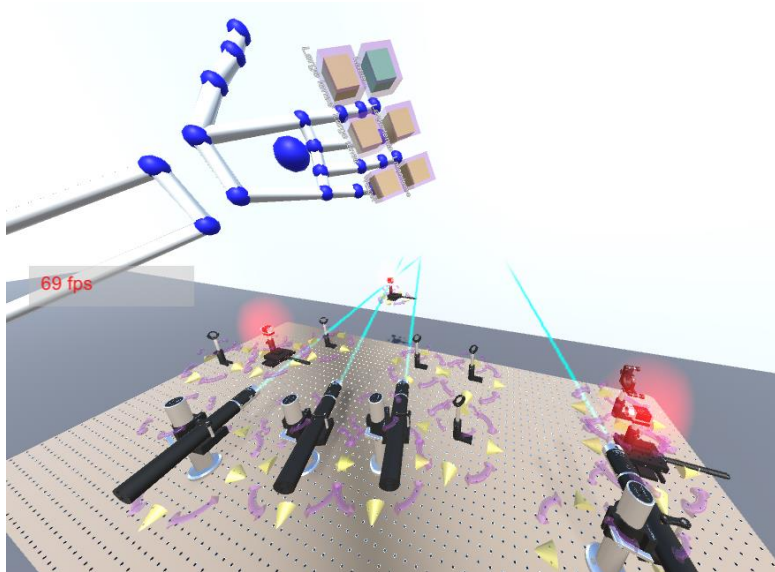


Figure 11: Frame rate test

The frame rate during the test was fluctuating between 60 and 80 frames per second, but did not drop below 60 at any point. As evident from the screenshot above, the test results were satisfactory.

6.4 User Interface Testing

User experience was a crucial part of the project, since that module is focused at providing intuitive and enjoyable user experience. It was a particularly difficult requirement (*ELI-NF-05*, test *Test-16*) to assess due to the notion being highly subjective. However, tests had been performed during and after the development and feedback has been gathered. Most of the comments included suggestions for improvements to the gesture based navigation and the inventory system. These systems have been adjusted and reconfigured until testers reported improvement to the user interface and judged it as intuitive and simple to adopt.

7 Project Evaluation

The following section will critically reflect on the project deliverables and propose a direction for future development. Some of the individual components of the application will also be reflected upon.

7.1 Overall Evaluation

The results of testing as presented in section 7 Testing prove the project to be successfully delivered. All of the required components were developed, and are fulfilling their purpose as intended.

7.2 Unity

Unity proved to be a reliable and intuitive development environment overall. The functionality it allows in terms of programming logics together with a convenient and powerful graphical user interface considerably decreases the amount of time and effort required from a developer. During the development process there were no defects that would be caused by the environment itself, or by the pre-created code supplied by Unity Technologies. Bugs would rather be caused by the incorrect use of the supplied components. However, the official documentation and community efforts discussed in section 3 Research allowed quickly removing such bugs in most cases.

7.3 Hand Tracking

It has been proved through user testing that the hand tracking based user interface was delivered and serves is intuitive and simple to grasp. However, some of the limitations of Leap Motion hand tracking are worth to critically reflect upon.

The first limitation is the relatively low accuracy of such controls. Due to the technical limitations of Leap Motion Development Kit, the precision of manipulations with objects in 3D space is limited. However, these limitations do not significantly affect the features incorporated in the project as no high-precision manipulations were intended or implemented.

Another limitation involves the angle at which Leap Motion is capable of tracking and recognizing user's hands. This field of view angle is 150°. This creates a limitation together with the fact that the device is attached to the virtual reality headset. Namely, when the user manipulates an object with his hands, and looks away from it, the device can no longer track the hands, and thus manipulation can be unsuccessful. In order to successfully manipulate the 3D environment, user should keep his hands within the field of view of Leap Motion. This limitation removes some of the intuitiveness from the user interface, since in real life a person does not need to look at his hands to interact with objects.

7.4 Oculus DK2

The headset used for the development of the project provides all of the required functionality. Nevertheless, some users reported uncomfortable feelings and motion sickness while using the headset.

According to Rift Info (2015), there may be a number of reasons for getting nausea while using a virtual headset. These reasons range from performance issues such as low frame rate, or delayed response to user inputs, to individual predisposition to motion sickness. As discussed in sections 7.3 Performance Testing and 7.4 User Interface Testing, no issues with low frame rate or high delay were discovered within the application. Thus it was concluded that the issue is indeed within the individual predisposition, and the inherited features of the headset itself.

7.5 Future Considerations

The future considerations for the project include removing the limitations of the solution provided, as well as adding new features and refactoring the currently implemented ones in the context of natural project evolution.

7.5.1 Addressing the Limitations

To address the current limitations we consider upgrading the hardware for hand tracking and virtual reality immersion.

Introducing newer versions of the Oculus headset while not directly benefiting the functional part of the project would improve user experience and further improve user experience. According to Digital Trends (2017), the updated version offers superior sensor accuracy, screen resolution and refresh rate. Such improvements are likely to reduce the change of users getting motion sickness, and to improve the visual experience while using the module.

Leap Motion (2017) has recently announced a new version of their hand tracking hardware that features reduced tracking precision and latency, and a 180° field of view. These improvements will likely reduce the impact of limitations connected with the user interface discussed previously.

