ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ

ΣΧΟΛΗ ΝΑΥΠΗΓΩΝ ΜΗΧΑΝΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ

ΧΡΗΣΗ ΤΗΣ ΒΙΒΛΙΟΘΗΚΗΣ ΛΟΓΙΣΜΙΚΟΥ VRPN ΓΙΑ ΤΗΝ ΣΥΝΕΡΓΑΣΙΑ ΠΕΡΙΦΕΡΕΙΑΚΟΥ ΕΞΟΠΛΙΣΜΟΥ ΕΙΚΟΝΙΚΗΣ ΠΡΑΓΜΑΤΙΚΟΤΗΤΑΣ ΜΕ ΤΟ ΠΕΡΙΒΑΛΛΟΝ UNITY3D

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

ΤΟΥ

ΣΤΑΥΡΟΥ ΓΕΩΡΓΙΟΥ

ΑΘΗΝΑ ΜΑΡΤΙΟΣ 2017

Ευχαριστίες

Θα ήθελα να ευχαριστήσω τον καθηγητή Αλέξανδρο – Αλβέρτο Γκίνη για την ευκαιρία που μου έδωσε να ασχοληθώ με ένα επίκαιρο θέμα διπλωματικής εργασίας, το οποίο πιστεύω θα έχει ευρεία εφαρμογή στις επιστήμες της μηχανικής στο άμεσο μέλλον. Χωρίς την συμβολή του και την επιστημονική γνώση του δεν θα ήταν ποτέ δυνατόν να αναπτύξω από μόνος μου το πεδίο της εικονικής πραγματικότητας. Επίσης θα ήθελα να ευχαριστήσω την υποψήφια διδάκτωρ Αναστασία Κατσαβριά για την υπομονή της και συμβολή σε θέματα που άπτονταν προγραμματισμού σε γλώσσα C#.

Επίσης, είμαι ευγνώμων στα υπόλοιπα μέλη της εξεταστικής επιτροπής της διπλωματικής εργασίας μου, Καθηγητές κκ. Γεώργιο Ζαραφωνίτη και Παναγιώτη Δ. Κακλή για την προσεκτική ανάγνωση της εργασίας μου και για τις πολύτιμες υποδείξεις τους.

Τέλος ευχαριστώ τους γονείς μου για την συμπαράσταση τους και αφιερώνω αυτήν την εργασία στην μνήμη του πατέρα μου Ιωάννη Σταύρου.

Περιεχόμενα

Περίληψη

Σκοπός της παρούσας διπλωματικής εργασίας είναι η διασύνδεση του εξοπλισμού του εργαστηρίου εικονικής πραγματικότητας της σχολής Ναυπηγών Μηχανολόγων Μηχανικών, συγκεκριμένα του ηλεκτρομαγνητικού συστήματος εντοπισμού αισθητήρων Flock of Birds της εταιρίας www.ascension-tech.com, τα γάντια εικονικής πραγματικότητας της εταιρίας 5DT, το ποντίκι έξι βαθμών ελευθερίας (6DOF) 3D Space Navigator καθώς και το σύστημα ανίχνευσης κίνησης με υπέρυθρες κάμερες της εταιρίας www.ar-tracking.com με την μηχανή παιχνιδιών (Game Engine) Unity3D διαμέσου της βιβλιοθήκης VRPN.

Αρχικά, έγινε η διασύνδεση των παραπάνω συσκευών σε τοπικό επίπεδο με απευθείας σύνδεση τους σε Serial αλλά και USB θύρα του υπολογιστή σταθμού, καθώς και μέσω τοπικού δικτύου TCP/IP.

Στην συνέχεια, πραγματοποιήθηκε μετάφραση (Compile) του διαθέσιμου πηγαίου κώδικα με την βοήθεια του Visual Studio για αρχιτεκτονική επεξεργαστή AMD64 και ενσωματώθηκε η παραχθείσα νέα βιβλιοθήκη VRPNWrapper.dll στην εφαρμογή Unity3D 64 Bit Ver. 5.5. Ακολούθως εμπλουτίστηκε η βιβλιοθήκη του VRPNServer με σκοπό την επίτευξη της επικοινωνίας των περιφερειακών συσκευών με τον Unity3D με μεσολάβηση του VRPNWrapper.dll.

