ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ
ΣΧΟΛΗ ΜΗΧΑΝΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ
ΤΟΜΕΑΣ ΒΙΟΜΗΧΑΝΙΚΗΣ ΔΙΟΙΚΗΣΗΣ &
ΕΠΙΧΕΙΡΗΣΙΑΚΗΣ ΕΡΕΥΝΑΣ

Creating a virtual environment and scenarios for studying human robot collaboration

---

Δημιουργία εικονικού περιβάλλοντος και σεναρίων για την μελέτη συνεργασίας ανθρώπου ρομπότ

Κοντραζής Δημήτριος-Πέτρος

Σχολή Μηχανολόγων Μηχανικών

5 Ιούλιου 2018

Ευχαριστίες

Θα ξεκινήσω λέγοντας ότι με αυτήν την διπλωματική κατάφερα να ανακαλύψω τι μου αρέσει να κάνω, ενώ πριν ήμουνα στο χάος. Ίσως βρήκα ένα δρόμο που είχα χάσει αφότου μπήκα στην σχολή.

Για αυτό το λόγο θα ήθελα να ευχαριστήσω κατ' αρχάς τον επιβλέποντα μου Επ. Καθηγητή Δ. Ναθαναήλ, ο οποίος μου ανέθεσε τη διπλωματική εργασία με βάση πρώτα κάτι που θα άρεσε σε εμένα. Επίσης για το γεγονός ότι ήταν εκεί καθ' όλη τη διάρκεια εκπόνησης της και μου έδειξε ότι για να επιτύχεις ένα μεγάλο έργο πρέπει να το σπας σε πολλά μικρά κομμάτια καθώς και για το ότι ήταν σε θέση και πρόθυμος να επιβλέπει κάθε κομμάτι.

Ένα μεγάλο ευχαριστώ στον Κ. Γκίκα όπου χάρις σε αυτόν μπόρεσε ένα περιβάλλον το όποιο ήταν ασύνδετο και απείχε από την πραγματικότητα να έχει πραγματικές αναλογίες.

Επιπλέον στον Λ. Ψαράκη ο όποιος σχεδίασε τα σενάρια, βοήθησε σε πολλά πιλοτικά τεστ και ήταν το άτομο το όποιο βοήθησε περισσότερο και με την υπομονή του τέσταρε τα σενάρια της διπλωματικής ώστε τελικά να καταλήξει να δουλεύει σωστά, καθώς και στο γεγονός ότι με βοήθησε να φτιάξω τις αναλογίες ώστε να έχουμε μια ρεαλιστική απεικόνιση.

Τέλος στον φίλο μου Α. Μουρελάτο ο οποίος ήταν πάντα εκεί για να επιδιορθώσει το hardware όταν παρουσιαζόντουσαν προβλήματα καθώς και για το γεγονός ότι με βοήθησε να εισάγω τον άνθρωπο μέσα στο εικονικό περιβάλλον. Ο Α. Μουρελάτος μαζί με τον Β. Γιαγκλίση έκαναν το περιβάλλον πιο ρεαλιστικό προσπαθώντας να φτιάξουν την απόκριση του χεριού καθώς και τη φυσικότητα του.

Τέλος θα ήθελα να ευχαριστήσω όλο το εργαστήριο για την καλή διάθεση και την προθυμία του. Ίσως τελικά εκεί που δουλεύεις το πιο σημαντικό δεν είναι το τι κάνεις αλλά με ποιους είσαι…

Περίληψη

Η εικονική πραγματικότητα είναι σήμερα πιο προσιτή σε περισσότερους ανθρώπους από ό, τι στο παρελθόν και υπήρξε πεδίο έρευνας τα προηγούμενα χρόνια, καθιστώντας προσομοιώσεις όλο και πιο ρεαλιστικές.

Σκοπός της διατριβής μας είναι να δημιουργήσουμε ένα σενάριο σε μια φουτουριστική πραγματική κατάσταση εργασιακού περιβάλλοντος, όπου ένας άνθρωπος και ένα ρομπότ συνεργάζονται μαζί για να ολοκληρώσουν ένα καθήκον στο μικρότερο δυνατό χρονικό διάστημα, λαμβάνοντας υπόψη τις διαδικασίες ασφαλείας, οπότε ο παίκτης κάνει την καλύτερη δυνατή προσπάθεια να αποφύγει το ρομπότ.

Σε αυτή τη διατριβή, θέλουμε να κάνουμε τον παίκτη να αισθάνεται πιο ασφαλής μέσα από διαφορετικές διαδικασίες σεναρίου. Πρώτον, θέλουμε να ελέγξουμε πώς ο παίκτης είναι σε θέση να συνεργαστεί χωρίς εξωτερική βοήθεια και ενώ στα επόμενα δυο σενάρια θα υπάρχει.

Στο δεύτερο σενάριο, ένας έγκυρος τρόπος για να είναι ο παίκτης πιο άνετος γινα να δουλέψει με ένα ρομπότ είναι το ρομπότ να πηγαίνει πιο αργά όταν βρίσκεται σε κάποια κοντική ακτίνα με τον παίκτη, καθιστώντας ετσι την αποφυγή του ρομπότ ευκολότερη.

Στο τελευταίο σενάριο υποδεικνύουμε τις μπάλες που πρόκειται να πάρει το ρομπότ, καθιστώντας το ακόμα πιο εμφανές και πιο βολικό για τον χρήστη και ως αποτέλεσμα να εμφανίζεται ως το πιο ασφαλές από τα προηγούμενα, επειδή μέρος της διαδρομής του ρομπότ αποκαλύπτεται στον Παίκτη.

Executive Summary

Virtual reality is nowadays more accessible to more people than previous and it has been a field of research in the previous years, rendering real world simulations more and more realistic.

The purpose of our thesis is to create a scenario in a futuristic real working environment situation, where a human and a robot are collaborating together to accomplish a task in the least possible time, taking into consideration the safety procedures, thus the player making the best effort trying to avoid the robot.

In this thesis, we want to make the player feel safer through different scenario procedures. First, we want to test how the player is able to co-operate without any external help and external help will be provided in the next two scenarios.

In the second scenario, a valid way to make the player feel more comfortable with working with a robot is to make the robot go slower when in a certain proximity with the Player's, thus making it appear safer to the user and being able to avoid the robot easier.

In the last scenario, we indicate the balls the robot is about to pick up, making it even more profound to the user and expecting this scenario to be the most convenient for the user and appear as the safest of the previous ones, since a part of the robot's path is revealed to the Player.

Contents

A. Table of figures

1. Introduction

The last 5 years VR technologies have become more and more available to people. They were first used in gaming and videos, thus making the experience more thrilling but its scientific value came to light these years.

Now it is widely used in many research fields, the most common one being the medical. Our field of interest is ergonomics and to be more specific human-robot collaboration, where the human and the machine work together to compliment their abilities.

The future factory suggests that there is no separation between automated and manual workstations thus robots and humans collaborate optimally.

Our thesis presents a simple task, where the Player and the robotic arm have to pick up some balls from a table and place them in an conveyor belt, as it would appear to be sent onto a different part of the factory, for a different procedure.

The Player will be able to control his in-game arm through Arduino sensors and the arm's movement will imitate the user's with enough precision to be considered realistic enough.

Our robotic arm is an ABB robotic arm with movement capabilities of 6 degrees of freedom, our robotic arm does not have physical properties, since it is only used for simulation purposes and the model is rigged, thus even animation can fully describe its movement with accurate precision to a real working robot.

In our thesis we plan on testing the hypothesis that there are ways that can make a working environment safer and also more efficient, thus we measure the number of collisions with the robot and the time the experiments last, thus measuring as a total the efficiency of our working environment.

2. Theoretical Background: VR and Human-Robot collaboration in VR

In the theoretical background we present related research concerning VR, VR in robotics and Human-Robot collaboration in VR.

2.1. VR

The term Virtual reality (VR) refers to a computer-generated environment in which the user can perceive, feel and interact in a manner that is similar to a physical place. This is achieved by combining stimulation over multiple sensory channels—such as sight, sound and touch—with force-feedback, motion tracking, and control devices. In an ideal VR system, the user would not be able to distinguish an artificial environment from its physical counterpart. Whilst none of the current VR systems would be able to pass this criterion, significant advances in the perceptual fidelity of virtual environments have been achieved over the last few years. VR now have become a topic of interest also in the scientific community has been a valuable tool for scientific community. Giuseppe Riva et al.(2016) developed a virtual reality setting that can help patients confront their problems in a controlled and safe setting, change their life through VR with meaningful experiences. Antoniou Stratos et al.(2016) wanted to introduce innovative learning frameworks to secondary education, using virtual reality to attract young students at manufacturing by developing a learning process in a CAVE system. Anouk Keizer et al.(2016) used VR on anorexic patients in order to see if they can experience ownership of their virtual body if their virtual body movements is identical to the real one. The previous examples are some fields of interest that virtual reality is a major component and they revolve around it.

2.2. VR in Robotics

Prior work in robotics includes various explorations of the value of VR for robot simulation. Burdea et al.(1999) provides a survey of the applications of VR in robotics. Tang and Yamada(2011) developed a robotics system for a construction robot using virtual reality. They then conducted experiments which confirmed that their method was superior in operability, safety and reduction of stress than the conventional visual display. Belousov et al.(2001) created a virtual control environment for robot teleoperation via internet. Having the working environment of the robot and the robot itself displayed in a dynamic, 3D virtual environment instead of the physical environment allowed suppressed time delay inherent in IP networks and accelerated work efficiency for the operator. Safaric et al.(2003) developed a training system involving usage of virtual robots to provide an inexpensive and safe method for

enterprises to train their employees. They then conducted experiments and confirmed the method to be viable, cheap and efficient. Kawasaki et al. proposed a virtual robot teaching method based on hand manipulability for multi-fingered robots, and demonstrated its effectiveness through a pick-and-place experiment. Last but not least, Oliver Liu et al.(2017) proved that interacting with a robot in virtual reality is far more realistic than interacting with it in just a laptop screen and it improves task performance.

2.3. Immersive VR with First Person integrated body parts

Avatar body representation and control in VEs allows for many types of research in various fields such as game development, H-R collaboration, training, medical rehabilitation, ergonomics, etc. Examples of such research include: Lange et.al (2011), used the PrimeSense depth sensor technology (also utilised in the Microsoft Kinect) to develop and evaluate a game-based rehabilitation tool for the balanced training of adults after neurological injury. Wittmann et. al (2015) developed a VR therapy game that continuously estimates the patient's arm reachable three-dimensional (3D) workspace based on Inertial Measurement Units (IMUs). Luo et. al (2011), created an interactive VR system for both arm and hand rehabilitation utilising both optical linear encoders (OLEs) and IMUs. Osumi et. al (2017) developed a quantitative method to measure movement representations of a phantom upper limb, and investigated whether short-term neurorehabilitation with a VR system would restore voluntary movement representations and alleviate phantom limb pain (PLP), using a combination of the Microsoft Kinect and Leap Motion. Merians et. al (2009) developed a complex system capable of exercising the arm and hand together or in isolation, providing for both unilateral and bilateral hand and arm activities in three-dimensional space. The system incorporated CyberGlove instrumented gloves for hand tracking and a CyberGrasp exoskeleton for haptic effects in a number of VR simulations. Moreover, MRI imaging was used to observe the engaged areas of the brain, in order to test the feasibility of using VE-based sensory manipulations to recruit select sensorimotor networks. All of the above research yielded encouraging results regarding the rehabilitation of patients, as well as gaining positive feedback from both patients and clinicians, thus demonstrating the applicability of combining motion tracking technologies and VR environments for rehabilitation purposes, and especially for offering a solution for the at-home rehabilitation of patients, in an enjoyable environment. Heidicker et. Al (2017) studied the effect of avatar appearance and motion control on communication and interaction in social virtual reality scenarios within immersive VEs. To that end, three different types of avatar in different VEs were compared. The results

demonstrated that motion control of avatar bodies plays an important role in the sense of presence within the VE, with full body avatars with full motion control exhibiting the best results regarding co-presence and behavioural interdependence. It is worth noting however that avatars consisting of head and hands with motion control showed better results than complete avatar bodies with pre-defined animations. A question left open for future research by the paper was how many and which body parts have to be visible to reproduce or even surpass that degree of co-presence. Specifically regarding the field of Ergonomics/Human Factors, by studying the relationship between an avatar body and the person operating it, it is possible to draw valuable conclusions regarding issues such as:

- The correlation between avatar body control and task effectiveness in a VE .
- The user's ability to assimilate a virtual representation of their body with their real-world body image.
- The ability to subsequently incorporate this representation into their body schemas.

For example, Kilteni et.al (2013) observed that subjects' behavioral and movement patterns within a VE can change depending on the visual aspects of an avatar body within said VE, by simulating a drumming task with avatar bodies of different skin tones and clothing. Moreover, it was observed that a stronger body ownership illusion corresponded with a greater behavioral change of the subjects. Slater et.al (2010) studied the illusion of ownership of an avatar body different than the subjects' physical one, demonstrating that a virtual female body that appears to substitute the male subjects' own bodies was sufficient to generate a body transfer illusion. In a different study, Kilteni et. al (2012) studied the ability of subjects to incorporate an avatar body exhibiting asymmetries in comparison to their physical one into their body schemas. This was achieved by creating elongating one the users' virtual arms to up to 4 times their normal length, and using questionnaire scores and defensive withdrawal movements in response to a threat to measure the degree of ownership of that arm experienced by the users. Results showed that users experienced a sense of ownership towards the elongated limb and were able to adapt their responses to this unnatural body image. That illusion did decline, however, with the length of the virtual arm, especially when the virtual arm exceeded three times the length of the physical one. Won et al. presented congruent results by examining the concept of "homuncular flexibility"—the idea that humans can learn to control bodies different from their own by changing the relationship between tracked and rendered motion. To that end, the researchers conducted

two different experiments. In one, the movements of the upper and lower limbs of the users real and virtual were remapped, making the physical arms control the virtual legs and vice versa, or attributing far increased range-of-motion to the virtual legs than the arms. In the other, a third arm was added to the avatar body, controlled by the rotation of the users' physical arms. The results of both experiments demonstrated that subjects were able to adapt their body schemas to the virtual bodies' capabilities, quickly learning how to utilise their more flexible limbs in the first experiment and their "third arm" in the second one to achieve better performance in tasks when compared to "normal" body representations.

2.4. Human-robot collaboration in Virtual Reality

Human-Robot Collaboration (HRC) in production engineering is a research topic for which Augmented Reality (AR) and VR have provided interfaces that, respectively, expands the quantities of features that operators can watch in their field of view or replace it completely with a virtual world. Typical industrial production applications span from manufacturing process simulation, which are able to provide real time enhanced information used for inspection or with focus on training, to collaborative factory planning during product design or redesign. In fact, VR can be used for collaborative (re)designing of production systems when analyzing and evaluating changes prior to implementation. This makes possible to prevent costly design mistakes. Even though some works have considered the human factor as part of the industrial process and adjusted the VR to accurately include the operator movements in the simulation, it is often the case that the operator experiences the VR/AR only from a static position, e.g. standing still or seated, where the input sensors are located. Patrick Ruckbert et al. (2017) wanted to simulate an assembly process with a manufacturing robot using virtual reality in order to increase immersion and to increase flexibility and adaptivity which are key elements of any production process. Luigi Gammieri et al. (2016) believed that safety is a key element to many manufacturing environments, so virtual reality would be an effective tool that is capable of simulating such complex systems with a high level of immersion, so they modeled a kinematic model of a robot that was able to reproduce safe behavior on the real robot and as well as to train operators. Szabolcs Suto et al. (2016) created a simulated environment of a real working space and the purpose would be to digitalize the human's movement in virtual reality so that collisions would not happen in reality. This will give the possibility to stop the robot in time, or to be able to generate a collision free path in real time.

3. Our custom motion tracking system

3.1. Hardware/Wearable

The motion tracking system that was used in this thesis was constructed by a previous thesis in our lab by Antreas Mourelatos.It is a custom build system capable of tracking the human arm beginning from the shoulder and including the palm.

It is consisted of 3 MPU 9250 IMUs connected to an Arduino nano with 24AWG wires and a mounted fingerless glove and elbow patch.The system is very reliable, inexpensive and since it is small it does not obstruct the user's movement.In the next paragraph we briefly describe the main components of our system.