7.5.1.1 *Alternative technologies*

Hand tracking using visual data from the camera has a major benefit: there are no physical input devices that user has to hold in his hands to control the 3D environment. However, there is also a set of drawbacks that have been discussed previously. Namely, hands have to be within the field of view of the camera, and the tracking precision is not currently perfect for off the shelf products.

Support of controllers for virtual reality headsets may be introduced to the module. Companies such as HTC (2017) and Oculus VR (2017) offer such controllers for their latest headset models. According to Betters (2015), the controller systems utilize sets of sensors set up around the user to track the position

of the controller. Such approach removes the limitation by the single visual tracker approach with a narrow field of view. Additionally, the latency and precision provided by such systems is higher since controllers are tracked from multiple points. Thus, two aforementioned drawbacks may be minimised at the cost of requiring the user to hold physical controllers.

7.5.2 Expanding the Functionality

The functionality presented in the final version of the module presented in the form of Unity package fully conforms with the requirements at the current stage of project development. However, additional functionality may be added in the future in order to allow using the module for deeper interactions with scientific equipment. Naturally, laser emitters feature more settings than just switching the emission on and off. For example, additional user interface controls may be added to modify the beam intensity, frequency and width. Lenses and mirrors for reflecting the beam can be configured to support variable focal size, focus depth and focus position. Adding these features would require further in-depth research not only on the topics of Unity development and hand tracking, but also on optics and laser physics in particular. Additionally, dynamic gestures may be added in the future. Such gestures may remove the need to directly interact with objects in the scene, and rather allow manipulating the environment remotely. Such improvements will require further cooperation with representatives of the client, including physics specialists from ELI Beamlines.

Conclusion

The project covered in this report successfully fulfilled the requirements provided by the client. The solution was built using Unity engine. Hand tracking was implemented using Leap Motion hardware and software solutions. Virtual reality technologies were integrated based on the hardware and software components produced by Oculus VR. The main deliverable was presented in the form of a Unity package incorporating all of the developed components. Supplementary components include a demonstration application showcasing the package built into a Unity project, and an integration manual that provides guidance for

The project was constantly evolving from the initial proposal, through research, design, implementation, and testing phases that were described in this report. Certain changes to the initial proposal had to be introduced due to the findings uncovered during the research phase. However, an alternative solution was found to fulfil the aim of the project. Eventually, all of the functional and non-functional requirements agreed upon with the client were successfully met, and acceptance tests have been passed.

Project was thoroughly evaluated, which included the identification of limitations of the solution, discussing alternative approaches, and stating the direction for future development. Overall the final solution delivered successfully addresses the purpose it was developed for, and demonstrates the possibilities for manipulating 3D environments in virtual reality using hand tracking based interfaces.

Reference list

Barros, P., et al. (2015) *A dynamic gesture recognition and prediction system using the convexity approach*. Available at: <https://www2.informatik.uni-hamburg.de/wtm/publications/2017/BJFBF17/1-s2.0-S107731421630159X-main.pdf> (Accessed: November 24, 2017).

International Scrum Institute (2017) *The Scrum Product Backlog*. Available at: http://www.scrum-institute.org/The_Scrum_Product_Backlog.php (Accessed: November 24, 2017).

Izuta, R., Murao, K., Terada, T., and Tsukamoto, M. (2014) *Early gesture recognition method with an accelerometer*. Available at: https://www.researchgate.net/publication/261960392_Early_gesture_recognition_method_with_an_accelerometer (Accessed: November 24, 2017).

James, M. (2015) *How Does Scrum Fit With Agile?*. Available at: <http://scrummethodology.com/> (Accessed: November 24, 2017).

Leap Motion (2017) *Leap Motion Controller*. Available at: <https://store-us.leapmotion.com/> (Accessed: November 25, 2017).

Leap Motion (2017) *API Overview*. Available at: https://developer.leapmotion.com/documentation/csharp/devguide/Leap_Overview.html (Accessed: 27 November 2017).

Leap Motion (2017) *VR Developer Kit*. Available at: <https://www.leapmotion.com/product/controller-vr> (Accessed: 27 November 2017).

Lengyel, E. (2012) *Mathematics for 3D Game Programming and Computer Graphics*. 3rd edn. Boston: Cengage Learning.

Loizos, C. (2017) *Unity, whose software powers half of all new mobile games, lands \$400 million from Silver Lake*. Available at: <https://techcrunch.com/2017/05/23/unity-whose-software-powers-half-of-all-new-mobile-games-lands-400-million-from-silver-lake/> (Accessed: November 24, 2017).

Microsoft (2017) *Introduction to the C# Language and the .NET Framework*. Available at: <https://docs.microsoft.com/en-us/dotnet/csharp/getting-started/introduction-to-the-csharp-language-and-the-net-framework> (Accessed: November 25, 2017).

Myo (2017) *Myo Gesture Control Armband*. Available at: <https://store.myo.com/> (Accessed: November 25, 2017).

Obbayi, S. R. (2010) *Structured vs. Object-Oriented Programming: A Comparison*. Available at: <http://www.brighthub.com/internet/web-development/articles/82024.aspx> (Accessed: 27 November 2017).

Oculus VR (2017) *SDK Release Notes*. Available at: https://developer.leapmotion.com/documentation/v2/csharp/supplements/SDK_Release_Notes.html (Accessed: November 24, 2017).

Oculus VR (2017) *Step into Rift*. Available at: <https://www.oculus.com/rift/> (Accessed: 27 November 2017).