Τέλος, δημιουργήθηκε ένα εικονικό περιβάλλον στο οποίο ο χρήστης δύναται να αλληλοεπιδράσει με αντικείμενα και να περιηγηθεί στον χώρο με την χρήση των περιφερειακών συσκευών.

1. Εισαγωγή

1.1.Η εικονική πραγματικότητα εν συντομία

Η εικονική πραγματικότητα είναι ένας όρος τον οποίο έχουμε ακούσει πολλές φορές στις μέρες μας. Εφαρμογές εικονικής πραγματικότητας (Virtual Reality) αλλά και επαυξημένης εικονικής πραγματικότητας (Augmented Reality) παρουσιάζονται ολοένα και πιο συχνά σε διάφορους τομείς και πεδία της επιστήμης καθώς και του χώρου της ψυχαγωγίας και της διασκέδασης.

Τι είναι λοιπόν η εικονική πραγματικότητα και σε ποιους απευθύνεται? Η υλοποίηση του περιβάλλοντος της εικονικής πραγματικότητας πραγματοποιείται από ένα ηλεκτρονικό υπολογιστικό σύστημα, το οποίο μπορεί να είναι ένας προσωπικός υπολογιστής ή συστοιχία υπολογιστών, και από περιφερειακές συσκευές των οποίων ο ρόλος είναι να δημιουργήσουν την ψευδαίσθηση στον χρήστη ότι βρίσκεται σε ένα διαδραστικό εικονικό περιβάλλον. Οι ανθρώπινες αισθήσεις που μπορεί να διεγείρει ένα εικονικό περιβάλλον είναι η όραση, η ακοή, η αίσθηση της αφής και πολύ σπάνια η όσφρηση. Οι εφαρμογές της εικονικής πραγματικότητας έχουν μεγάλη επιτυχία στην ιατρική όπου πραγματοποιούνται εικονικές εγχειρήσεις από εκπαιδευόμενους ιατρούς, στην μηχανολογία για την εκπαίδευση νέων μηχανικών σε καινούρια συστήματα, στο στρατό για την εκπαίδευση πιλότων μαχητικών αεροσκαφών, σε επαγγελματίες οδηγούς μηχανοκίνητου αθλητισμού ακόμα και σε ψυχολόγους όπου υποβάλλουν τους ασθενείς τους σε διάφορα τεστ για την διάγνωση και αντιμετώπιση της ασθένειας τους.

1.2.Τι είναι το VRPN (Virtual Reality Peripheral Network)

Το Virtual Reality Peripheral Network (VRPN) είναι ένα σύνολο από προγράμματα, γραμμένα σε γλώσσα C# και ένα σύνολο από διαχειριστές (Servers) οι οποίοι είναι σχεδιασμένοι να δημιουργούν ένα διαφανές δίκτυο μεταξύ των προγραμμάτων αυτών και των φυσικών εξωτερικών συσκευών, όπως είναι οι ανιχνευτές αισθητήρων (Trackers), που χρησιμοποιούνται σε εφαρμογές εικονικής πραγματικότητας.

Η βασική ιδέα είναι να έχουμε έναν ηλεκτρονικό υπολογιστή ο οποίος αποτελεί τον σταθμό για τον έλεγχο των περιφερειακών συσκευών**,** όπως τους ανιχνευτές αισθητήρων (Trackers) που προαναφέραμε, συσκευές αίσθησης αφής (Haptic Devices), συσκευές με αναλογικές εισόδους δεδομένων (Analog Inputs), πηγές δημιουργίας ήχου (Sound Generators) κ.τ.λ. Πρέπει να σημειώσουμε ότι υπάρχει η δυνατότητα να χρησιμοποιήσουμε το VRPN με συσκευές οι οποίες είναι συνδεδεμένες απευθείας με τον ηλεκτρονικό υπολογιστή ή με συσκευές οι οποίες είναι συνδεδεμένες με έναν άλλο υπολογιστή σε ένα προσβάσιμο τοπικό δίκτυο.