3.2. MPU 9250

The MPU 9250(see Figure 1), produced by InvenSense, is a 9-axis Motion Processing UnitTM (MPU), meaning that it combines an accelerometer, gyroscope and magnetometer for position, orientation and acceleration tracking. It additionally contains an embedded Digital Motion Processing (DMP) unit, which can acquire the data from these sensors, process those utilising data fusion algorithms, and return position and orientation information. A breakout board of the chip was used to better facilitate prototyping and connection to the Arduino. The board, also denotes the IMU's X and Y axes. The Z axis can be inferred from these by the right-hand rule.

Figure 1. Mpu 9250

3.3. Arduino Nano

For this project, a version of the Arduino Nano v3.0 board (see Figure 2), was used as a host processor device, to capture IMU measurements and process them, in order to calculate the orientation and movements of the user's arm. The board was powered through a Micro-USB connection with a PC. The same connection was used to exchange data with the computer via the serial monitor included in the Arduino software, which allows simple textual data to be sent to and from the board. The Arduino receives data from the three MPU 9250 IMUs utilizing I2C communication through the A4 (SDA) and A5 (SCL) pins. I2C communication is supported by the appropriate libraries. The Arduino Integrated Development Environment (IDE) was used for writing the code used in the project and uploading it to the processor

Figure 2 . Arduino Nano

The circuit(see Figure 3):

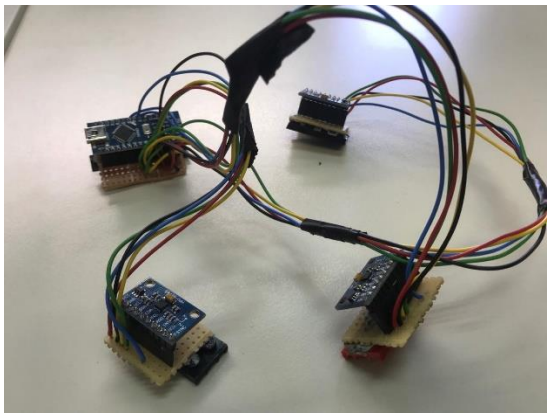

Figure 3. Whole Circuit

And the final Configuration mounted on a person's hand(see Figure 4):

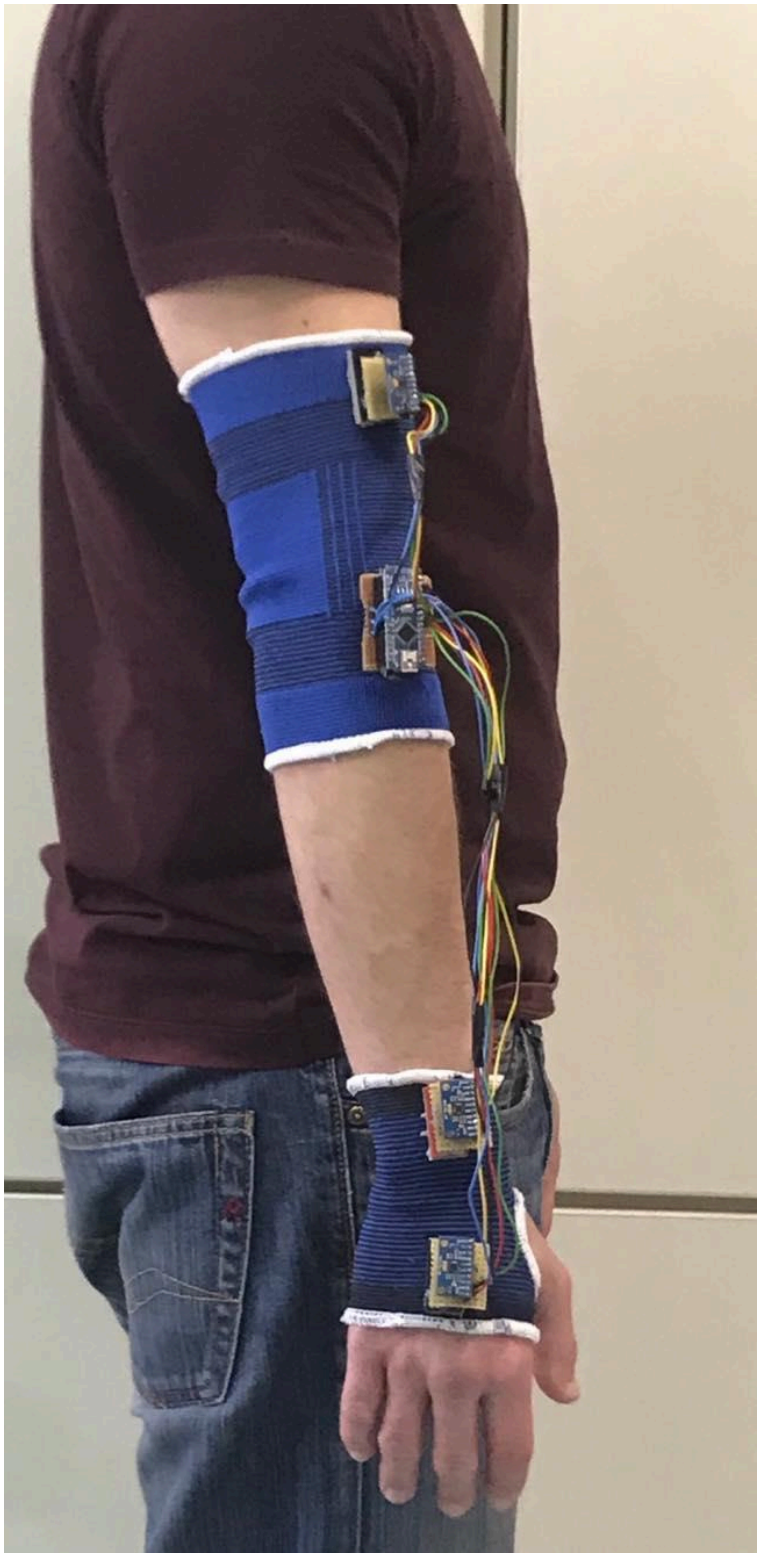

Figure 4 Whole Circuit mounted on a person's arm

One sensor (IMU 1) was placed of the dorsal surface of the palm, approximately over the third metacarpal bone.

One sensor (IMU 2) was placed on the dorsal surface of the forearm, over the wrist joint.

One sensor (IMU 3) was placed on the dorsal surface of the upper arm, over the elbow joint.

Using the abovementioned system our Players are able to control a humanoid avatar in the environment we created, that appears to have physical properties, in order to appear more appealing to the player and feel more immersed and as well as to use these models to create a more realistic and more precise approach maybe to a whole body motion capture system that maybe will lead to make whole body immersive experiences that can simulate even more complex scenarios.

4. Programming Languages (Environments)

4.1. Unity

In this thesis we use as our main programming environment the powerful open Source language called Unity(see Figure 5) in order to make the scenarios as described below.

Figure 5. Unity logo

Unity is a game engine and it is mostly used to make games for all platforms but its uses have expanded over the last years, since the wide use of VR, many “serious games” have arrived and the use of such software has been widely used and upgraded.

4.2. Unity is used for real world simulations.

Given the costs and limitations in gathering real-world data, it is natural to consider replacing or augmenting it with synthetic data generated by a game engine such as Unity. Due to recent advances in graphics hardware, rendering, and advent of virtual and augmented reality, the Unity Engine has evolved into a complete 3D modeling tool, able to generate highly photo-realistic simulations. This has been noticed by both industry and academia, and many projects have been developed to take advantage of Unity’s simulation capabilities. One of the most recognized projects is SYNTHIA.

Developed entirely on Unity by The Computer Vision Center (CVC) at Universitat Autónoma de Barcelona (UAB), the project focuses on creating a collection of synthetic images and videos depicting street scenes in a diverse range of episode variations.

CVC has been pioneering computer vision research for the past 20 years and its SYNTHIA dataset has become a seminal source for those working on autonomous vehicles perception systems.

For Vision Zero, Unity joined forces once again with CVC to provide the City of Bellevue with the best technology and expertise available.

Through imagery and 3D models provided by the City of Bellevue(see Figure 6), and leveraging the integration of Otoy's OctaneRender with Unity Engine, CVC took on creating a set of scenes that can be leveraged to improve both the training and the evaluation of the computer vision models built by Microsoft.

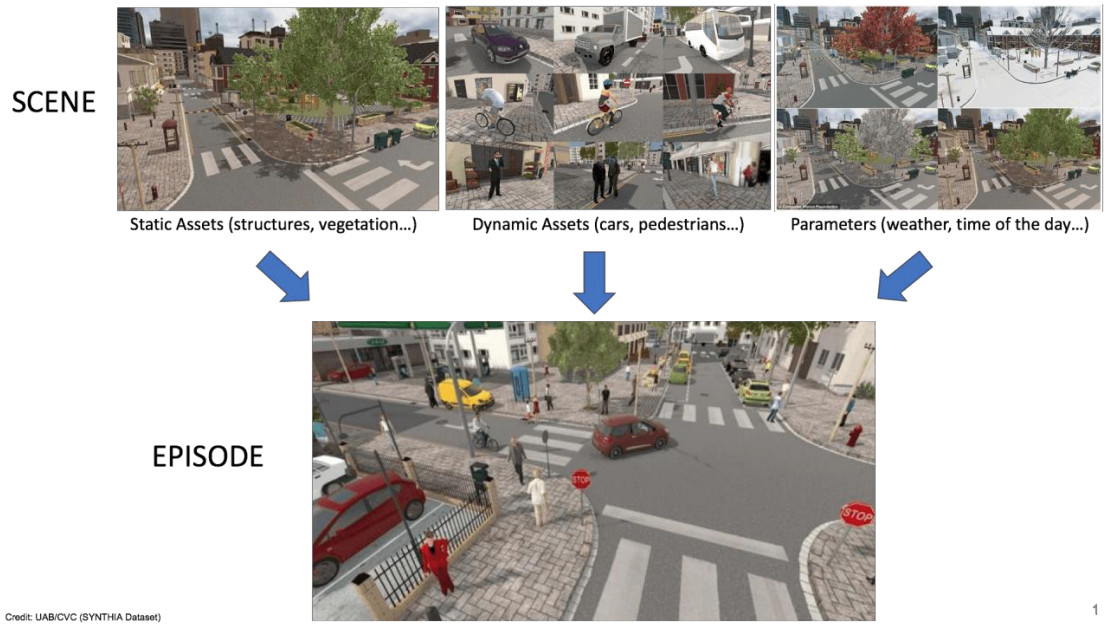

Figure 6. Simulating Real world Scenarios aka Road Scene (Image taken from Unity Website)

4.3. Unity can use CAD data for real time development.

Unity today is used across many industries beyond gaming, including aerospace, architecture, automotive, construction, gambling, transportation, manufacturing, medical, and more. The benefits of using real-time in these industries include accelerating innovation through better design collaboration, developing VR and AR training to improve training outcomes, and creating immersive experiences that increase engagement and drive higher sales.

Industry professionals are increasingly seeing the value of bringing real-time into their workflows, recognizing its potential to completely transform the way products and experiences are conceived and built. Despite the fact that forward-thinking individuals in these industries have made great strides to bring real-time into their creation processes, they have encountered persistent challenges, particularly when it comes to preparing or importing large CAD assemblies for real-time development(see Figure 7).

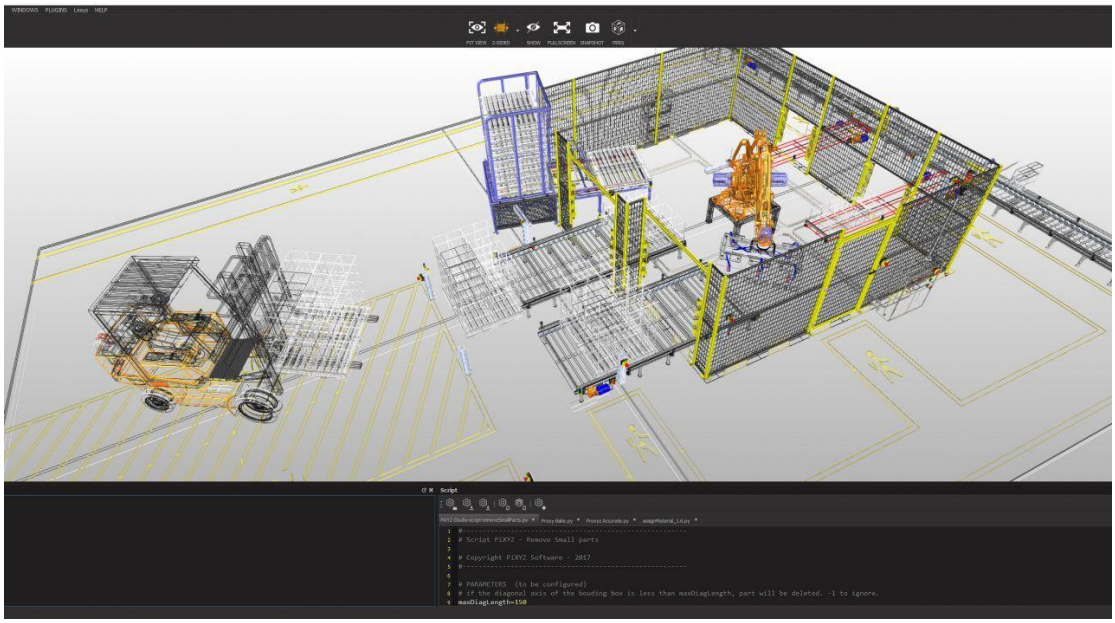

Figure 7 Importing Real Cad models into Unity (Image taken from Unity Website)

4.4. Unity can be used for medical simulations and training.

Since virtual reality immersed, many companies have developed environments where new potential doctors can be trained using real life scenarios, this medical training can include scenarios and real world procedures that may be impossible or extremely expensive to recreate in real world. Companies such as Biofight VR, Osso VR and Archvirtual offer such environments(see Figure 8).

Figure 8 Simulating Real world scenario; Medical operation (Image taken from Unity Website)

5. Creating the environment

5.1. Robotic Animation

As mentioned above, this thesis is about creating a ‘’serious game’’ where a human and a robot have to work together to finish a task. This task is about picking up balls from a table and then putting them in another one. The human and robot are placed opposite to each-other and share the same workspace. This picking task was explicitly conceived as the main challenge for the human subject is to perform fast manipulations while at the same time paying attention not to collide his arm with the robot.

The rigging of the robot was not part of the thesis, since through extensive search a model was found, that was already rigged and it would be easily modified in order to fit our purpose.

5.1.1. Robotic arm

The robot is an ABB robot which is modelled in blender and it is rigged. Rigging is generally used to add control to objects, for the purpose of animation.

The rigging is made with the inverse kinematics of the model, so its movement is natural, smooth and it is not different from a real robot, so it is safe to assume that it can be used in real working environment simulation.

5.1.2. Rigging

Rigging is practically a skeleton that is bound to the 3D mesh of the object we are using. The model rig is like a real skeleton and it is made of joints and bones, so that the animated object can be moved to the desired pose.

- **Joint Hierarchy**

In order for a rig to work properly, the bones and joints must follow a logical hierarchy. When setting up a character's skeleton, the first joint you place is called the *root joint*. Every subsequent joint will be connected to the root either directly, or indirectly through another joint.

- **Forward Kinematics**

Forward kinematics (FK) is one of two basic ways to calculate the joint movement of a fully rigged character. When using FK rigging, any given joint can only affect parts of the skeleton that fall below it on the joint hierarchy.

- **Inverse Kinematics**

Inverse Kinematics (IK) rigging is the reverse process from forward kinematics and is often used as an efficient solution for rigging a character's arms and legs. With an IK rig, the terminating joint is directly placed by the animator, while the joints *above* it on the hierarchy are automatically interpolated by the software. IK is most appropriate when the animation calls for a terminating joint to be placed very precisely.

5.1.3. The task of picking

In order to make a realistic task, the robot has to be able to pick the balls in a similar manner concerning each ball, so animating the robot was rather simple.