Oculus VR (2017) *Unity Developer Guide*. Available at: <https://developer.oculus.com/documentation/unity/latest/concepts/book-unity-dg/> (Accessed: 27 November 2017).

Oculus VR (2017) *Oculus Unity Getting Started Guide*. Available at: <https://developer.oculus.com/documentation/unity/latest/concepts/book-unity-gsg/> (Accessed: 27 November 2017).

Rift Info (2015) *Oculus Rift VR Motion Sickness – 11 Ways to Prevent It*. Available at: <https://riftinfo.com/oculus-rift-motion-sickness-11-techniques-to-prevent-it> (Accessed: 27 November 2017).

Sanders, B. (2014) *Mastering Leap Motion*. Birmingham: Packt Publishing.

Shao, L (2016) *Hand movement and gesture recognition using Leap Motion Controller*. Available at: https://stanford.edu/class/ee267/Spring2016/report_lin.pdf (Accessed: November 24, 2017).

Sommerville, I. (2011) *Software Engineering*. 9th end. Boston: Addison-Wesley.

Unity Technologies (2017) *Unity 2017: The world-leading creation engine for gaming*. Available at: <https://unity3d.com/unity> (Accessed: November 25, 2017).

Unity Technologies (2017) *MonoBehaviour*. Available at: <https://docs.unity3d.com/ScriptReference/MonoBehaviour.html> (Accessed: November 25, 2017).

Unity Technologies (2017) *Company Facts*. Available at: <https://unity3d.com/public-relations> (Accessed: November 25, 2017).

Unity Technologies (2017) *Welcome to the Unity Scripting Reference!* Available at: <https://docs.unity3d.com/ScriptReference/index.html> (Accessed: November 25, 2017).

Unity Technologies (2017) *The leading global game industry software*. Available at: <https://unity3d.com/public-relations> (Accessed: 27 November 2017).

Unity Technologies (2017) *Asset Packages*. Available at: <https://docs.unity3d.com/Manual/AssetPackages.html> (Accessed: November 24, 2017).

Ward, J. (2016) *Orion basic gestures*. Available at: <https://community.leapmotion.com/t/orion-basic-gestures/5030> (Accessed: November 24, 2017).

Wingfielf, N. (2016) *Unity Technologies, Maker of Pokémon Go Engine, Swells in Value*. Available at: https://www.nytimes.com/2016/07/14/technology/unity-technologies-maker-of-pokemon-go-engine-swells-in-value.html?_r=0 (Accessed: November 25, 2017).

Wright, S. (2016) *The Fundamental Steps of Leap Motion Combination in Your Unity 3D Project*. Available at: <https://dzone.com/articles/the-fundamental-amp-steps-of-leap-motion-combinati> (Accessed: November 24, 2017).

Appendix A: Code Examples

The following Appendix offers examples from the programming code written by the author as part of the project. Please note that only the key classes and methods are featured in the examples. Comments are also omitted for the sake of concision. Full version of the code is available as part of *project artefacts* submitted together with this report.

1 Gesture-based Navigation

```
1 public class GestureMovement : MonoBehaviour {
2     public gestureMoveHandler gestureHandler;
3     public bool debug;
4     public GameObject pointingFinger;
5     public GameObject middleFinger;
6     public GameObject thumbFinger;
7     public GameObject cameraTrack;
8     public Action toMoveForward;
9     public Action toMoveBackward;
10    private EnumEventTable _eventTable;
11
12    void Update () {
13        if (debug){
14            Debug.DrawRay(pointingFinger.transform.position.normalized, pointingFinger.transform.forward, Color.red);
15            Debug.DrawRay(middleFinger.transform.position.normalized, middleFinger.transform.forward, Color.blue);
16            Debug.DrawRay(thumbFinger.transform.position.normalized, thumbFinger.transform.forward, Color.green);
17        }
18        if (Vector3.Dot(pointingFinger.transform.forward.normalized, middleFinger.transform.forward.normalized) < -0.8F){
19            if (Vector3.Dot(pointingFinger.transform.forward.normalized, cameraTrack.transform.forward.normalized) > 0.7F){
20                Vector3 flatMovement = new Vector3(pointingFinger.transform.forward.x, 0,
21                pointingFinger.transform.forward.z);
22                gestureHandler.MoveCharacter(flatMovement);
23            }
24        }
25        if (Vector3.Dot(thumbFinger.transform.forward.normalized, cameraTrack.transform.forward.normalized) < -0.7F){
26            Vector3 flatMovement = new Vector3(thumbFinger.transform.forward.x, 0, thumbFinger.transform.forward.z);
27            gestureHandler.MoveCharacter(flatMovement);
28        }
29    }
```

30 }

2 Inventory System

1.1 Inventory Menu

```
1 public class FaceCameraScript : MonoBehaviour {
2     public GameObject CameraToTrack;
3     public GameObject palm;
4     public GameObject palmMenu;
5     public bool debugMode = true;
6     private bool menuEnabled = false;
7     private bool facing = false;
8
9     void Update () {
10         if (CheckFacing()) {
11             palmMenu.SetActive(true);
12         }
13         else palmMenu.SetActive(false);
14     }
15
16     bool CheckFacing(){
17         if (Vector3.Dot(CameraToTrack.transform.forward.normalized, -palm.transform.up) < -0.2F){
18             return true;
19         }
20         else return false;
21     }
22 }
```