Η επιτυχία του VRPN σε μια εφαρμογή εικονικής πραγματικότητας έγκειται στο γεγονός ότι χρησιμοποιούμε έναν σταθμό ελέγχου για πάρα πολλές συσκευές τις οποίες διαχειριζόμαστε σχεδόν με τον ίδιο τρόπο, γράφοντας δηλαδή scripts σε γλώσσα C#, αποφεύγοντας να χρησιμοποιήσουμε προγράμματα οδηγούς (Drivers) από τις εταιρίες των συσκευών αυτών.

Το VRPN έχει δοκιμαστεί και λειτουργήσει με επιτυχία σε PC/Win32, PC/Cygwin, PC/Linux, και Mac/OSX 32 και 64-bits αρχιτεκτονική καθώς και σε ARM Linux και Android**,** ενώ σε συστήματα SGI/Irix, HP700/Hpux, Sparc/Solaris, Ipaq/Linux και Zaurus/Linux έγινε αρχικά δοκιμή όμως μειώθηκε το ενδιαφέρον για περαιτέρω ανάπτυξη στις πλατφόρμες αυτές και σταμάτησε η εξέλιξή τους.

1.3.Χάρτης της διπλωματικής εργασίας

Στα κεφάλαια που ακολουθούν γίνεται αναφορά εν συντομία της διαδικασίας διασύνδεσης του περιφερειακού εξοπλισμού εικονικής πραγματικότητας που διαθέτει το εργαστήριο της σχολής Ναυπηγών Μηχανολόγων Μηχανικών, καθώς και της διαδικασίας μετάφρασης (build) των απαραίτητων προγραμμάτων.

Στο κεφάλαιο 2 γίνεται μετάφραση του πηγαίου κώδικα της βιβλιοθήκης VRPN με χρήση των προγραμμάτων CMake και Visual Studio οπότε παράγουμε τα προγράμματα **vrpn_print_devices**, **vrpn_print_performance**, **vrpn_print_messages**, **vrpn_server** που αποτελούν τον VRPNServer καθώς και το αρχείο **VRPNWrapper.dll** το οποίο μας βοηθάει για την διασύνδεση του VRPNServer με το Unity3D.

Ακολουθεί το κεφάλαιο 3 στο οποίο περιγράφουμε την δημιουργία του εικονικού κόσμου με τα εργαλεία που διαθέτει το Unity3D, καθώς και την διαδικασία εισαγωγής και βελτιστοποίησης τρισδιάστατων μοντέλων μέσο του προγράμματος 3D Studio Max.

Εν συνεχεία στο κεφάλαιο 4 γίνεται αναφορά για την χρηστικότητα και ευελιξία που μπορεί να επιτευχθεί με χρήση του VRPNServer και VRPNWrapper όσον αφορά την διασύνδεση περιφερειακών συσκευών εικονικής πραγματικότητας με το πρόγραμμα Unity3D.