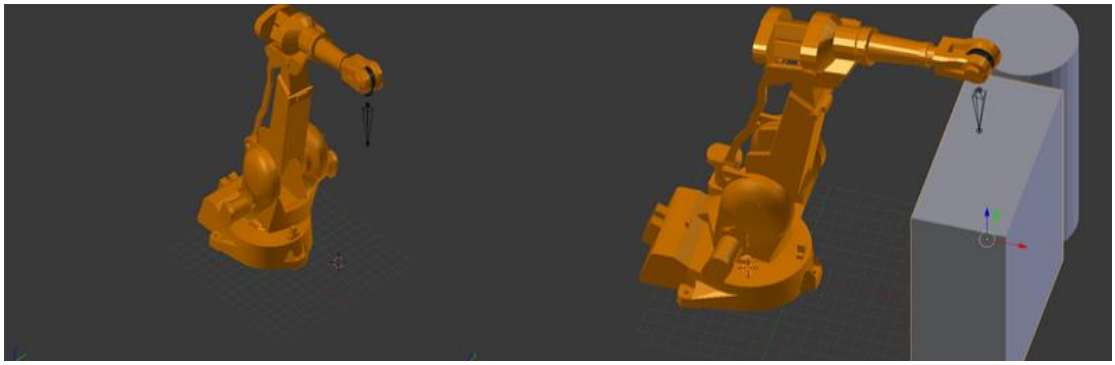

Figure 9 Modifying the Robot

The picture above(see Figure 9) shows the initial setting of our robot as it was downloaded from blender forum.This six digrees of freedom (dpf) robotic arm is rigged and its inverse kinematics are calculated moving just the end effector thus making the motion planning simple.

We continue by changing the setting so that we will be able to place our balls.

We add a table and a “Cylinder”. The purpose of the table is to hold and place our balls.The cylinder will be used as our Placeholder where the player will have to let the item he/she is currently holding in to.

The table’s position to the robot is placed in such a way so that the player can reach the end of the table by fully extending his arm. Respectively the robot’s position to the table is such so that it can reach the end and not be extremely close to the player’s face, thus making it seem dangerous.

In order to create our animation we used the Animation window in blender. This is achieved by dragging the arrow from the robot End-effector to the point where we want our robot to be.

Then in the Action editor we created the states for the robotic arm.

The First state is the **idle** state, where the robot is static, we need that state, as we are going to see later, as a base line action for the robot. This state will be referred to as the starting and ending state of our animation procedure thus saving us time and making the movement and the transition more realistic.

The Second state of the animation is about picking up the balls we intend to place in our experiment. We start by moving the robot's end effector to our desired position on the table.

The whole process of the robot that seems to pick up a ball is shown on the picture shown below(see Figure 10), the example presented is about picking up the first ball.

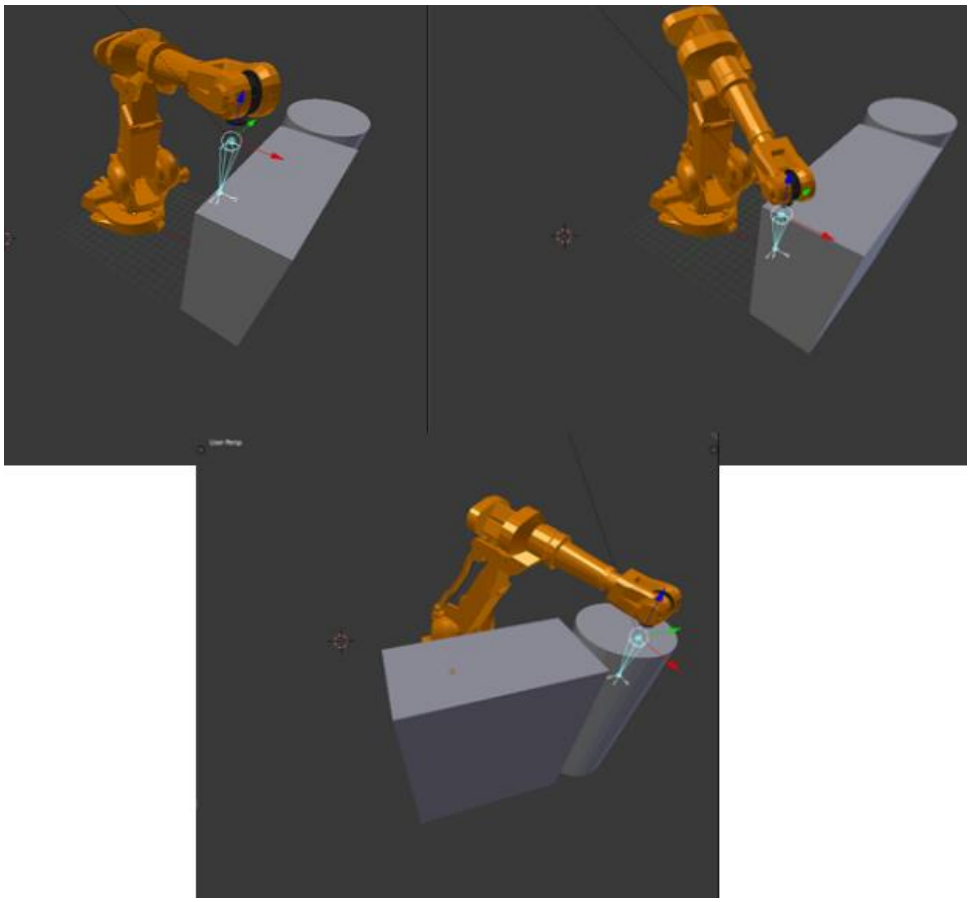

Figure 10 Whole robot animation for picking up a ball

Then we proceed by dragging down the End effector's arrow as a representation of a robot grabbing a ball. This the act could be accomplished both in unity and blender;~~,~~ but we preferred to execute it in Unity in order to

be able to change it more easily, since most animations are hard-coded and are very difficult to change.

For each ball the whole animation procedure lasts about 180 frames or approximately about 7 seconds. Throughout the animation the robot positions are exact in order to make the whole animation realistic and constant, since we do not want any inconsistency with its movement.

The last animation we created is the movement the robot is performing when hitting the player. The robot makes a jittery move to show the player that is being hit and to give feedback to the player. Due to high frame action sequence, it is impossible to

show this animation, in pictures, since only the final product of this animation gives us the effect we want.

We choose to animate the robot's arm movement, not to program it since it is efficient and very easy to make.

5.1.4. Environment

In order to make the scene more realistic for the user, and make him feel like he/she is inside a manufacturing plant we chose to locate it in an industrial hangar(see Figure 11) taken from the blender library.

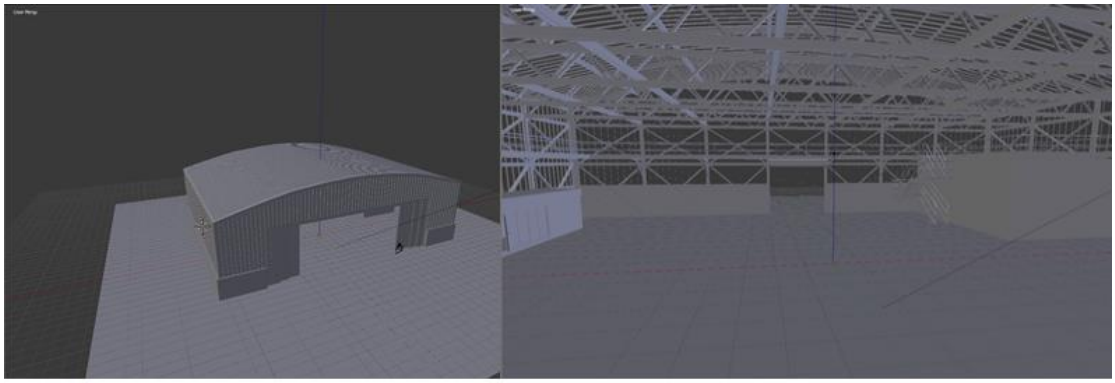

Figure 11 Hangar and the inside of our environment

5.2. Describing the Unity 3d Environment

5.2.1. Programming in unity

Unity as aforementioned is a game engine, which is used for more than just games, since it is easy to use and its libraries are updated constantly, making it very easy to use.

The programming language we mostly use is ***C#***.

The fundamental building blocks of unity:

- **GameObjects**

Every kind of content in Unity begins with a GameObject. Any object in your game is a GameObject: characters, lights, special effects, props– everything.

GameObjects can't do anything on their own. To actually become something, you need give a GameObject properties, which you do by adding Components.

- **Components**

Components define and control the behavior of GameObjects they are attached to.

Components examples are : Rigidbody , Scripts , Colliders , Colors.

- **Variables**

Components have any number of editable properties that can be tweaked via the Inspector window in the editor, and/or via script. In the above example, some properties of the light are range, color and intensity.

Unity's built-in Components are very versatile, but you will soon find you need to go beyond what they can provide to implement your own gameplay logic. Using scripts, you can implement your own game logic and behaviour by simply applying them to the game objects.

Your script Components will allow you to do many things: trigger game events, check for collisions, apply physics, respond to user input, and much, much more.

5.2.2. Using our virtual arm

5.2.2.1 Upgrading from previous version to the current one

Using knowledge from a previous thesis (Mourelatos 2017), we created a jointed arm for the purposes of the experiment, controlled by the signal from the three IMUs on the user's arm. This arm was created using simple capsule and sphere objects found(see Figure 12) in the Unity Game Engine, since our focus was directed more towards it being capable of accurately capturing the movements of the user than to it being aesthetically pleasing or realistic. Each

capsule represents a segment of the arm (palm, forearm, and upper arm), and each sphere one of the arm's joints (wrist, elbow and shoulder). The signal from each IMU controls the position and orientation of the arm's corresponding joint—the IMU mounted on the upper arm controls the shoulder, the one mounted on the forearm controls the elbow, and the one on the palm controls the wrist. The capsules and spheres are connected via parent-child object relations starting from the shoulder and moving down to the palm. This way the movement of the virtual arm corresponds to the movement of the user's arm in the real world—for example, if the user rotates their forearm about the elbow, the wrist and palm move as well, but if the user simple flexes or extends their wrist while keeping the rest of the arm immobile, the same movement will occur on the virtual arm.

Figure 12 Previously used Arduino Controlled Arm

In order to control the avatar's right hand we need a script that is able to read the Arduino's serial data. This was achieved by using the ***Oculus_motion Control*** script which is able to read the input from the 3 mpu's in the Arduino and serialize it. The data from the Arduino is basically the arm's rotation and it is expressed in **Quartenions**.The script reads that data, separates it with commas and the output is given as rotation coordinates to the humanoid's arm. The data is not read in the primarily function of unity which is ***Update(),*** since when the Arduino and the Oculus rift mask are connected together, Unity choses to read the data first from Oculus thus making the Arduino data lag. For this reason we used a thread that reads parallel το Oculus our serial data.

In order to control properly our humanoid we have to change the structure of its tree to match the previous version of our controlled arm. Thus we created 3 empty GameObjects :

- Shoulder
- Elbow

- Wrist

They are the objects that receive our data. The script controlling the arm is attached to Shoulder GameObject.

The abovementioned script (see Figure 13) enables to control the humanoid's arm as it was our own, following the user's physical movement and making it relatively realistic, with a latency that is acceptable and can be ignored.

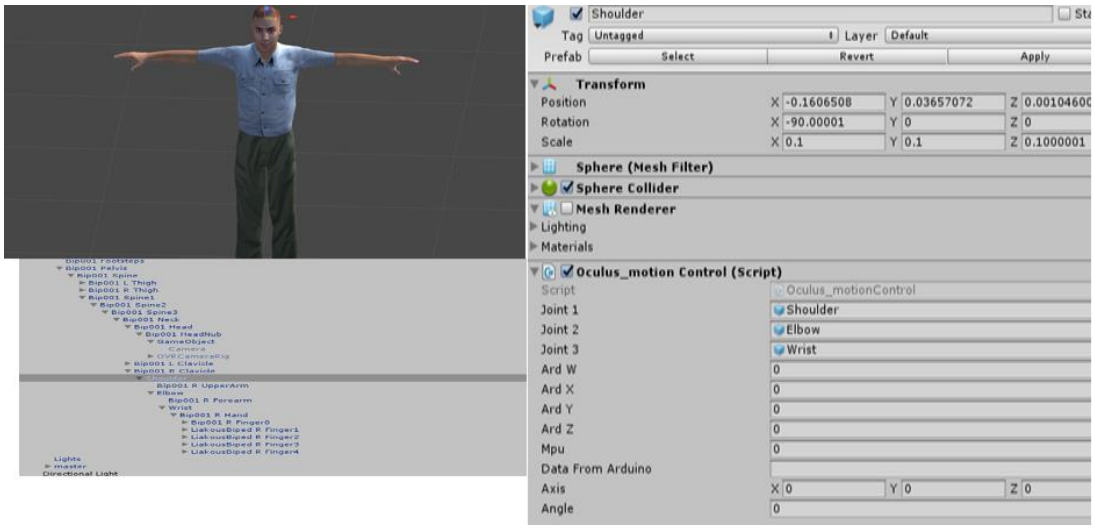

Figure 13 The Unity script that is able to control the Humanoid's aka User's arm

5.2.3. Describing the Setting

The setting consists of multiple objects including the Robotic arm, the balls, the conveyor belt and the basket(see Figure 14)

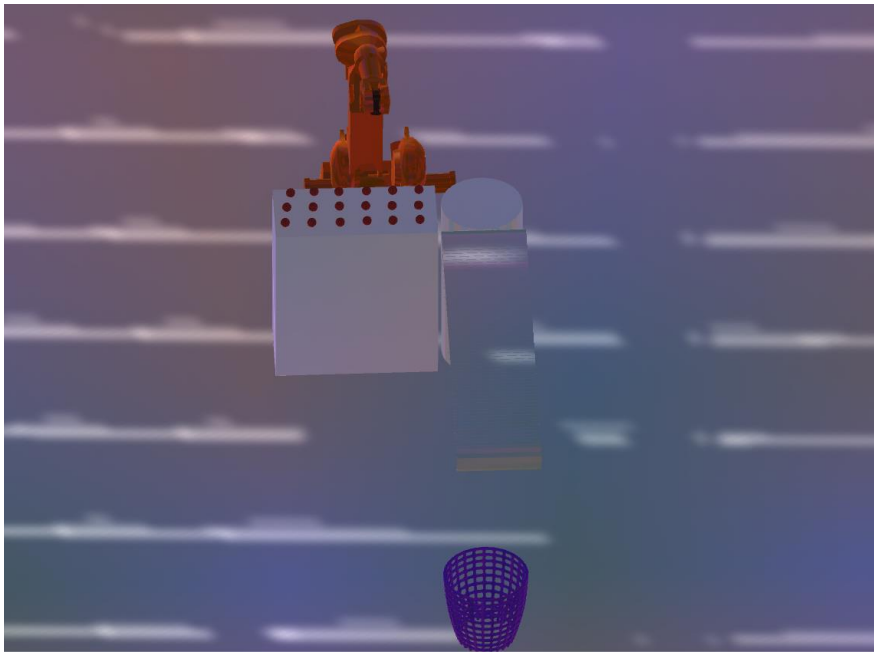

Figure 14 Whole setting

We are going to describe each element, its importance and last but not least the logic that makes everything work.

The default setting consists of these elements :

- Robot-Balls Interaction
- Human-Balls Interaction
- Robot-Human Interaction
- Balls-Conveyor belt Interaction
  Metrics

5.2.3.1 Robot-Balls Interaction

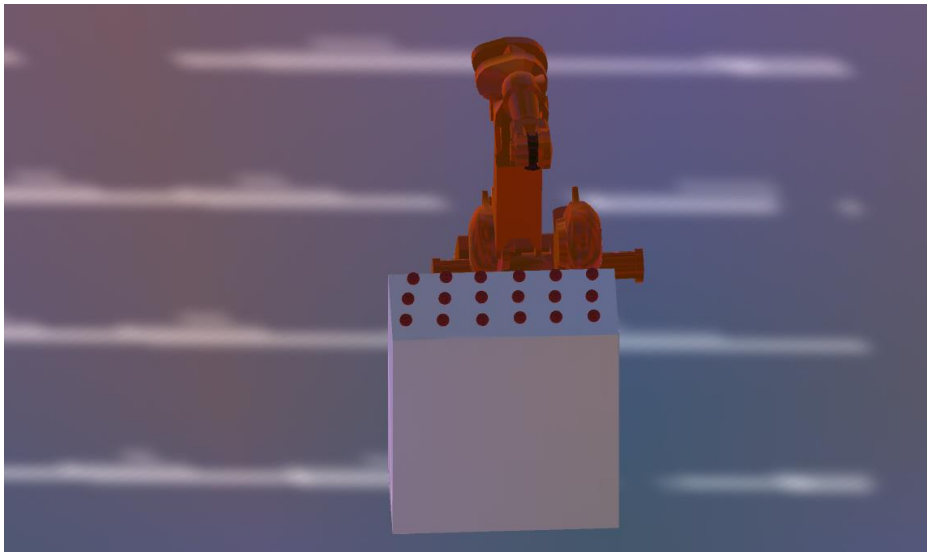

Figure 15 Focusing only on the balls and the robot

The first we were able to create is a logic behind the robot's animation, the robot has to be able to pick up the balls(see Figure 15) by a specific order and since it is a collaborative task the balls have to be close to the balls the Player has to pick.