1.2 Inventory Item

```
1 public class ReleaseBehaviour : MonoBehaviour {
2
3     private GameObject parent;
4     private AnchorableBehaviour parentBehaviour;
5     InteractionBehaviour interaction;
6     public GameObject associatedObject;
7     private GameObject createdObject;
8     private bool isAttached;
9     private bool isCreated;
10    public LayerMask myLayerMask;
```

```

11     private Vector3 objectSize;
12     private GameObject physicalObject;
13     private GameObject associatedTable;
14     public bool needsOffsetting;
15
16     [Range(0.0F, 1.0F)]
17     public float anchorHeightOffset = 0F;
18
19     void Start () {
20         isCreated = false;
21         parent = gameObject.transform.parent.gameObject;
22         interaction = gameObject.GetComponent<InteractionBehaviour>();
23         parentBehaviour = (AnchorableBehaviour)gameObject.GetComponent(typeof(AnchorableBehaviour));
24         interaction.OnGraspEnd += OnGraspEndHandler;
25     }
26
27     void OnGraspEndHandler(){
28         RaycastHit groundHit;
29         isAttached = parentBehaviour.isAttached;
30         if (!isAttached){
31             Ray ray = new Ray(transform.position, Vector3.down);
32
33             if (Physics.Raycast(ray, out groundHit, 100.0F, myLayerMask)){
34                 associatedTable = groundHit.collider.gameObject;
35                 Vector3 objectPosition = groundHit.point;
36
37                 if (!isCreated){
38                     createdObject = Instantiate(associatedObject, objectPosition.Snap(), associatedObject.transform.rotation);
39                     OffsetPosition(createdObject);
40                     isCreated = true;
41                 }
42
43                 createdObject.transform.parent = associatedTable.transform;
44                 gameObject.transform.parent = null;
45                 gameObject.transform.parent = GetPhysicalObject(createdObject).transform;
46
47                 repositionAnchor(createdObject);
48             }
49         }
50
51         else if (isAttached && isCreated){
52             gameObject.transform.parent = parent.transform;
53             createdObject.SetActive(false);
54             isCreated = false;

```



```

55     }
56 }
57
58 private void OffsetPosition(GameObject _object){
59     physicalObject = GetPhysicalObject(_object);
60     objectSize = GetPhysicalSize(physicalObject);
61     _object.transform.localPosition += new Vector3(0, objectSize.y/2, 0);
62 }
63
64 private GameObject GetPhysicalObject(GameObject associated)
65 {
66     if (associated.tag == "PhysicalObject"){
67         return associated;
68     }
69     else{
70         Transform parent = associated.transform;
71         foreach (Transform child in parent){
72             if (child.gameObject.tag == "PhysicalObject"){
73                 return child.gameObject;
74             }
75         }
76         return this.gameObject;
77     }
78 }
79
80 private Vector3 GetPhysicalSize(GameObject physical)
81 {
82     if (physical.GetComponent<BoxCollider>())
83     {
84         return physical.GetComponent<BoxCollider>().bounds.size;
85     }
86     return new Vector3();
87 }
88
89 private Vector3 GetVisibleSize(GameObject visible)
90 {
91     if (visible.GetComponent<MeshRenderer>()){
92         return visible.GetComponent<MeshRenderer>().bounds.size;
93     }
94     Debug.Log("Object " + visible.name + " doesn't have meshrenderer attached" );
95     return new Vector3();
96 }
97
98 void repositionAnchor(GameObject _object)

```

```

99     {
100         Debug.Log("Offsetting the anchor's position. Before = " + this.transform.position);
101         this.transform.position = _object.transform.position + new Vector3(0, anchorHeightOffset, 0);
102     }
103 }

```

2 Object Manipulation

2.1 Transformation Tool

```

1  public class TransformTool : MonoBehaviour {
2      [Range (-1.0F, 1.0F)]
3      public float offset;
4      public InteractionManager interactionManager;
5      public Transform target;
6      private Vector3 _moveBuffer = Vector3.zero;
7      private Quaternion _rotateBuffer = Quaternion.identity;
8      private HashSet<Handle> _transformHandles = new HashSet<Handle>();
9      private enum ToolState { Idle, Translating, Rotating }
10     private ToolState _toolState = ToolState.Idle;
11     private HashSet<Handle> _activeHandles = new HashSet<Handle>();
12
13     void Start() {
14         if (target == null){
15             target = transform.parent;
16         }
17
18         foreach (var handle in GetComponentsInChildren<Handle>()) {
19             _transformHandles.Add(handle);
20         }
21
22         PhysicsCallbacks.OnPostPhysics += OnPostPhysics;
23         OffsetTool(offset);
24     }
25
26     private void OffsetTool(float _offsetValue){
27         this.transform.position += new Vector3(0,_offsetValue,0);
28     }
29
30
31     public void NotifyHandleMovement(Vector3 deltaPosition){
32         _moveBuffer += deltaPosition;

```

```
33     }
34
35     public void NotifyHandleRotation(Quaternion deltaRotation){
36         _rotateBuffer = deltaRotation * _rotateBuffer;
37     }
38
39     private void OnPostPhysics() {
40         target.transform.rotation = _rotateBuffer * target.transform.rotation;
41         this.transform.rotation = target.transform.rotation;
42         target.transform.position += _moveBuffer;
43         this.transform.position = target.transform.position;
44         foreach (var handle in _transformHandles) {
45             handle.syncRigidbodyWithTransform();
46         }
47         _moveBuffer = Vector3.zero;
48         _rotateBuffer = Quaternion.identity;
49     }
50
51     public void SnapOnRelease(){
52         target.transform.position = target.transform.position.Snap();
53         this.transform.position = target.transform.position;
54     }
55
56     public void RoundAngleOnRelease(){
57         Vector3 pre = target.transform.rotation.eulerAngles;
58         Vector3 post = pre.RoundRotation();
59         target.transform.eulerAngles = post;
60     }
61 }
```