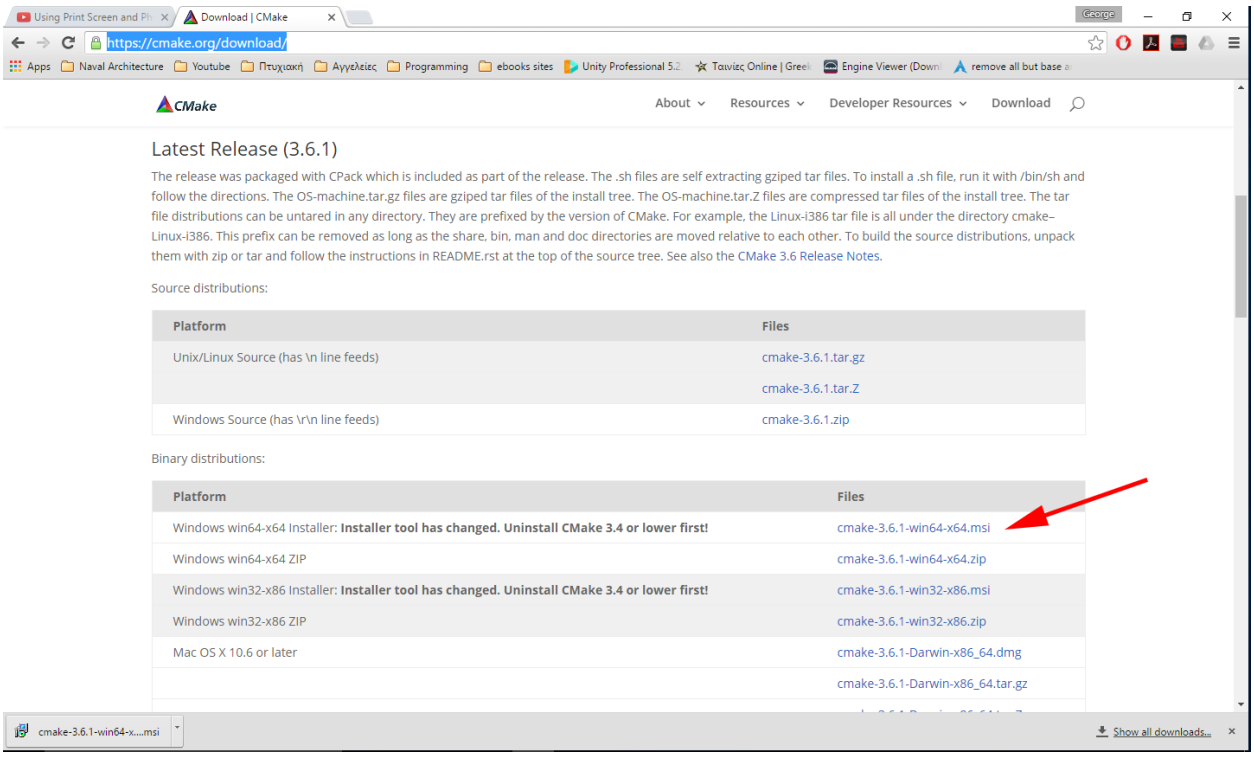
Το κεφάλαιο 5 περιλαμβάνει λεπτομερή περιγραφή του τρόπου διαχείρισης των περιφερειακών συσκευών του εργαστηρίου συγκεκριμένα του ηλεκτρομαγνητικού συστήματος εντοπισμού αισθητήρων Flock of Birds της εταιρίας www.ascension-tech.com, των γαντιών εικονικής πραγματικότητας της εταιρίας 5DT, του ποντικίου έξι βαθμών ελευθερίας (6DOF) 3D Space Navigator καθώς και του σύστηματος ανίχνευσης κίνησης με υπέρυθρες κάμερες της εταιρίας www.ar-tracking.com.

Τέλος η ανάπτυξη και εξέλιξη του κώδικα που αφορά των τρόπο διαχείρισης των περιφερειακών συσκευών από το Unity3D γίνεται στο κεφάλαιο 6, ενώ στο κεφάλαιο 7 εν συντομία αναφέρουμε σχόλια και παρατηρήσεις. Το κεφάλαιο 8 περιλαμβάνει χρήσιμες ιστοσελίδες ενώ στο κεφάλαιο 9 γίνεται αναφορά της βιβλιογραφίας που χρησιμοποιήθηκε για την διεκπεραίωση της παρούσας διπλωματικής εργασίας.