In order to be easier for us, we set our balls as a prefab with a script attached to them.

The script is called ***Balls_Colliders_Grabbing_Scenario#,*** where # is the number of the scenario currently playing and its purpose is multifunctional.

First of all, each ball has a Rigidbody Component attached to it and a sphere collider.

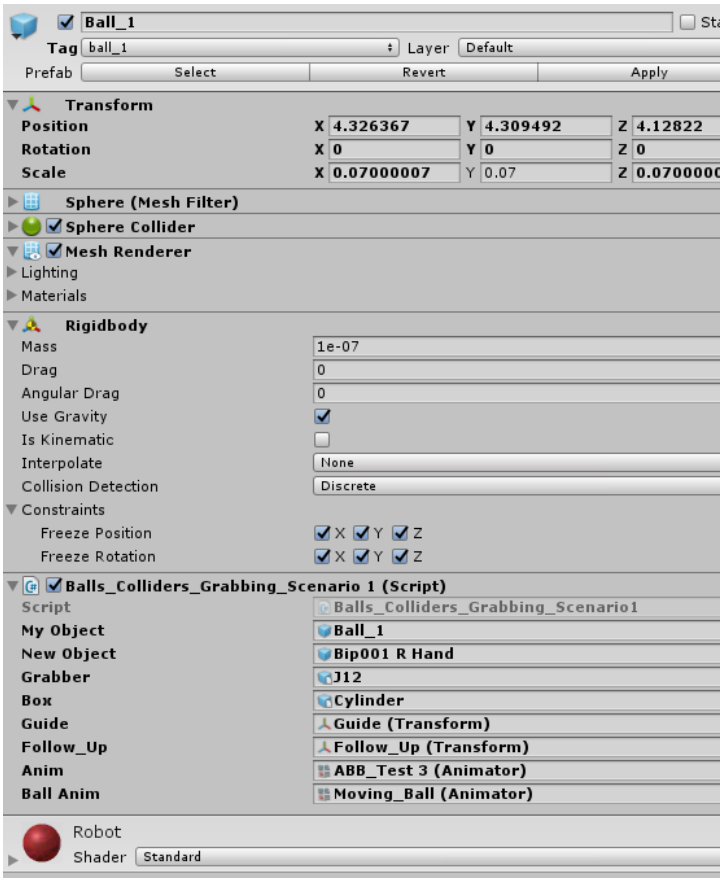

Figure 16 Scripts and Components attached to the ball; controlling most elements in the scenario

The Rigidbody Component and the Sphere Collider(see Figure 16,17) are the most important components for our interactions. The Collider in each component is indicated by the green shaped area.

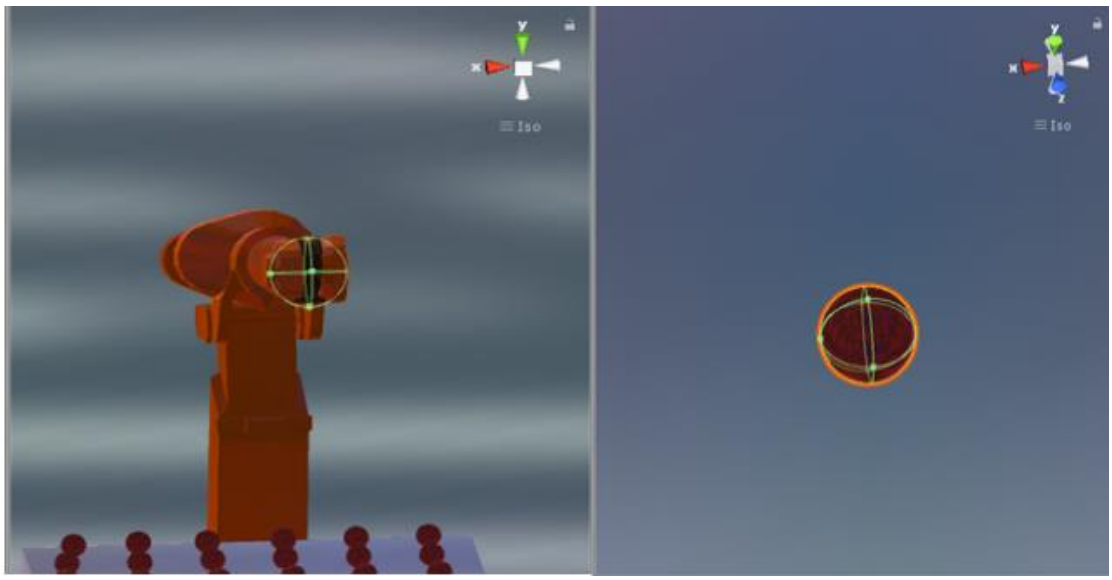

Figure 17 Robot and ball Colliders

We want the robot to be able to pick up each ball when moving downwards, so we also add a sphere collider to the robot's End effector.

The Rigidbody component is crucial for our objects because the function ***OnCollisionEnter(),*** in order to detect a collision needs the bodies that collide have a Rigidbody. The Rigidbody component have the disadvantage that it adds physics to our objects, thus it can fall through the table or slide through it, so in order to avoid this we freeze our object's position and rotation.

With this procedure and using the Function ***OnCollisionEnter()***, Unity detect when the bounds of the sphere collider of the robot touches the Sphere Collider of the ball and then the ball is attached to the robot's End effector by creating a Parent-Child relation between the two objects. The ball's position is carefully placed slightly below the robot in order not to pass through the robot avoid making it unrealistic to the user. As we can see from the previous image the ***My_Object***, Ball is attached to the grabber and follows its movement using the ***Follow_Up*** transform.

As a result the final product of the following logic, the robot when colliding with a ball (see Figure 18) and grabs it.:

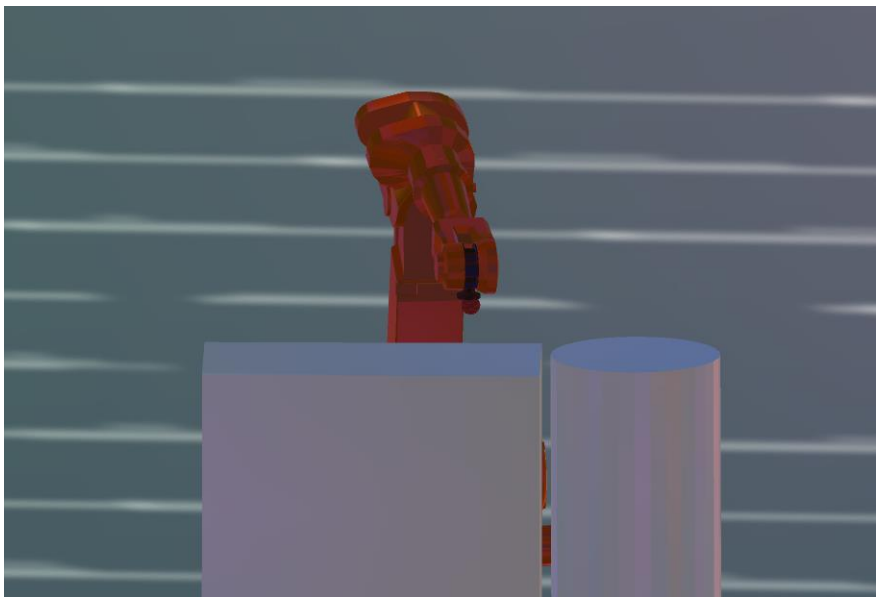

Figure 18 Robot picks up a Ball

In order for the robot to be able to take the different array of balls we need to put some logic to the robotic arm, so we attach to it the Animator Component(see Figure 19).

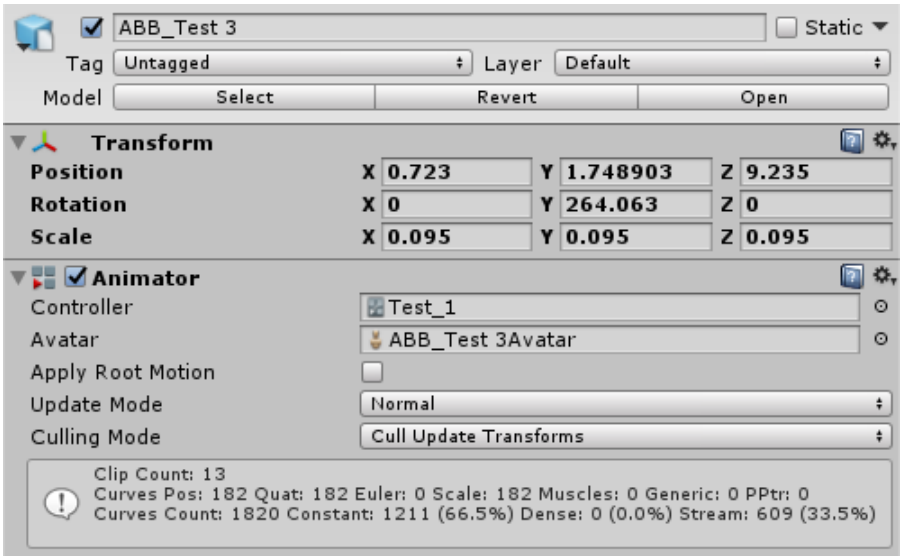

Figure 19 Animator Component; Logic behind the robot's movement

The logic behind the order of the balls is for the Player and the robot to be able to interact with our balls.

The balls that the robot is able to interact with are predetermined to us but it is not known to the Player interacting with it.

In the first two scenarios the order for the balls to be picked up are the following :

- 1
- 9
- 10
- 13
- 16
- 8
- 18
- 15
- 17

The rest 9 balls are left for the player to pick them, but also in a particular order.

So the animation logic for the balls is made in the animator Component adding each state (see Figure 20).

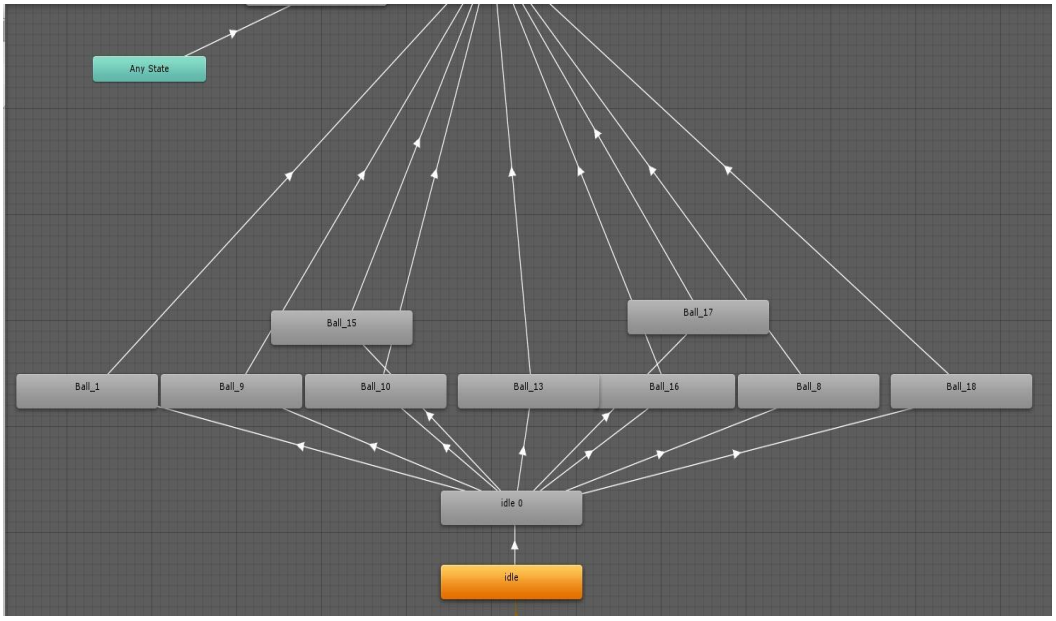

Figure 20 Animator Component Tree

The **Animator Component** is accessed both by the Scripts previously mentioned and it is attached to the balls and the Script that is attached to the End effector.

The robot when starting the scenario is in idle position, this state is the starting position and the stopping position of our robot, this is because we don't want the robot to start moving immediately when the scenario is starting, but we want it to start when we trigger an event, such as pressing a button and grabbing a ball.

5.2.3.2 Human-Balls Interaction

With the move of our right arm we can move the right arm of our humanoid in real time with no latency. We want to be able to pick up our balls when our hand is on them and when a left click is pressed. This is done by the previously mentioned script ***Balls_Colliders_Grabbing_Scenario#.*** We added a Box Collider in our Humanoid's hand and a Transform called Guide in front of our hand.

This time we didn't use the function ***OnCollisionEnter(),*** because this function is called the exact time the collision happens and the left click of the player must be instantaneous, this is very difficult even impossible to happen.

We need the player to be able to grab the ball when the collider intersect each other, in order to do that we use a condition that is called ballColider bounds intersect with handCollider, this is done by the following command :

- ***ballCollider.bounds.Intersects(handCollider.bounds) ;***

When a left click is pressed which is achieved by :

- ***Input.GetMouseButtonDown(0);***

The ball follows the hand's movement, according to a Parent-Child relation, similarly with the robot. The Parent-Child transform has a particular problem, the ball's scale, which is the Child is relative to the Parent's, which is the player's arm, so due to different sizes, it changes. In order to keep the original scale we need to rescale it, matching our starting scale.

Last but not least, we don't want our ball to pass through the player's hand and seem unphysical, thus we add a Guide transform in front of our hand which shows the position of our ball when is picked up by the player(see Figure 21).

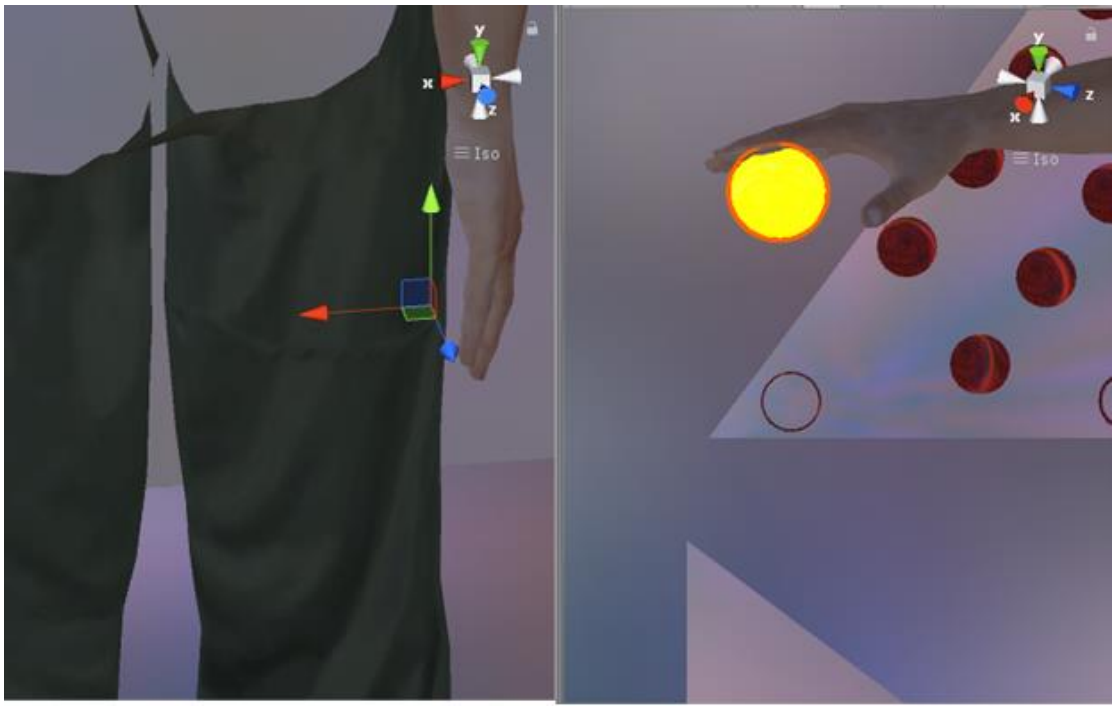

Figure 21 Player picking up a ball

The previous picture shows the position of the guide relative to the Player's hand and how the ball is attached to the player's arm when he picks it up.