2.2 Handle Superclass

```

1 public abstract class Handle : MonoBehaviour {
2     protected InteractionBehaviour interaction;
3     protected void Start () {
4
5         interaction = GetComponent<InteractionBehaviour>();
6         interaction.OnGraspBegin += OnGraspBeginHandler;
7         interaction.OnGraspEnd += OnGraspEndHandler;
8     }
9     protected abstract void OnGraspBeginHandler();
10    protected abstract void OnGraspEndHandler();
11
12    public void syncRigidbodyWithTransform(){
13        interaction.rigidbody.position = this.transform.position;
14        interaction.rigidbody.rotation = this.transform.rotation;
15    }
16 }

```

2.3 Position Handle

```

1 public class PositionHandle : Handle {
2     TransformTool tool;
3     void Start () {
4         base.Start();
5         tool = gameObject.GetComponentInParent<TransformTool>();
6         interaction.OnGraspedMovement += OnGraspedMovementHandler;
7     }
8
9     void OnGraspedMovementHandler(Vector3 initialPos, Quaternion initialRot, Vector3 finalPos, Quaternion finalRot,
10 List<InteractionController> controllers){
11         Vector3 deltaPos = finalPos - initialPos;
12         Vector3 handleForwardDirection = initialRot * -Vector3.left;
13         Vector3 deltaAxisPos = handleForwardDirection * Vector3.Dot(handleForwardDirection, deltaPos);
14         tool.NotifyHandleMovement(deltaAxisPos);
15         gameObject.GetComponent<Rigidbody>().position = initialPos;
16         gameObject.GetComponent<Rigidbody>().rotation = initialRot;
17     }
18
19     protected override void OnGraspEndHandler(){
20         tool.SnapOnRelease();
21     }
22 }

```

2.4 Rotation Handle

```

1  public class RotationHandle : Handle{
2      TransformTool tool;
3      void Start(){
4          base.Start();
5          tool = gameObject.GetComponentInParent<TransformTool>();
6          interaction.OnGraspedMovement += OnGraspedMovementHandler;
7      }
8
9      void OnGraspedMovementHandler(Vector3 initialPos, Quaternion initialRot, Vector3 finalPos, Quaternion finalRot,
10 List<InteractionController> controllers){
11          Vector3 presolveToolToHandle = initialPos - tool.transform.position;
12          Vector3 solvedToolToHandleDirection = (finalPos - tool.transform.position).normalized;
13          Vector3 constrainedToolToHandle = Vector3.ProjectOnPlane(solvedToolToHandleDirection, (initialRot *
14 Vector3.back)).normalized * presolveToolToHandle.magnitude;
15          Quaternion deltaRotation = Quaternion.FromToRotation(presolveToolToHandle, constrainedToolToHandle);
16          tool.NotifyHandleRotation(deltaRotation);
17          interaction.rigidbody.position = initialPos;
18          interaction.rigidbody.rotation = initialRot;
19      }
20
21      protected override void OnGraspEndHandler(){
22          tool.RoundAngleOnRelease();
23      }
24 }

```

3 Object Snapping

```

1  public static class ObjectSnapping{
2      public static float gridSize;
3      public static int angleStep;
4
5      public static Vector3 Snap(this Vector3 prePosition){
6          Vector3 snappedPosition = new Vector3();
7          snappedPosition.x = Mathf.Round(prePosition.x * 1/gridSize) * gridSize;
8          snappedPosition.y = prePosition.y;
9          snappedPosition.z = Mathf.Round(prePosition.z * 1/gridSize) * gridSize;
10         return snappedPosition;
11     }
12
13     public static Vector3 RoundRotation(this Vector3 preRotation){

```

```

14     Vector3 finalAngle = new Vector3();
15     finalAngle.x = Mathf.Round(preRotation.x / angleStep) * angleStep;
16     finalAngle.y = Mathf.Round(preRotation.y / angleStep) * angleStep;
17     finalAngle.z = Mathf.Round(preRotation.z / angleStep) * angleStep;
18     return finalAngle;
19 }
20 }

```

4 Beam Emission

4.1 Beam Manager

```

1  public static class LaserManager{
2      static List<Emitter> laserList = new List<Emitter>();
3      static List<Emitter> mirrorList = new List<Emitter>();
4      public static void AddToMirrorList(this Emitter newEmitter){
5          mirrorList.Add(newEmitter);
6      }
7
8      public static void AddToLaserList(this Emitter newEmitter){
9          laserList.Add(newEmitter);
10     }
11
12     public static void RestartAllEmissions(this Emitter emitter){
13         if(mirrorList.Count != 0)
14             foreach(Emitter em in mirrorList){
15                 em.StopEmit();
16             }
17         if (laserList.Count != 0)
18             foreach (Emitter las in laserList){
19                 las.SetLaserPoints();
20                 las.StartEmit();
21             }
22     }

```

4.2 Beam Emitter

```

1  public class Emitter : MonoBehaviour {
2      public ParticleSystem emitter;
3      private Vector3 initialOrientation;
4      private Vector3 initialPosition;
5
6      protected void Start () {