In order to be a collaborate task, the Player and the robot have to pick different balls from our robot. The balls to be picked have different color, a bright yellow

color and it is indicated by a script called ***Color_Indicator*** and it is attached to the empty GameObject of Balls which is Parent of all our balls in our setting.

Each ball is a GameObject with a different tag, as a result we can easily find each ball. ***Color_Indicator*** script does exactly what is name say, it indicates for the player which ball is to be picked up next. The way it is done is by looking for GameObjects with specific tag and in order to be in a consecutive order the previously tagged objects needs to empty.

The player has to only grab the balls in the order that is indicated by the script in order to successfully complete the scenario.

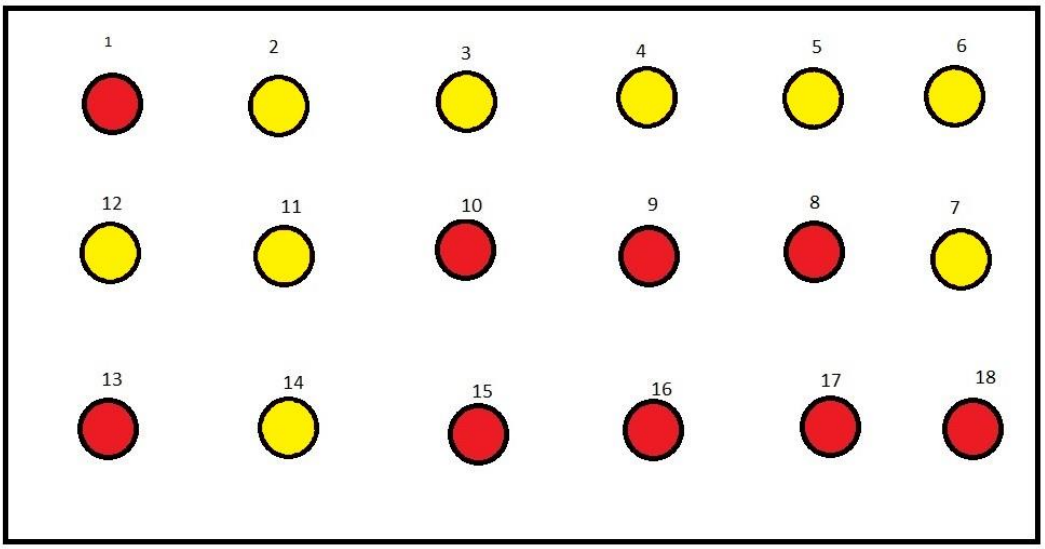

Figure 22 Yellow balls picked up by Player, Red are picked up by Robot

The Yellow balls are the balls to be picked up by the player while the Red balls by the Robot (see Figure 22). Last but not least, there is a delay between each ball that is highlighted in order not to confuse the Player with the immediate change.

The balls are strategically put in such a way, in order to mix the path of the Player and the robot, since if the order was not determined the player would pick the balls that would be more convenient or close to him/her and it would be very difficult for a collision to happen, since the robot will be easily ignored. After the first set of balls is finished, another set appears with a little delay in order to have more stable metrics in our scenario.

The second set of balls is pictured below with the player picking them up with different order than before, the same applies to the robot. (See Table 1)

Set 2

Player	Robot
3	11
15	16
2	10
8	9
5	12
14	7
13	17
6	18
4	2

Table 1 : The balls the robot and the player have to grab.

5.2.3.3. Robot-Human Interaction

Perhaps the most important state of our Thesis is the Robot-Human interaction, what will happen when the robot End effector and human arm collide in such a task.

We want to ensure that the robot will make a reaction when in touch with the human arm. So in the robotic arm we add 3 Colliders covering the whole arm, up until the back of the robot (see Figure 23).

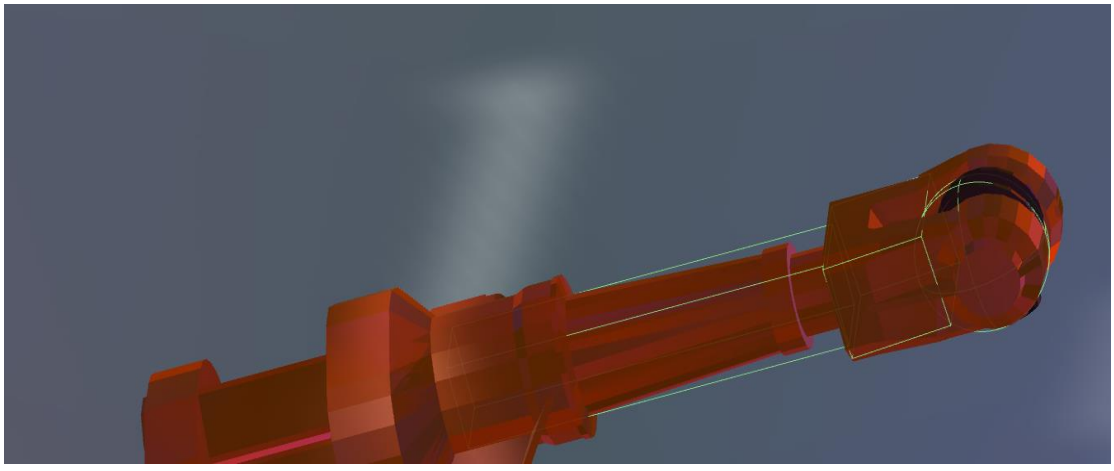

Figure 23 : Robot Colliders

We want for the Player to understand when he is touching our robot. So we need to give feedback when he does so. There are 3 types of impact that are used in our scenarios :

- Visual feedback (2 types)
- Audio feedback

The first visual feedback concerns the robot animation, when the player accidentally touches the robot, it jitters making it appear like it's being hit.

This is done in the Animator Component and when the GameObjects Hand and Player are colliding a trigger called ***"move***", is instantiated and the robot makes the jittery move and then stops, it then follows a logic procedure that is about whether is holding a ball or not.

When it is holding a ball and the robot return to its position and its first action is to put the ball intro place and continue grabbing the remaining balls.

The Any State states that wherever the robot is, if it is hit by the Player it has to change its course of action and follow the procedure described above.

The second visual feedback is a commonly used in videogames, to indicate that the Player is being hit, which is a camera shake.

The ***CameraShake_Scenario#*** script is attached to hand of the Humanoid.

When the collision happens, it triggers. It takes as input, the main camera's, in our situation, the two main camera's positions which is indicated by the Prefab ***OVRCameraRig***, then it moves the cameras inside a unit sphere by a random value, changing their positions only for a fraction of time, as a result when the shake value is 0, they return to their initial positions, which are saved before the shaking happens. Last but not least, a parameter is added called ***ShakeStrenght*** which indicate how much will the camera move inside the unit sphere. The ***Shake*** parameter is used to show how much time will the shaking last.

This is done by the command :

- ***cam_VR.transform.localPosition = originalPosition + (Random.insideUnitSphere * shake);***

Last but not least, we need an Audio feedback when we touch the robot. The audio is a warning sound that it triggers when the collision happens.

5.2.3.4. Balls-Conveyor belt Interaction

When the robot or the player grab a ball and drag it in the box which is side to the table the GameObject ball, is destroyed and it disappears, this is used as the condition for the ***Color_Indicator*** script in order to indicate the next ball, but a consecutive indication without really showing where the ball goes confuses the Player and it makes him that the ball he put into the Cylindrical shape just transferred and appeared as the next ball.

We want to give the user the impression that after putting his ball in the Cylindrical shape it is physically transferred somewhere.

We want to make the ball look like it slides throughout our conveyor belt, the way we do that is by adding an extra ball to our setting which is hidden from the Player's field of vision.

This ball is called ***Moving_Ball*** and it is along with our conveyor belt the final touches to the setting of the first and second scenarios.

The ball has the Animator Component attached to it and a simple animation logic, which is triggered when the Player or the robot puts a ball into the cylindrical shape.

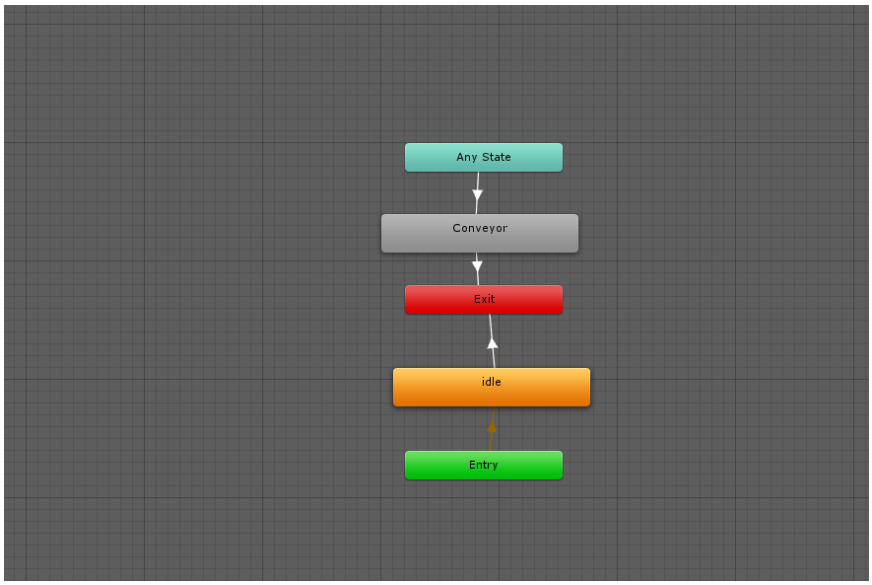

Figure 24 Animation tree of Conveyor Belt

The animator's logic is pretty simple since it only uses one trigger that fires as mentioned above(see Figure 24).

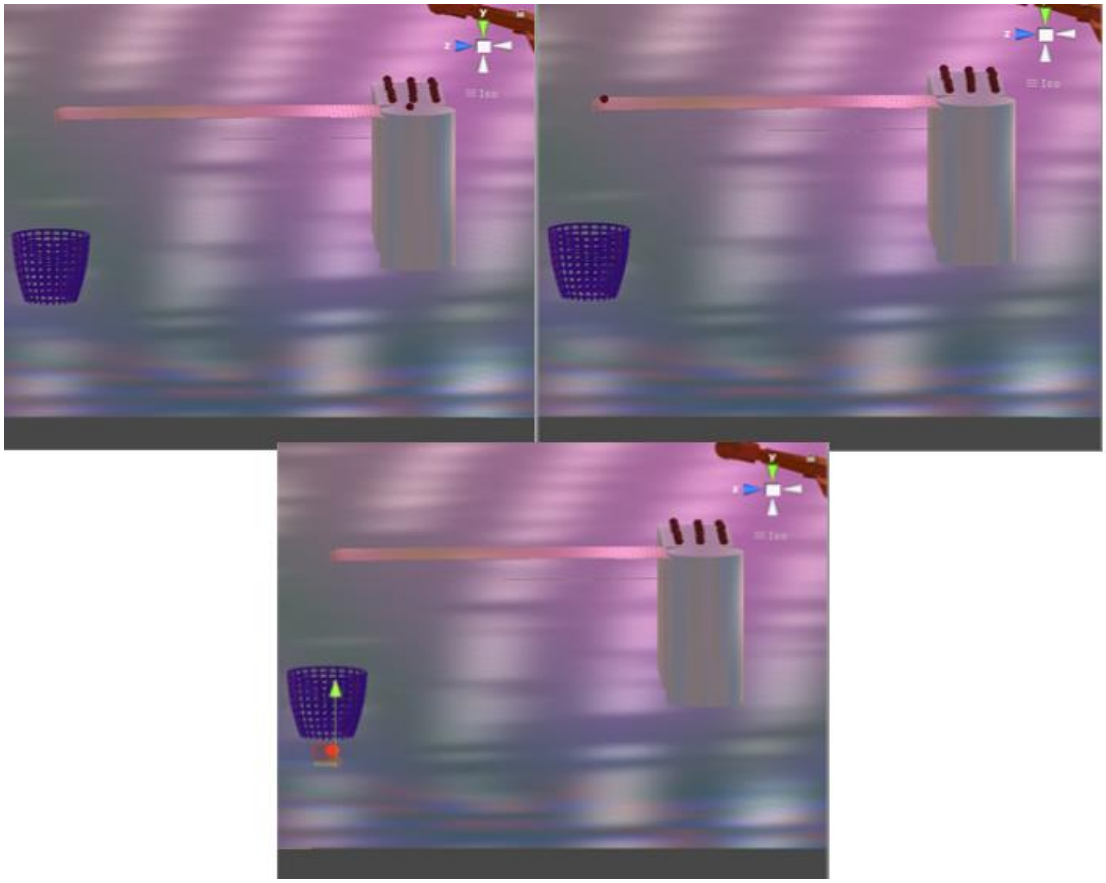

Figure 25 The animation of the ball going through the Conveyor belt

It is important to notice that the conveyor belt isn't moving neither does it have an animation, this is because the ball animation that shows that the ball is moving, it is easier to accomplish and it makes the player think that the conveyor belt is working.

The final position creates the illusion that the ball falls through the basket.

The abovementioned animation (see Figure 25) is miscellaneous for our environment but it is important because it created the illusion that a certain work is to be made, so it makes the player understand the environment as most immersive.

5.2.4.5. Metrics

In this thesis we want to count the number of times the Player collides with the robot and the time when the collision happens. This is done by creating a script called ***UI_Timer.*** We put this script in the ***GameManager*** along with a Text component.

In ***GameManager*** we also attach another script called ***Counting_Script_Scenario#,*** which continuously checks if the balls of the robot is less than the player's if so, the robot stats automatically and wait for the player to continue his/her task.

The purpose of this script is to start a timer when an action happens. As a result we are able to capture the time of the collision when it happens.

Another problem that we faced is the previously mentioned 3 Colliders on the robotic arm, due to the way Unity works and the fact that the colliders are too close to each other, each collision happening at the same time could be counted for more than once. As result the metrics would be ruined leading to a lot more collisions than it really happened.

In order to fix this problem we used a delay time in our script **CameraShake_Scenario#**, this delay time is expressed in frames and what it really does is that it stops the script from writing multiple times the collision number.

Last but not least, in order to keep track of our metrics we need a function that can write on a .txt file this is done by creating a new void called ***WriteToLogFile.***

6. Adjusting the setting

The player must feel immersed to the environment he is in, so we need to adjust its setting, adjusting the setting in order to feel more realistic to each subject was tested multiple times and through trial and error multiple errors were nullified.

The first problem that appeared during the development of the software is the user to be able to grab all the balls in the table. When we first used the Humanoid, the player with fully extended arm was not able to grab, so adjustments were made. First of all, we moved the setting; robot, balls, table, conveyor belt in a position that is relative to the Player's position, in order for the fully extended arm to reach the boundaries but when we first started the pilot test another immediate problem arose; the fact that even though the full extended arm seem to reach the end of table, due to Arduino errors and miscalculations, it would not be able to pick the balls in the third row from the player's perspective, so we confronted that problem by carefully elongated the Player's arm, while keeping the relations, trying to be realistic. This elongation resulted in a right arm which was bigger than the left one but since the player could not use the left one, he/she do not noticed any difference between them and as a result the elongation eliminated another complication that occurred during pilot testing the fact that the player would lean forward in order to grab the balls in the first row.

We wanted to make the Player feel the grabbing of the balls more natural, so in order to accomplish that, with the help of the colleague Antreas Mourelatos and Vaggelis Giagglisis. They managed to design a pushable button that is integrated with the whole Arduino circuit, the button is placed in the palm of the hand, thus the picking task seeming natural, in order for us to use that button, we needed to adjust the script controlling the humanoid arm from ***Oculus_motion_Controller*** to ***ThreadMotion_Control_3MPU'S*** in order to receive the button as input, as well as change codes where we utilized the mouse as input.