```

```




7      initialOrientation = this.transform.forward;
8      initialPosition = this.transform.position;
9      emitter = gameObject.GetComponentInChildren<ParticleSystem>();
10     if (gameObject.tag == "LaserSource"){
11         SetLaserPoints();
12         StartEmit();
13         this.AddToLaserList();
14     }
15     else this.AddToMirrorList();
16 }
17
18 void Update () {
19     if (this.transform.forward != initialOrientation || this.transform.position != initialPosition){
20         initialOrientation = this.transform.forward;
21         initialPosition = this.transform.position;
22         this.RestartAllEmissions();
23     }
24 }
25
26 public void StartEmit(){
27     var em = emitter.emission;
28     em.enabled = true;
29 }
30
31 public void StopEmit(){
32     var em = emitter.emission;
33     em.enabled = false;
34 }
35
36 public void SetLaserPoints(){
37     Ray ray = new Ray(transform.position, transform.forward);
38     RaycastHit hitPoint;
39     if (Physics.Raycast(ray, out hitPoint, 20)){
40         TriggerReflection(hitPoint);
41     }
42 }
43
44 public void TriggerReflection(RaycastHit _hitPoint){
45     GameObject hitObject = _hitPoint.collider.gameObject;
46     if (hitObject.tag == "Mirror"){
47         Emitter laserEmitter = hitObject.GetComponentInChildren<Emitter>();
48         laserEmitter.SetLaserPoints();
49         laserEmitter.StartEmit();

```

```
50         ParticleSystem hitEmitter = hitObject.GetComponentInChildren<ParticleSystem>();
51         var em = hitEmitter.emission;
52         em.enabled = true;
53     }
54     if (hitObject.tag == "LaserTarget"){
55         var light = hitObject.GetComponentInChildren<Light>();
56         light.color = Color.green;
57     }
58 }
59 }
```


Appendix B: Solution Evaluation by the Client

The communication with the client was mostly done through informal meetings to exchange thoughts on the progress. Several formal meetings including visits to the facility have also been conducted. A formal evaluation has been requested to present within this paper. Please see the evaluation below:

 **Grosz Jakub** <Jakub.Grosz@eli-beams.eu> 5:51 PM (2 hours ago) ☆  

to me ▾

Hi Alex,
I am sending the evaluation:

The application developed by Alexei does have the key features as specified in the requirements based on needs at ELI Beamlines European laser research facility. The core functions have been implemented correctly and after fine tuning of additional functions, the application will be available for to visitors and staff. Expected first use will be mainly for testing of the used approach based on optically-tracked hand gesture manipulation and creation of basic optical and experimental layouts in VR. Eventually, full integration of the features developed by Alexei into our Virtual Beamline application that contains complete virtual model of the facility should be tested and implemented, however, this will likely require further development. I look forward to future collaboration with Alexei. He has built a very good tool that has great potential for not only science popularization, but also for new immersive ways for designing complex systems for research and innovation.

Jakub Grosz B.A.

Virtual Reality Specialist
ELI Beamlines
Institute of Physics of the Czech Academy of Sciences
jakub.grosz@eli-beams.eu
[+420 604 648 133](tel:+420604648133)
[+420 266 051 250](tel:+420266051250)
www.eli-beams.eu

Appendix C: Package Integration Manual

Introduction

This document provides guidance for integrating the Gesture Control Module package into your Unity project. The module was designed for controller-free VR experiences.

The module features the following functionality:

- Navigation using hand tracking from Leap Motion
- Interactive menu with a list of items that can be instantiated
- Tools for manipulating object position and rotation on runtime using Leap Motion hands
- System for shooting and reflecting laser beams using Unity's RayCast and Particle Systems

Prerequisites:

Naturally you would need Leap Motion and Oculus devices to make use of this module. Additionally, this module relies on Oculus Utilities, Leap Motion Core Assets, and Leap Motion Interaction Engine packages that are available on the official websites of Oculus VR and Leap Motion respectively. Please download these packages and integrate them into the project prior to proceeding to Gesture Control Module integration.

Download locations:

- Oculus Utilities: <https://developer.oculus.com/downloads/package/oculus-utilities-for-unity-5/>
- Leap Motion Utilities: <https://developer.leapmotion.com/unity>

Integrating the module

1. Right-click in the Unity Project tab and select Import Package.. Custom Package.
2. Navigate to the location of Gesture Control Module package, select it and click Import. A folder called Gesture Control Module will be created in the project view. The folder contains the following prefabs and their associated scripts:
 - a. Inventory menu
 - b. Inventory item
 - c. Transformation tool

- d. Laser manager
- e. Laser emitter
- f. Mirror
- g. Snapping manager

Using the module

In order to successfully use the module please make sure your scene features an Interaction Manager from Leap Motion and a FPS Controller from Unity Standard Assets prefab with LMHeadMountedRig, HandModels, and Attachment Hands attached.

Once you got the components in the scene, simply drag and drop the contents on the scene to use them. Enjoy!

Note: Inventory Menu should be attached to Attachment Hands, and not to the hand models directly.

Appendix D: Project Backlog

Project backlog

The table below presents all of the tasks that have ever been added to the project backlog. In addition to the standard backlog structure, the status of an item is added to this table, the requirement addressed, and the sprint it has been assigned to. Estimation has been done in days it would take to develop the solution. Priorities are set on a 10-point scale, where 10 is the highest priority.

Item	Estimation	Priority	Requirements	Status	Sprint
Setup the scene, integrate Leap Motion and Oculus core assets.	2	10	ELI-NF-01, ELI-NF-02, ELI-NF-03	Implemented	1
Create inventory menu component.	1	10	ELI-FUN-05, ELI-FUN-06, ELI-NF-06, ELI-NF-05	Implemented	1
Develop the item menu class.	4	8	ELI-FUN-01, ELI-FUN-03, ELI-FUN-07	Implemented	1
Develop logics for enabling and disabling the menu.	2	6	ELI-FUN-05, ELI-FUN-06, ELI-FUN-07	Implemented	1
Make the inventory accessible from anywhere in the scene.	1	3	ELI-FUN-07	Implemented	1
Develop system for dragging items out of the inventory and into the scene.	4	10	ELI-FUN-07,	Implemented	2

Develop functionality for instantiating objects after an item is dragged from the menu.	4	6	ELI-FUN-01, ELI-NF-05	Implemented	2
Functionality for positioning objects on a work surface upon instantiation.	2	5	ELI-FUN-09	Implemented	2
Transformation tool for manipulating object position.	8	10	ELI-FUN-10, ELI-NF-05	Implemented	3
Transformation tool for manipulating object rotation.	4	10	ELI-FUN-11, ELI-NF-05	Implemented	4
Snapping functionality for limiting object's position and rotation values.	2	5	ELI-FUN-09,	Implemented	4
Integration of external 3D models.	1	6	ELI-FUN-01	Implemented	5
Transformation tool for manipulating individual components of an object.	3	8	ELI-NF-05, ELI-FUN-01	Implemented	5
System for shooting an invisible ray and triggering an action upon hitting another object.	2	8	ELI-FUN-08	Implemented	5
System for 'reflecting' rays between objects.	3	10	ELI-FUN-08	Implemented	6
Visualisation of the 'beam' using a particle system.	3	7	ELI-FUN-08, ELI-NF-07	Implemented	6

Table: Project backlog

Appendix E: Project Proposal

Computing project proposal

Gesture Recognition Utility for Unity 3D

Prague College

BSc Computing

Oleksii Strapchev

Semester 1701-1704

Word count: 1073

Table of Contents

1 Working title	3
2 Overview	3
3 Motivation/Rationale	3
4 Areas of investigation	4
5 Background research	4
6 Literature overview	4
7 Methodology.....	5
8 Research ethics	5
9 Project Deliverables	6
10 Project Plan	7
Reference list	7

Table of Figures

Figure 1: Project timeline Gantt chart	7
--	---

1 Working title

Gesture Recognition Utility for Unity.

2 Overview

Game development engines allow video game developers to speed up the development process and overcome some of the common problems involved in it Moran suggests (2016). One of the most popular engines nowadays is Unity. During the last couple of years, the emerging field of consumer virtual reality devices has triggered a question of building immersive virtual reality games to the market. While the common input devices such as conventional keyboard and mouse, or special gaming pads are suitable for interacting with virtual reality environments, the technology of hand gesture recognition may offer a more intuitive experience, Plemmons and Mandel add (2016). One of the devices that relies on gesture recognition is Leap Motion.

Currently, video game developers who want to implement Leap Motion interaction into their game have to select their gestures from a library of gestures provided by the Leap Motion company (Leap Motion, 2017). Otherwise, developers seeking to implement custom gestures and assign them to actions have to go through a cumbersome process of defining such gestures parametrically, CS Koh (2013) suggests.

The essence of the project will lie within the development of a hand gesture recognition utility for Unity game development engine. The utility proposed will allow developers working in Unity to create and edit custom hand gestures during runtime, thus reducing the difficulty of development process.

3 Motivation/Rationale

The field of the project was chosen because of my long-lasting interest in interactive applications development. Gesture recognition technologies are closely related to virtual reality technologies as they can potentially lift the restriction of having to rely on physical input devices to control interactive applications. Furthermore, virtual reality technologies have been rapidly developing in the last several years, and are claimed by technology visionaries to be determinative in the future of interactive applications development, Virtual Reality Society (2017). That being said,

one would be safe to claim that the skills acquired through designing and implementing such project would find a sufficient demand on the labour market of the future.

4 Areas of investigation

- Explore the core principles of gesture recognition used in contemporary interactive applications.
- Explore the Leap Motion hardware capabilities and limitations.
- Conduct research on the Leap Motion API for Unity.
- Explore the capabilities of Unity engine to handle runtime gesture recognition.

5 Background research

The background research was mainly focused on the Leap Motion software available for developers, development community resources and the existing solutions available on the Unity Asset store, which is a distribution platform for Unity software extensions, which are developed by Unity Technologies and various contributors (Unity Technologies, 2017).

The background research revealed that there are currently no competing solutions available on the market, despite the fact that some developers have announced their intentions to develop similar products. According to Leap Motion's official website (2017), the company does not restrict developing custom gestures, only a limited number of gestures are supported by the software development kit, which can be used across all applications. However, it provides technical documentation intended to help developers create their custom gestures parametrically, i.e., through code, McInerney (2013) suggests.

6 Literature review

Gesture recognition is an established area of computer science, which according to Mitra and Acharya (2007) possesses a wide area of possible implementations in areas such as medical rehabilitation, hardware operation techniques, and virtual reality. Such recognition has triggered the development of vast number of scholarly articles and publications.