Another problem that appeared during pilot testing is the fact that the player is confused when he/she placed the ball and another one popped up immediately; thus confusing the player and makes him/her think that the ball teleported, in order to fix that error we simply added a little countdown timer when the next ball will appear about 200ms.

One of the most difficult problems to solve in gaming and in visual environments is the fact that the Arduino controlled arm of the humanoid passes through the table and through the balls, despite the colliders. This is due

to fact that it travels at such high speed so that it teleports between frames, so unity cannot detect the collisions and as a result it pass through our objects and thus making unity unable to render it, including our balls and table. The fact that the table is transparent to the movement of the Player and it doesn't block it from running through it is a huge problem, since we are inclined to find the best possible way around, the Player will choose instead of moving his/her virtual hand across the table to move it underneath it since it will be easier to complete the task, thus avoiding the robot all the time without any real interaction happening or referring to any real life situation.

Another problem is the Arduino calibration. The Arduino must first be calibrated in order to be placed into the human arm. The calibration is achieved using Lego bricks in order to keep its position stable for a few seconds (see Figure 26).

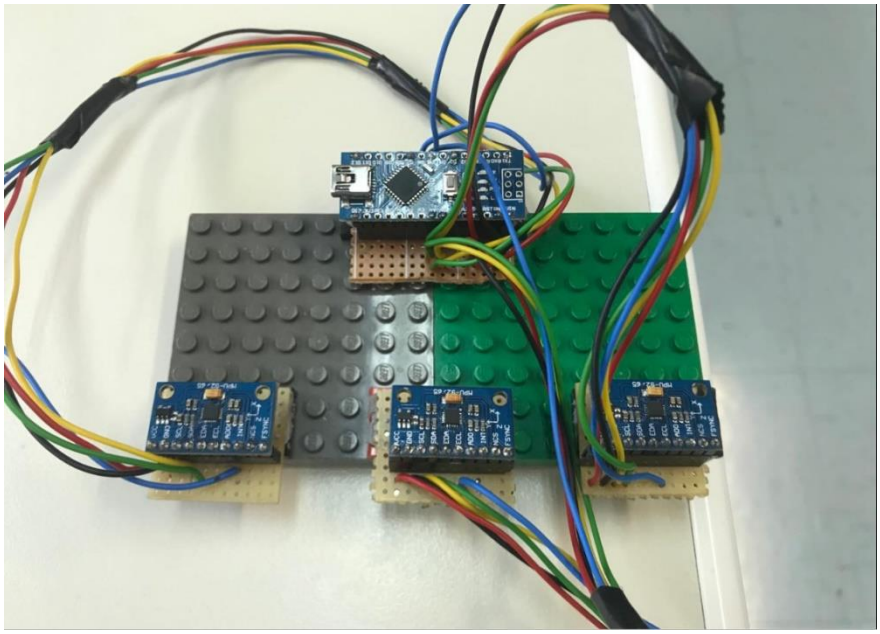

Figure 26 Calibrated Arduino

After calibrating to the right extent, it is mounted on the Player's arm.

Another problem of the calibration is that each time it has a deviation when it is mounted on a person's arm. So in order to solve both the technical problems of the table and the problem of it being too low or high in front of us we propose a solution.

Before starting the game we explain to our Player to sit in a chair with a desk in front of him. The physical table is used in order to counter the Arduino's calibration, we move the setting in the edit mode during the playing mode so that both the calibration and the fact that our hand pass through is fixed and the player cannot avoid the robot, just by moving his/her hand below the table.

The result of this adjustment is that the Player is moving his/her arm in the physical table and it also moves in the scenario in the virtual table, being unable to pass through it, thus making it realistic.

7. Scenarios

Our thesis consists of three scenarios and a configuration test, each one serving its purpose and will be described below. For each of these scenarios Arduino must be calibrated separetively and we need to adjust our setting relevant to the calibration.

7.1. Configuration Test

Before we start any of our experiments we need to show our Players how to interact with our environment, so a configuration test is made, much like a tutorial in every game.

The configuration test is the simpler than our 3 scenarios and it just contains the very basic features of our scenarios, that is the balls, the robot and the Cylindrical object that we put our balls.

This scene only contains 5 balls and when the game is starting the player must grab each one and put it in the Cylindrical object. The second use of this scenario is for the player to acknowledge how a collision sounds like and how it seems to the Player.

The player move his/hers physical hand and as a result also move his/hers virtual arm. He is instructed to grab the ball in front of him, there are only 5 balls, the Player grabs the ball by hovering his hand in the ball as described above and clicking the left click of his mouse. After grabbing the first yellow colored ball, he/she is told to put it in the cylindrical object as described above, this is done similarly by moving the ball in the ***Collider bounds*** of the Cylindrical

object and again left click is pressed. When he/she does this procedure the ball is destroyed and the animation with the ball moving in conveyor belt and dropping in the basket is playing to show that he correctly put the ball in the right place.

After repeating this procedure 5 times for each of the ball he/she then must touch the robot with his/her hand in order to feel and see how a collision sounds like and the response of the robot during the collision. This will help him/her understand how to know that a collision has been made.

7.2. Scenario 1 – Base

The first scenario of our thesis, is called base and it is the simplest scenario of the 3. In the first scenario, the robot's behavior, is dependent to the player's movements, thus being said that the robot will follow the pace of the Player without being able to surpass him. Now the setting is also different from the configuration and even from the initial setting, since many differences were made in order to be able to make an easy configurable set of experiments.

A new element is added to these scenarios, which is not described in the pre-test development scene, the element is a red button. The button's purpose is to restart the robot, when a collision happens, since we want the robot to follow the player's movement, the robot completely stops and so it is required for the player to press the button in order for it to continue grabbing the balls.

The idea behind these types of scenarios is to test the collaboration in a futuristic factory where humans and robotic arms are able to work together in a production line, so in these scenarios, each scenario's purpose is to for the player and the robot to finish the task in the least time possible, under the umbrella of safety.

Another new element of these scenarios is that a new game condition is used, we presume that the task is requires both the robot's ball and the player's in order to continue, then if the Player hits the robot and the robot stops, then the player's ball which was highlighted will be unmarked and as a result, the player would not be able to continue picking up the next ball.

In the first scenario, we assume the hypothesis that since there is no indication where the robot is going, the player would collide with it many more times, than the next two scenarios where the would be safety measures to prevent collisions from happening.

Another element of these scenarios is that two set of balls are used in order to make the experiment last more and have a more complete image about the collisions of the user.

7.3. Scenario 2 – Prevention

In this scenario all parameters remain the same as the base scenario but one crucial change is made; the robot now slows down by a significant amount when it is in close proximity with the Player.

This procedure is done with relative ease, thus by adding three more colliders that are scaled in bigger dimensions in order to create an area, which is supposed to set the bounds that slows the robot movement.

We expect in this scenario, that the collisions would diminish in relation with the first scenario.

7.4. Scenario 3 – Anticipation

The third scenario is another child of the base but in this scenario, we want to increase the safety of the working environment by indicating now to the player, where the robot is going next, an easy way to accomplish that is by indicating the next ball the robot is about to pick, the robot's ball is indicated by a color that resembles the robot, so that the user would be able to easy distinguish the differences between them.

In this scenario, we expect both that the Player would avoid the robot most of the times, in comparison to the other 2.

8. Testing the environment with users

In order to be able to make the abovementioned changes we ran a couple of tests with multiple users. The first couple of test which was pilot determined the changes.

When we ran the abovementioned scenarios through multiple users we received the following data(See Table 2,3).

Subject	**Collisions**		
	Base	**Prevention**	**Anticipation**
Maria	2	4	4
Petros	6	0	3
Antreas	7	1	0
Eva	10	5	4
Katerina	3	5	2
Giannis	6	4	5

Table 2 : Number of Collisions in the three scenarios

Subject	**Time**		
	Base	**Prevention**	**Anticipation**
Maria	1:53	2:11	1:51
Petros	1:59	1:58	1:58
Antreas	2:14	1:59	1:54
Eva	2:38	2:30	2:18
Katerina	2:11	2:26	2:04
Giannis	2:06	2:25	2:17

Table 3 : Total time when completing each scenario

The above sample which only consists of 5 subjects shows that the prediction about fewer collisions both in prevention and anticipation is valid, most of the users avoided the robot most of times in these scenarios.

9. Conclusions and further development

From the previous data it can be concluded the following(see table 4).

	Base	**Prevention**	**Anticipation**
Collision Number	5.57	3.2	3
Time (Min : Seconds)	2:10	2:15	2:07

Table 4 : Mean number of collisions and total mean time of the five players

The anticipation scenario proved the safest of the three as well as the scenario executed at the least possible time, making it thus the most efficient. Another conclusion that can be extracted from running the scenarios is that our environment was stable and did not impose any problem; all test subjects were able to run it smoothly without having any physical or software problem during gameplay. As a result, it can be concluded that scenarios like this can be implemented and as a result many more can be created to test a variety of different human – robot collaboration hypotheses relevant for future real world collaboration settings.

9.1 Future testing scenarios

The environment that we created for this thesis can be expanded for further use or even become the baseline for more advanced scenarios. A more advanced scenario is to create an environment where the player can use both arms in order to interact with objects, thus providing a more realistic simulation of real world human robot collaboration. The player would be able to control both arms and lean forward, greatly improving the player's feeling of immersion and subsequently the ecological validity of the experimental environment. The above development could form the baseline for more elaborated scenarios such as in a more complex assembly line where the human-robot co-operation will engage the full upper body of the player, but also whole body movement in space. Examples of such tasks are various assembly tasks in the automobile and aerospace manufacturing industries, where it is critical to combine human perceptual sensitivity and decision making with robot precision and load manipulating capacity.

Another possible way to upgrade the work is to upgrade the simulator environment with a software like VR robotics simulator, where many different types of predefined robotic arms and manufacturing processes can be simulated. In such an environment the user would able to interact with more

than one virtual object, thus creating a robust tool that would simulate with high immersion a real world manufacturing plant for inexpensive training and safety training for future employees.

References

Antoniou S., Rentzos L., Mavrikios D., Georgoulias K., Mourtzis D., Chryssolouris G. (2016) A Virtual Reality Application to Attract Young Talents to Manufacturing.

F. Caputo, A. Greco, E. D'Amato, I. Notaro, Spada S.(2016),On the use of Virtual Reality for Human Centered workplace design, Portuguese Conference on Fracture, PCF 2016, 10-12 February 2016, Paço de Arcos, Portugal.
G. C. Burdea, "Invited review: the synergy between virtual reality androbotics," IEEE Transactions on Robotics and Automation, vol. 15,no. 3, pp. 400–410, 1999.

Giuseppe R., Cristina B., Rosa B., Fabrizia M.,Azucena G.P., Soledad Q., Silvia S., Claudia R., Antonios D., Daniela V., Stefano T., and Andrea G. (2015) Presence-Inducing Media for Mental Health Applications.

Heidicker, P., Langbehn, E., & Steinicke, F. (2017, March). Influence of avatar appearance on presence in social VR. In 3D User Interfaces (3DUI), 2017 IEEE Symposium on (pp. 233-234). IEEE.

Hongyi L., Lihui W. (2017) An AR-based Worker Support System for Human-Robot Collaboration,27th International Conference on Flexible Automation and Intelligent Manufacturing, FAIM2017,27-30 June 2017, Modena, Italy.

Karakikes M. (2017). Development and Evaluation of a Wearable Motion Tracking System, to Support Hand-Tool Design, Diploma Thesis, National Technical University of Athens.

Kilteni, K., Bergstrom, I., & Slater, M. (2013). Drumming in immersive virtual reality: the body shapes the way we play. IEEE transactions on visualization and computer graphics, 19(4), 597-605.

Kilteni, K., Normand, J. M., Sanchez-Vives, M. V., & Slater, M. (2012). Extending body space in immersive virtual reality: a very long arm illusion. PloS one, 7(7), e40867.

Lange, B., Suma, E. A., Newman, B., Phan, T., Chang, C. Y., Rizzo, A., & Bolas, M. (2011, July). Leveraging unencumbered full body control of animated virtual characters for game-based rehabilitation. In International Conference on Virtual and Mixed Reality (pp. 243-252). Springer, Berlin, Heidelberg.

Liu O., Rakita D., Mutlu B., and Gleicher M.(2016) Understanding Human-Robot Interaction in Virtual Reality.

Luigi G., Marco S., Luigi P., Giuseppe D.G., Philipp K. (2016) Coupling of a redundant manipulator with a virtual reality environment to enhance human-robot cooperation, 10th CIRP Conference on Intelligent Computation in Manufacturing Engineering - CIRP ICME '16.

Luo, Z., Lim, C. K., Chen, I. M., & Yeo, S. H. (2011). A virtual reality system for arm and hand rehabilitation. Frontiers of Mechanical Engineering, 6(1), 23-32.

Matsas, E. & Vosniakos, GC. (2015). Design of a virtual reality training system for human–robot collaboration in manufacturing tasks. Int J Interact Des Manuf (2017) 11: 139.

Merians, A. S., Tunik, E., & Adamovich, S. V. (2009). Virtual reality to maximize function for hand and arm rehabilitation: exploration of neural mechanisms. Studies in health technology and informatics, 145, 109.

Osumi, M., Ichinose, A., Sumitani, M., Wake, N., Sano, Y., Yozu, A., & Morioka, S. (2017). Restoring movement representation and alleviating phantom limb pain 62 through short-term neurorehabilitation with a virtual reality system. European journal of pain, 21(1), 140-147.pp. 1878–1883.

R. Belousov, R. Chellali, and G. J. Clapworthy, “Virtual reality tools for internet robotics,” in Robotics and Automation, 2001. Proceeding 2001 ICRA. IEEE International Conference on, vol. 2. IEEE, 2001,

R. Safaric, S. Sinjur, B. Zalik, and R. M. Parkin, “Control of robot arm with virtual environment via the internet,” Proceedings of the IEEE, vol. 91, no. 3, pp. 422–429, 2003.

Ruckbert P., Wohlformm L.,Tracht K.(2017),Implementation of virtual reality systems for simulation of human-robot collaboration, 6th International Conference on Through-life Engineering Services,TESConf 2017,7-8 November,Bremen,Germany.

Slater, M., Spanlang, B., Sanchez-Vives, M. V., & Blanke, O. (2010). First person experience of body transfer in virtual reality. PloS one, 5(5), e10564.

Suto S., Forgo Z.,Ferenc T.(2016),Simulation Based Human Robot Co-working,10th International Conference Interdisciplinary in Engineering,INTER-ENG,2016.

Wittmann, F., Lambercy, O., Gonzenbach, R. R., van Raai, M. A., Höver, R., Held, J., ... & Gassert, R. (2015, August). Assessment-driven arm therapy at home using an IMU-based virtual reality system. In Rehabilitation Robotics (ICORR), 2015 IEEE International Conference on (pp. 707-712). IEEE.

X. Tang and H. Yamada,(2011)“Tele-operation construction robot controlsystem with virtual reality technology,” Procedia Engineering, vol. 15,pp. 1071–1076,2011

Codes Appendix

In this section we put the codes for the base scenario, the other two scenarios have the at most part the same coding but with a few small changes.