Consumer gesture recognition hardware however is a relatively new concept, and therefore it might be problematic to conduct secondary research of scientific papers on highly relevant studies. Leap Motion provides comprehensive documentation for supported platforms intended to guide aspiring and experienced developers alike in the efforts to integrate Leap Motion hardware into their interactive applications. There is also a broad developers' community online, which conveys the impression of being keen to provide peer support and feedback. Hence, it is intended to perform most of the research on the official Leap Motion documentation, and resources provided by the development community.

7 Methodology

The project will serve to satisfy the development needs of a particular client, therefore the requirements will be collected through interviewing the client's representatives.

A constant dialogue with the client is expected, where the project progress will be reviewed and auxiliary requirements can be adjusted. Therefore, it has been decided that the project will follow the iterative model, although the specific development methodology is yet to be decided.

8 Research ethics

The resulting software created is not intended to be used in risk involving activities. No potentially dangerous hardware will be used in the research and development phases of the project.

The iterative process of project development implies conducting user tests during the development phase. Users will be informed of potential risks involving the use of virtual reality and gesture tracking hardware, e.g., experiencing dizziness or epileptic seizures, according to hardware manufacturers' health and safety disclaimers.

Any personal data collected during the research phase will not be used outside the research, and will not be distributed to third parties unless required by law.

9 Project Deliverables

- Unity asset that will allow creating and saving custom gestures on runtime.
- Demonstration scene.
- User guide.

10 Project Plan

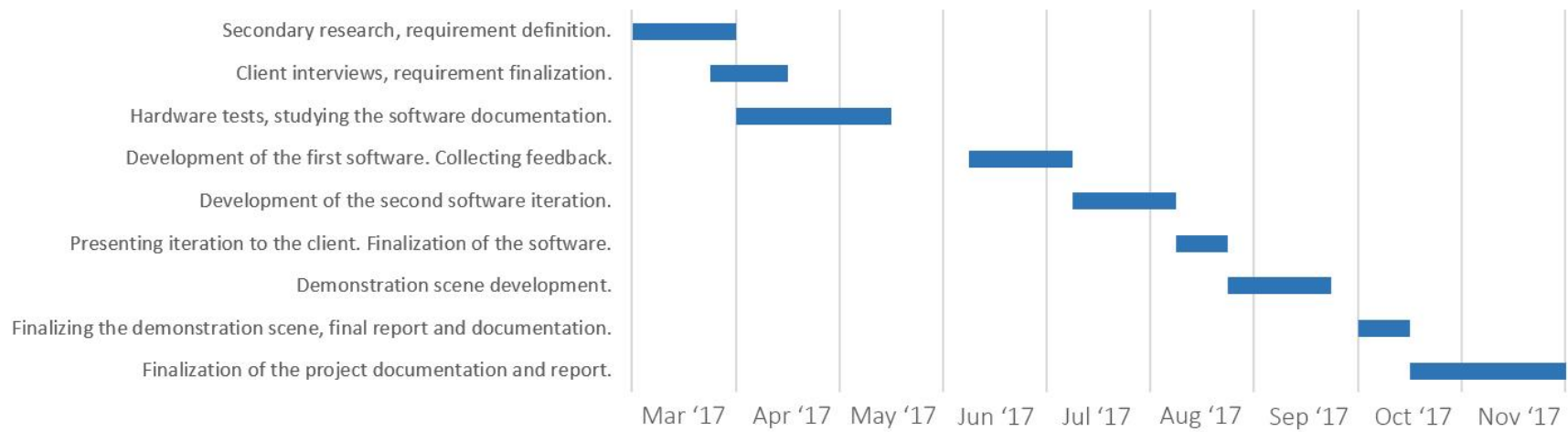


Figure 1: Project timeline Gannt chart

Reference list

CS Koh (2013) *How can you implement non-standard Leap Motion gestures?*. Available online: <http://stackoverflow.com/questions/20473951/how-can-you-implement-non-standard-leap-motion-gestures> (Accessed: March 14, 2017).

Leap Motion (2017) *Gestures*. Available online: https://developer.leapmotion.com/documentation/csharp/devguide/Leap_Gestures.html (Accessed: March 14, 2017).

Leap Motion (2017) *Unity scripting reference*. Available online: https://developer.leapmotion.com/documentation/unity/api/Leap_Classes.html?proglang=unity (Accessed: March 14, 2017).

McInerney, J. (2013) *Learning Custom Gestures in Leap Motion*. Available online: <http://jamesmc.com/blog/testing/> (Accessed: March 14, 2017).

Mitra, S. and Acharya, T. (2007) *Gesture Recognition: A Survey*. Available online: <http://ieeexplore.ieee.org/abstract/document/4154947/?reload=true> (Accessed: March 14, 2017).

Moran, D. (2016) *5 Leading Game Engines for indie game developers*. Available online: http://www.gamasutra.com/blogs/DylanMoran/20160729/278145/5_Leading_Game_Engines_for_indie_game_developers.php (Accessed: March 14, 2017).

Plemmons, D. and Mandel, P. (no date) *Introduction to Motion Control*. Available online: <https://developer-archive.leapmotion.com/articles/intro-to-motion-control> (Accessed: March 14, 2017).

Unity Technologies (2017) *Importing from the Asset Store*. Available online: <https://docs.unity3d.com/Manual/AssetStore.html> (Accessed: March 14, 2017).

Virtual Reality Society (2017) *Applications of virtual reality*. Available online: <https://www.vrs.org.uk/virtual-reality-applications/> (Accessed: March 14, 2017).