Balls_Colliders_Grabbing

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class Balls_Colliders_Grabbing_Base_ : MonoBehaviour {

    //bool to check if the player holds a ball
    public static bool playerIsHolding = false;
    //objects to be used
    public GameObject m_MyObject, m_NewObject,grabber,box;
    //colliders for the collisions
    Collider m_Collider, m_Collider2,m_Collider3,box_collider;
    //grabbing transforms
    public Transform guide,Follow_Up;
    Transform m_transform;
    public Animator anim,ballAnim;
    Vector3 originalPos;
    //color of the next object
    Color32 nextObject = new Color32(7,201,25,255);
    public static float referencingTime;
    //colliders that slow down the robot animation
    //public Collider J10slowCollider,J11slowCollider,J12slowC
ollider;
    //counts the balls to be grabbed by the player in order to
 check the robot
    public static bool beAbleToPickUpNextBall = true;
    //οριζουμε τα 3 κυρια objects για τα collisions και για κα
θε object βρισκουμε το collider του ετσι ώστε να γίνονται grab
 καθε φορα


    void Start()
    {
        //anim bool for the button we need that to be true, th
e button is pressed when starting
        anim.SetBool ("Restart", true);
        anim.SetBool ("Start", false);

        //Check that the first GameObject exists in the Inspec
tor and fetch the Collider
        if (m_MyObject != null)
            m_Collider = m_MyObject.GetComponent<Collider> ();
        m_transform = m_MyObject.GetComponent<Transform> ();
```

```
        originalPos = m_transform.position;

        //Check that the second GameObject exists in the Inspe
ctor and fetch the Collider
        if (m_NewObject != null)
            m_Collider2 = m_NewObject.GetComponent<Collider> (
);

        //checks if the robot component is empty or not
        if (grabber != null)
            m_Collider3 = grabber.GetComponent<Collider> ();

        if (box != null)
            box_collider = box.GetComponent<Collider> ();

    }

    void OnCollisionEnter(Collision col)
    {
        if(col.gameObject.tag == "box" && m_Collider.GetCompon
ent<Renderer>().material.color != nextObject)
        {
            BallDestroyed ();
        }

    }

    void Update()
    {
        if (referencingTime > 0) {
            referencingTime -= Time.deltaTime;
            if (referencingTime < 0) {
                referencingTime = 0;
            }
        }
        //determines the animation speed
        anim.speed = 1.3f;

        //If the first GameObject's Bounds enters the second G
ameObject's Bounds, output the following and the ball must hav
e the Yellow color in order for the user to grab it
        if (m_Collider.bounds.Intersects (m_Collider2.bounds)
&& ThreadMotCont3MPUS.buttonState == 0 && playerIsHolding == f
alse && m_Collider.GetComponent<Renderer> ().material.color ==
 nextObject )
        {
            PlayerBallGrabbing ();

        } else if ((m_Collider.bounds.Intersects (m_Collider3.
```

```
bounds)) && m_Collider.GetComponent<Renderer> ().material.colo
r != nextObject )
        {
            RobotGrabsBall ();
        }

        //Player puts the ball in the desired position, meanin
g the Cylindrical object
        else if (m_Collider.bounds.Intersects (box_collider.bo
unds) && ThreadMotCont3MPUS.buttonState == 0 && playerIsHoldin
g == true)
        {
            PlayerLetsBallDown ();
        }

        //when the player finishes the robot grabs all remaini
ng balls
        if (CameraShake_Base_.Test1 == true) {
            if (GameObject.FindGameObjectWithTag ("ball_7") ==
 null) {
                anim.SetBool ("Start", true);
            } else if (GameObject.FindGameObjectWithTag ("ball
_17") == null) {
                anim.SetBool ("Start", false);
            }
        }

        //test 2
        else if (CameraShake_Base_.Test1 == false) {
            if (GameObject.FindGameObjectWithTag ("ball_4") ==
 null) {
                anim.SetBool ("Start", true);
            } else if (GameObject.FindGameObjectWithTag ("ball
_1") == null) {
                anim.SetBool ("Start", false);
            }
        }

        //ball returns to original position when a click is pr
essed
        /*if (Input.GetMouseButtonDown (1)) {
            gameObject.transform.parent = null;
            m_transform.position = originalPos;
            isHolding = false;
        }*/

    }
    //Triggers when the robot touches the box and destroys the
 ball
```

```
    void BallDestroyed()
    {

        Destroy (gameObject);

        //triggers the robot-ball animation
        anim.SetBool ("holdingBall", false);

        // triggers the conveyor belt animation
        ballAnim.SetTrigger ("move");

    }

    //Triggers when Player touches the ball and as a result gr
abs it
    void PlayerBallGrabbing()
    {

        anim.SetBool ("Start", true);

        referencingTime = 30f;

        //Player grabs the ball
        transform.SetParent (m_NewObject.transform, true);
        transform.localScale = new Vector3 (0.07f, 0.07f, 0.07
f);
        m_transform.position = guide.position;
        Destroy (GetComponent<Rigidbody> ());

        //isHolding is for checking if the player has a ball c
an grab only one ball
        playerIsHolding = true;

    }

    //The robot grabs a ball
    void RobotGrabsBall()
    {
        transform.SetParent (grabber.transform, true);
        m_transform.position = Follow_Up.position;
        anim.SetBool ("holdingBall", true);
    }

    //Player destroys the ball by putting it in the Cylindrica
l object
    void PlayerLetsBallDown()
    {
        Destroy (m_MyObject);
        playerIsHolding = false;
```

```
        ballAnim.SetTrigger ("move");
    }

}
```

Camera_Shake_Base

```
using UnityEngine;
using System.Collections;
using UnityEngine.UI;
using System.IO;

public class CameraShake_Base_ : MonoBehaviour {

    float shakeStrength = 0.01f;
    float shake = 0f;
    public static int colCounter = 0;
    //the second set of balls which is inactive
    public GameObject ballsTest2;
    public Vector3 originalPosition;
    public Animator anim;
    public OVRCameraRig cam_VR;
    public Text timer;
    //this bool is for proceeding to each test
    bool robotFinish1 = false,humanFinish1 = false,robotFinish
2 = false,humanFinish2 = false;
    //this stops the player from colliding before grabbing a b
all
    public static bool start = false;
    public AudioClip robotCol;
    //dealy time to stop multiple collisions to be written in
the log
    public static float delayTime = 0;
    //Test 1 is for starting the next set of balls
    //collisionStart detects the collision and reminds the oth
er scripts
    public static bool Test1 = true,collisionStart = false;
    public Collider forearmCollider,j10col,j11col,j12col;



    void Start()
    {
        //originalPosition = cam_VR.transform.position;
        GetComponent<AudioSource>().playOnAwake = false;
        GetComponent<AudioSource>().clip = robotCol;
        anim.SetInteger ("Counter", 0);
    }
```

```
    void Update()
    {

        if (delayTime > 0) {
            delayTime -= Time.deltaTime;
            if (delayTime < 0) {
                delayTime = 0;
            }
        }
        cam_VR.transform.localPosition = originalPosition + (R
andom.insideUnitSphere * shake);
        shake = Mathf.MoveTowards (shake, 0, Time.deltaTime *
shakeStrength);
        if (shake == 0) {
            cam_VR.transform.localPosition = originalPosition;
        }

        //timer when ending each task
        if (GameObject.FindGameObjectWithTag ("ball_17") == nu
ll && Test1 == true)
        {
            if (robotFinish1 == false) {
                WriteToLogFile ("Robot Finishes First set at"
+ " " + timer.text.ToString ());
                robotFinish1 = true;
            }
        }
        //timer when second task is finished
        else if (GameObject.FindGameObjectWithTag ("ball_1") =
= null && Test1 == false)
        {
            if (robotFinish2 == false) {
                WriteToLogFile ("Robot Finishes Second set at"
 + " " + timer.text.ToString ());
                robotFinish2 = true;
                start = false;
                anim.SetInteger ("Counter", 18);
                anim.SetBool ("Start", false);
            }
        }
        //same here
        if (GameObject.FindGameObjectWithTag ("ball_7") == nul
l && Test1 == true)
        {
            if (humanFinish1 == false)
            {
                WriteToLogFile ("Player Finishes First set at"
```

```
 + " " + timer.text.ToString ());
                humanFinish1 = true;
            }
        }
        //same here
        else if (GameObject.FindGameObjectWithTag ("ball_4") =
= null && Test1 == false)
        {
            if (humanFinish2 == false) {
                WriteToLogFile ("Player Finishes Second set at
" + " " + timer.text.ToString ());
                WriteToLogFile("Robot remaining balls are " +
(9 - Counting_Script_Base_.ballsRobot).ToString());
                humanFinish2 = true;
            }
        }

        if (GameObject.FindGameObjectWithTag ("ball_7") == nul
l && GameObject.FindGameObjectWithTag ("ball_17") == null && T
est1 == true)
        {

            WriteToLogFile ("Next set of balls starting " + ti
mer.text.ToString ());
            ballsTest2.SetActive (true);
            Test1 = false;
            anim.SetInteger ("Counter", 9);
            anim.SetBool ("Start", false);


        }

        if (forearmCollider.bounds.Intersects (j10col.bounds)
&& delayTime == 0f && start == true) {
            HumanRobotCollision ();
        }else if (forearmCollider.bounds.Intersects (j11col.bo
unds )&& delayTime == 0f && start == true) {
            HumanRobotCollision ();
        }else if (forearmCollider.bounds.Intersects (j12col.bo
unds) && delayTime == 0f && start == true) {
            HumanRobotCollision ();
        }
        //The input comments are used in order to test new sce
nario elements if they are working or not, don't erase them
        /*if(Input.GetKeyDown(KeyCode.Q))
        {
            shake = shakeStrength;
        }*/
```

```
        /*if (Input.GetKeyDown (KeyCode.Space))
        {
            start = true;
            WriteToLogFile("Robot remaining balls are " + (9 -
 Counting_Script_Base_.ballsRobot).ToString());
        }*/

        /*if (Input.GetKeyDown (KeyCode.T))
        {
            Debug.Log ("CollisionNumber" + " " + colCounter +
" " + "time" + " " + timer.text);
            WriteToLogFile ("CollisionNumber" + " " + colCount
er.ToString() + " " + "time" + " " + timer.text.ToString());
        }*/

    }

    void OnCollisionEnter(Collision other)
    {

        if ((other.gameObject.tag == "End Effector" || other.g
ameObject.tag == "Robot") && start == true && delayTime == 0f)
{
            HumanRobotCollision ();
        }
        else if (other.gameObject.tag == "ball_2" && Test1 ==
true) {
            start = true;
            WriteToLogFile ("New Player : Base" + " " + System
.DateTime.Now.ToString ("G"));
        }


    }

    void  WriteToLogFile(string message)
    {
        try{
            using (System.IO.StreamWriter logFile = new System
.IO.StreamWriter(@"C:\Users\MaSys\Desktop\Vr data backup\14-
06-2018\Giannos\DataFileBase.txt", true))

            {
                logFile.WriteLine(message);
            }
            using (System.IO.StreamWriter logFile2 = new Syste
m.IO.StreamWriter(@"C:\Users\MaSys\Desktop\Vr data backup\14-
06-2018\Giannos\DataFileBase.txt", true))
```

```
            {
                logFile2.WriteLine(message);
            }
        }catch{
            Debug.LogError("Change Path, data is not written,
find the path VR data and the Back Up");
        }

    }

    //self explanatory
    void HumanRobotCollision()
    {
        //triggers the shake animation
        anim.SetTrigger ("move_back");
        shake = shakeStrength;
        colCounter++;
        delayTime = 0.8f;

        //Debug.Log ("CollisionNumber" + " " + colCounter + "
" + "time" + " " + timer.text);
        WriteToLogFile ("CollisionNumber" + " " + colCounter.T
oString () + " " + "time" + " " + timer.text.ToString ());

        //sets Animation to starting mode and it is needed to
press the button
        GetComponent<AudioSource> ().Play ();
        anim.SetBool ("Restart", false);

        //stops the balls from turning yellow
        Balls_Colliders_Grabbing_Base_.beAbleToPickUpNextBall
= false;

        //checks the collision
        CameraShake_Base_.collisionStart = true;
    }
}
```

Button_Press_Base

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;


public class Button_Press_Base_ : MonoBehaviour {

    public AudioClip buttonPress;
```

```
    public Collider handCol,buttonCol;
    Color redColor;
    Color brightRed = new Color(255f,0f,0f,255f);
    public Animator anim;

    // Use this for initialization
    void Start () {

        GetComponent<AudioSource>().playOnAwake = false;
        GetComponent<AudioSource>().clip = buttonPress;
        redColor = GetComponent<Renderer> ().material.color;
        GetComponent<Renderer> ().material.color = redColor;

    }

    void Update()
    {
        if (handCol.bounds.Intersects(buttonCol.bounds) && Thr
eadMotCont3MPUS.buttonState == 0 && Balls_Colliders_Grabbing_B
ase_.playerIsHolding == false && GetComponent<Renderer>().mate
rial.color == brightRed)
        {

            //animation parameters change when pushing the but
ton
            anim.SetBool ("Restart", true);

            // sees if the robot has a ball and automatically
starts
            if (anim.GetBool ("holdingBall") == true) {
                anim.SetBool ("Start", true);
            }

            GetComponent<AudioSource> ().Play ();
            GetComponent<Renderer> ().material.color = redColo
r;

            //set the conditions when the button is pressed
            //condition1
            Balls_Colliders_Grabbing_Base_.beAbleToPickUpNextB
all = true;

            //condition2
            CameraShake_Base_.collisionStart = false;


        }
        else if (CameraShake_Base_.collisionStart == true && C
ameraShake_Base_.start == true)
```

```
        {
            GetComponent<Renderer> ().material.color = brightR
ed;
        }
    }
}
```

Counting_Script_Base

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class Counting_Script_Base_ : MonoBehaviour {


    public Animator anim;
    public static int ballsRobot=0,ballsPlayer=0;

    // Update is called once per frame
    void Update () {

        if (ballsRobot < ballsPlayer ) {
                anim.SetBool ("Start", true);
        } else if (ballsRobot >= ballsPlayer && Balls_Collider
s_Grabbing_Base_.referencingTime == 0 ) {
                anim.SetBool ("Start", false);
            }




        if ( CameraShake_Base_.Test1 == true )
        {

            if (GameObject.FindGameObjectWithTag ("ball_17") =
= null) {
                ballsRobot = 9;
                anim.SetInteger ("Counter", 9);
            } else if (GameObject.FindGameObjectWithTag ("ball
_15") == null) {
                ballsRobot = 8;
                anim.SetInteger ("Counter", 8);
            } else if (GameObject.FindGameObjectWithTag ("ball
_18") == null) {
                ballsRobot = 7;
                anim.SetInteger ("Counter", 7);
            } else if (GameObject.FindGameObjectWithTag ("ball
```

```
_8") == null) {
                ballsRobot = 6;
                anim.SetInteger ("Counter", 6);
            } else if (GameObject.FindGameObjectWithTag ("ball
_16") == null) {
                ballsRobot = 5;
                anim.SetInteger ("Counter", 5);
            } else if (GameObject.FindGameObjectWithTag ("ball
_13") == null) {
                ballsRobot = 4;
                anim.SetInteger ("Counter", 4);
            } else if (GameObject.FindGameObjectWithTag ("ball
_10") == null) {
                ballsRobot = 3;
                anim.SetInteger ("Counter", 3);
            } else if (GameObject.FindGameObjectWithTag ("ball
_9") == null) {
                ballsRobot = 2;
                anim.SetInteger ("Counter", 2);
            } else if (GameObject.FindGameObjectWithTag ("ball
_1") == null) {
                ballsRobot = 1;
                anim.SetInteger ("Counter", 1);
            }


            if (GameObject.FindGameObjectWithTag ("ball_7") ==
 null) {
                ballsPlayer = 9;
            } else if (GameObject.FindGameObjectWithTag ("ball
_14") == null) {
                ballsPlayer = 8;
            } else if (GameObject.FindGameObjectWithTag ("ball
_5") == null) {
                ballsPlayer = 7;
            } else if (GameObject.FindGameObjectWithTag ("ball
_6") == null) {
                ballsPlayer = 6;
            } else if (GameObject.FindGameObjectWithTag ("ball
_3") == null) {
                ballsPlayer = 5;
            } else if (GameObject.FindGameObjectWithTag ("ball
_12") == null) {
                ballsPlayer = 4;
            } else if (GameObject.FindGameObjectWithTag ("ball
_11") == null) {
                ballsPlayer = 3;
            } else if (GameObject.FindGameObjectWithTag ("ball
_4") == null) {
```

```
                ballsPlayer = 2;
            } else if (GameObject.FindGameObjectWithTag ("ball
_2") == null) {
                ballsPlayer = 1;
                CameraShake_Base_.start = true;
            }
        }
        else if( CameraShake_Base_.Test1 == false)
        {
            ballsRobot = 0;
            ballsPlayer = 0;

            if (GameObject.FindGameObjectWithTag ("ball_1") ==
 null) {
                ballsRobot = 9;
                anim.SetInteger ("Counter", 18);
            } else if (GameObject.FindGameObjectWithTag ("ball
_18") == null) {
                ballsRobot = 8;
                anim.SetInteger ("Counter", 17);
            } else if (GameObject.FindGameObjectWithTag ("ball
_17") == null) {
                ballsRobot = 7;
                anim.SetInteger ("Counter", 16);
            } else if (GameObject.FindGameObjectWithTag ("ball
_7") == null) {
                ballsRobot = 6;
                anim.SetInteger ("Counter", 15);
            } else if (GameObject.FindGameObjectWithTag ("ball
_12") == null) {
                ballsRobot = 5;
                anim.SetInteger ("Counter", 14);
            } else if (GameObject.FindGameObjectWithTag ("ball
_9") == null) {
                ballsRobot = 4;
                anim.SetInteger ("Counter", 13);
            } else if (GameObject.FindGameObjectWithTag ("ball
_10") == null) {
                ballsRobot = 3;
                anim.SetInteger ("Counter", 12);
            } else if (GameObject.FindGameObjectWithTag ("ball
_16") == null) {
                ballsRobot = 2;
                anim.SetInteger ("Counter", 11);
            } else if (GameObject.FindGameObjectWithTag ("ball
_11") == null) {
                ballsRobot = 1;
                anim.SetInteger ("Counter", 10);
            }
```

```
            if (GameObject.FindGameObjectWithTag ("ball_4") ==
 null) {
                ballsPlayer = 9;
            } else if (GameObject.FindGameObjectWithTag ("ball
_6") == null) {
                ballsPlayer = 8;
            } else if (GameObject.FindGameObjectWithTag ("ball
_13") == null) {
                ballsPlayer = 7;
            } else if (GameObject.FindGameObjectWithTag ("ball
_14") == null) {
                ballsPlayer = 6;
            } else if (GameObject.FindGameObjectWithTag ("ball
_5") == null) {
                ballsPlayer = 5;
            } else if (GameObject.FindGameObjectWithTag ("ball
_8") == null) {
                ballsPlayer = 4;
            } else if (GameObject.FindGameObjectWithTag ("ball
_2") == null) {
                ballsPlayer = 3;
            } else if (GameObject.FindGameObjectWithTag ("ball
_15") == null) {
                ballsPlayer = 2;
            } else if (GameObject.FindGameObjectWithTag ("ball
_3") == null) {
                ballsPlayer = 1;
            }
        }
   }
}
```

Color_Indicator_Base

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class Color_Indicator_Base_ : MonoBehaviour {

    Renderer[] ballRenderer;
    Color previousColor;
    Color32 nextObject = new Color32(7,201,25,255);
    float delayTime = 0;

    // Use this for initialization
```

```
    void Start () {
        ballRenderer = GetComponentsInChildren<Renderer>();
        previousColor = GetComponentInChildren<Renderer> ().ma
terial.color;
    }

    void Update()
    {
        if (delayTime > 0) {
            delayTime -= Time.deltaTime;
            if (delayTime < 0) {
                delayTime = 0;
            }
        }
        try {
            if (GameObject.FindGameObjectWithTag ("ball_2") !=
 null && delayTime == 0) {
                ballRenderer [1].material.color = nextObject;
                delayTime = 0.65f;

            }
            if(Balls_Colliders_Grabbing_Base_.beAbleToPickUpNe
xtBall == true)
            {
                if (GameObject.FindGameObjectWithTag ("ball_4"
) != null && GameObject.FindGameObjectWithTag ("ball_2") == nu
ll && delayTime == 0 ) {
                    ballRenderer [3].material.color = nextObje
ct;
                    delayTime = 0.65f;
                }
                else if (GameObject.FindGameObjectWithTag ("ba
ll_11") != null && GameObject.FindGameObjectWithTag ("ball_4")
 == null && delayTime == 0) {
                    ballRenderer [10].material.color = nextObj
ect;
                    delayTime = 0.65f;
                }
                else if (GameObject.FindGameObjectWithTag ("ba
ll_12") != null && GameObject.FindGameObjectWithTag ("ball_11"
) == null && delayTime == 0) {
                    ballRenderer [11].material.color = nextObj
ect;
                    delayTime = 0.65f;
                }
                else if (GameObject.FindGameObjectWithTag ("ba
ll_3") != null && GameObject.FindGameObjectWithTag ("ball_12")
 == null && delayTime == 0) {
                    ballRenderer [2].material.color = nextObje
```

```
ct;
                    delayTime = 0.65f;
                }
                else if (GameObject.FindGameObjectWithTag ("ba
ll_6") != null && GameObject.FindGameObjectWithTag ("ball_3")
== null && delayTime == 0) {
                    ballRenderer [5].material.color = nextObje
ct;
                    delayTime = 0.65f;
                }
                else if (GameObject.FindGameObjectWithTag ("ba
ll_5") != null && GameObject.FindGameObjectWithTag ("ball_6")
== null && delayTime == 0) {
                    ballRenderer [4].material.color = nextObje
ct;
                    delayTime = 0.65f;
                }
                else if (GameObject.FindGameObjectWithTag ("ba
ll_14") != null && GameObject.FindGameObjectWithTag ("ball_5")
 == null && delayTime == 0) {
                    ballRenderer [13].material.color = nextObj
ect;
                    delayTime = 0.65f;
                }
                else if (GameObject.FindGameObjectWithTag ("ba
ll_7") != null && GameObject.FindGameObjectWithTag ("ball_14")
 == null && delayTime == 0) {
                    ballRenderer [6].material.color = nextObje
ct;
                    delayTime = 0.65f;
                }
            }
            else if(Balls_Colliders_Grabbing_Base_.beAbleToPic
kUpNextBall == false)
            {
                if(Balls_Colliders_Grabbing_Base_.playerIsHold
ing == false)
                {

                    if (GameObject.FindGameObjectWithTag ("bal
l_4") != null && GameObject.FindGameObjectWithTag ("ball_2") =
= null && delayTime == 0 ) {
                        ballRenderer [3].material.color = prev
iousColor;
                    }
                    else if (GameObject.FindGameObjectWithTag
("ball_11") != null && GameObject.FindGameObjectWithTag ("ball
_4") == null && delayTime == 0) {
                        ballRenderer [10].material.color = pre
```

```
viousColor;
                    }
                    else if (GameObject.FindGameObjectWithTag
("ball_12") != null && GameObject.FindGameObjectWithTag ("ball
_11") == null && delayTime == 0) {
                        ballRenderer [11].material.color = pre
viousColor;
                    }
                    else if (GameObject.FindGameObjectWithTag
("ball_3") != null && GameObject.FindGameObjectWithTag ("ball_
12") == null && delayTime == 0) {
                        ballRenderer [2].material.color = prev
iousColor;
                    }
                    else if (GameObject.FindGameObjectWithTag
("ball_6") != null && GameObject.FindGameObjectWithTag ("ball_
3") == null && delayTime == 0) {
                        ballRenderer [5].material.color = prev
iousColor;
                    }
                    else if (GameObject.FindGameObjectWithTag
("ball_5") != null && GameObject.FindGameObjectWithTag ("ball_
6") == null && delayTime == 0) {
                        ballRenderer [4].material.color = prev
iousColor;
                    }
                    else if (GameObject.FindGameObjectWithTag
("ball_14") != null && GameObject.FindGameObjectWithTag ("ball
_5") == null && delayTime == 0) {
                        ballRenderer [13].material.color = pre
viousColor;
                    }
                    else if (GameObject.FindGameObjectWithTag
("ball_7") != null && GameObject.FindGameObjectWithTag ("ball_
14") == null && delayTime == 0) {
                        ballRenderer [6].material.color = prev
iousColor;
                    }
                }
            }

        } catch {
        }
    }
}
```

Color_Indicator_Set2_Base

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class Color_Indicator_Set2_Base_ : MonoBehaviour {
    Renderer[] ballRenderer;
    Color previousColor;
    Color32 nextObject = new Color32(7,201,25,255);
    float delayTime = 0;

    // Use this for initialization
    void Start () {
        ballRenderer = GetComponentsInChildren<Renderer>();
        previousColor = GetComponentInChildren<Renderer> ().ma
terial.color;
    }

    void Update()
    {
        if (delayTime > 0) {
            delayTime -= Time.deltaTime;
            if (delayTime < 0) {
                delayTime = 0;
            }
        }
        try {


            if(Balls_Colliders_Grabbing_Base_.beAbleToPickUpNe
xtBall == true)
            {
                if (GameObject.FindGameObjectWithTag ("ball_3"
) != null && delayTime == 0) {
                    ballRenderer [2].material.color = nextObje
ct;
                    delayTime = 0.65f;
                }
                else if (GameObject.FindGameObjectWithTag ("ba
ll_15") != null && GameObject.FindGameObjectWithTag ("ball_3")
 == null && delayTime == 0) {
                    ballRenderer [14].material.color = nextObj
ect;
                    delayTime = 0.65f;
                }
                else if (GameObject.FindGameObjectWithTag ("ba
ll_2") != null && GameObject.FindGameObjectWithTag ("ball_15")
 == null && delayTime == 0) {
                    ballRenderer [1].material.color = nextObje
ct;
```

```
                    delayTime = 0.65f;
                }
                else if (GameObject.FindGameObjectWithTag ("ba
ll_8") != null && GameObject.FindGameObjectWithTag ("ball_2")
== null && delayTime == 0) {
                    ballRenderer [7].material.color = nextObje
ct;
                    delayTime = 0.65f;
                }
                else if (GameObject.FindGameObjectWithTag ("ba
ll_5") != null && GameObject.FindGameObjectWithTag ("ball_8")
== null && delayTime == 0) {
                    ballRenderer [4].material.color = nextObje
ct;
                    delayTime = 0.65f;
                }
                else if (GameObject.FindGameObjectWithTag ("ba
ll_14") != null && GameObject.FindGameObjectWithTag ("ball_5")
 == null && delayTime == 0) {
                    ballRenderer [13].material.color = nextObj
ect;
                    delayTime = 0.65f;
                }
                else if (GameObject.FindGameObjectWithTag ("ba
ll_13") != null && GameObject.FindGameObjectWithTag ("ball_14"
) == null && delayTime == 0) {
                    ballRenderer [12].material.color = nextObj
ect;
                    delayTime = 0.65f;
                }
                else if (GameObject.FindGameObjectWithTag ("ba
ll_6") != null && GameObject.FindGameObjectWithTag ("ball_13")
 == null && delayTime == 0) {
                    ballRenderer [5].material.color = nextObje
ct;
                    delayTime = 0.65f;
                }
                else if (GameObject.FindGameObjectWithTag ("ba
ll_4") != null && GameObject.FindGameObjectWithTag ("ball_6")
== null && delayTime == 0) {
                    ballRenderer [3].material.color = nextObje
ct;
                    delayTime = 0.65f;
                }
            }
            else if(Balls_Colliders_Grabbing_Base_.beAbleToPic
kUpNextBall == false)
            {
                if(Balls_Colliders_Grabbing_Base_.playerIsHold
```

```
ing == false)
                {
                    if (GameObject.FindGameObjectWithTag ("bal
l_3") != null && delayTime == 0) {
                        ballRenderer [2].material.color = prev
iousColor;
                    }
                    else if (GameObject.FindGameObjectWithTag
("ball_15") != null && GameObject.FindGameObjectWithTag ("ball
_3") == null && delayTime == 0) {
                        ballRenderer [14].material.color = pre
viousColor;
                    }
                    else if (GameObject.FindGameObjectWithTag
("ball_2") != null && GameObject.FindGameObjectWithTag ("ball_
15") == null && delayTime == 0) {
                        ballRenderer [1].material.color = prev
iousColor;
                    }
                    else if (GameObject.FindGameObjectWithTag
("ball_8") != null && GameObject.FindGameObjectWithTag ("ball_
2") == null && delayTime == 0) {
                        ballRenderer [7].material.color = prev
iousColor;
                    }
                    else if (GameObject.FindGameObjectWithTag
("ball_5") != null && GameObject.FindGameObjectWithTag ("ball_
8") == null && delayTime == 0) {
                        ballRenderer [4].material.color = prev
iousColor;
                    }
                    else if (GameObject.FindGameObjectWithTag
("ball_14") != null && GameObject.FindGameObjectWithTag ("ball
_5") == null && delayTime == 0) {
                        ballRenderer [13].material.color = pre
viousColor;
                    }
                    else if (GameObject.FindGameObjectWithTag
("ball_13") != null && GameObject.FindGameObjectWithTag ("ball
_14") == null && delayTime == 0) {
                        ballRenderer [12].material.color = pre
viousColor;
                    }
                    else if (GameObject.FindGameObjectWithTag
("ball_6") != null && GameObject.FindGameObjectWithTag ("ball_
13") == null && delayTime == 0) {
                        ballRenderer [5].material.color = prev
iousColor;
                    }
```

```
                    else if (GameObject.FindGameObjectWithTag
("ball_4") != null && GameObject.FindGameObjectWithTag ("ball_
6") == null && delayTime == 0) {
                        ballRenderer [3].material.color = prev
iousColor;
                    }
                }
            }
        } catch {
        }
    }
}
```

ThreadMotion_Control_3MPU'S

```
using UnityEngine;
using System.Collections;
using System.IO.Ports;
using System.Threading;

public class ThreadMotCont3MPUS : MonoBehaviour {

    public GameObject joint1,joint2,joint3;
    public float ardW, ardX, ardY, ardZ;
    public int mpu;
    public string dataFromArduino;
    private bool shouldExit = false;
    string[] sInput = new string[6] {"0", "0", "0", "0", "0" ,
"0"};
    public static int buttonState=1;
    //public Vector3 axis=Vector3.zero;
    //public float angle;

    SerialPort mySerialPort = new SerialPort ("COM4", 9600);

    Quaternion quat1= new Quaternion(1,0,0,0);
    Quaternion quat2= new Quaternion(1,0,0,0);
    Quaternion quat3= new Quaternion(1,0,0,0);


    void Start ()
    {

        Thread myThread = new Thread (new ThreadStart (ThreadW
orker));
        myThread.Start();
        mySerialPort.Open ();
```

```
    }

    void Update ()
    {
        if (mySerialPort.IsOpen == true) {
            if (sInput.Length == 6) {
                mpu = int.Parse (sInput [0]);
                ardW = float.Parse (sInput [1]);
                ardX = float.Parse (sInput [2]);
                ardY = float.Parse (sInput [3]);
                ardZ = float.Parse (sInput [4]);
                buttonState = int.Parse (sInput [5]);

                if (mpu == 1) {
                    quat1.w = ardW;
                    quat1.x = ardY;
                    quat1.y = -ardZ;
                    quat1.z = -ardX;
                } else if (mpu == 2) {
                    quat2.w = ardW;
                    quat2.x = ardY;
                    quat2.y = -ardZ;
                    quat2.z = -ardX;
                } else if (mpu == 3) {
                    quat3.w = ardW;
                    quat3.x = ardY;
                    quat3.y = -ardZ;
                    quat3.z = -ardX;

                }

                if (buttonState == 0) {
                    joint1.GetComponent<Renderer> ().material.
color = Color.black;
                } else if (buttonState == 1) {
                    joint1.GetComponent<Renderer> ().material.
color = Color.white;
                }


                //joint1.GetComponent<Renderer> ().material.co
lor = Color.black;

                joint1.transform.rotation = quat1;
                joint2.transform.rotation = quat2;
                joint3.transform.rotation = quat3;


            }
```

```
        } else {
            Debug.Log ("Connect the serial Port");
        }

    }
    void ThreadWorker ()
    {
        while (shouldExit == false) {
            try{
                dataFromArduino = mySerialPort.ReadLine ();
                sInput = dataFromArduino.Split (',');
            }catch(System.Exception){
            }
        }
    }

    void OnApplicationQuit()
    {
        mySerialPort.Close ();
        shouldExit = true;
    }
}
```

UI_Timer

```
using UnityEngine;
using System.Collections;
using UnityEngine.UI;

public class UI_Timer : MonoBehaviour {

    public Text timerLabel;

    private float startTime = -1;

    void Update() {
        float time = startTime >= 0 ? UnityEngine.Time.time -
startTime : 0;

        var minutes = time / 180; //Divide the guiTime by sixt
y to get the minutes.
        var seconds = time % 60;//Use the euclidean division f
or the seconds.
        var fraction = (time * 100) % 100;

        //update the label value

        timerLabel.text = string.Format ("{0:00} : {1:00} : {2
```

```
:000}", minutes, seconds, fraction);

        if (Input.GetKeyDown (KeyCode.Space)) {

            startTime = UnityEngine.Time.time;

        }

        /*if (Input.GetKeyDown(KeyCode.Space))
        {
            if (Time.timeScale == 1)
            {
                Time.timeScale = 0;
            }
            else
            {
                Time.timeScale = 1;
            }*/
        }

    }
```