2. Κτίσιμο (Build) του VRPN Server και VRPNWrapper

2.1. VRPN Server

Σε αυτό το κεφάλαιο θα αναλύσουμε τον τρόπο σύνταξης (Compile) ή κτίσιμο (υλοποίηση) των VRPN Server και VRPNWrapper με τη βοήθεια προγραμμάτων. Μεταβαίνουμε με τον αγαπημένο μας Internet Browser στην ιστοσελίδα https://cmake.org/download/ και κατεβάζουμε την τελευταία έκδοση του προγράμματος **Cmake** για αρχιτεκτονική επεξεργαστή 64Bit, όπως δείχνει και το κόκκινο βέλος στην παρακάτω εικόνα, και το κάνουμε εγκατάσταση στον υπολογιστή μας.

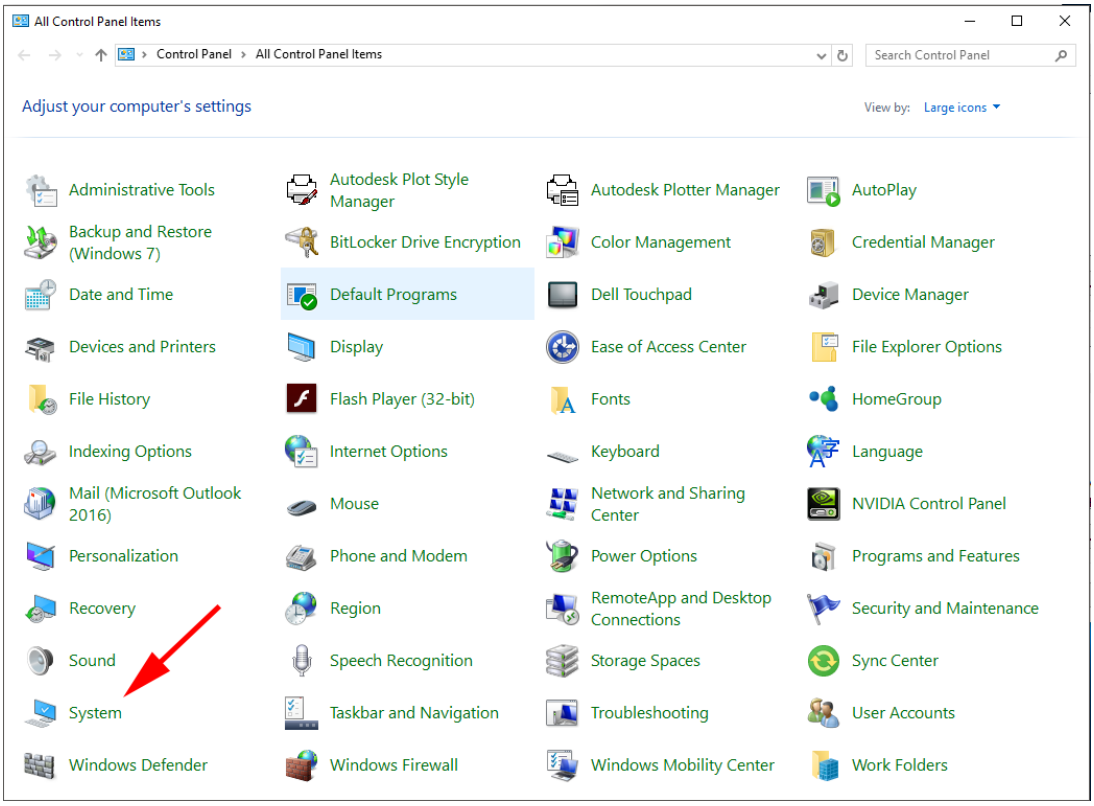


Για να κτιστεί (Build) ο VRPN Server φτιάχνουμε ένα φάκελο στο σκληρό μας δίσκο στην διαδρομή C: και τον ονομάζουμε libs οπότε τελικά θα έχουμε την διαδρομή C:\libs. Στο φάκελο αυτό δημιουργούμε πέντε άλλους φακέλους στους οποίους εισάγουμε επιμέρους προγράμματα που χρειαζόμαστε για να κτίσουμε (Build) με επιτυχία τον VRPN Server αλλά και το VRPNWrapper. Οι ονομασίες των φακέλων αυτών είναι οι παρακάτω.

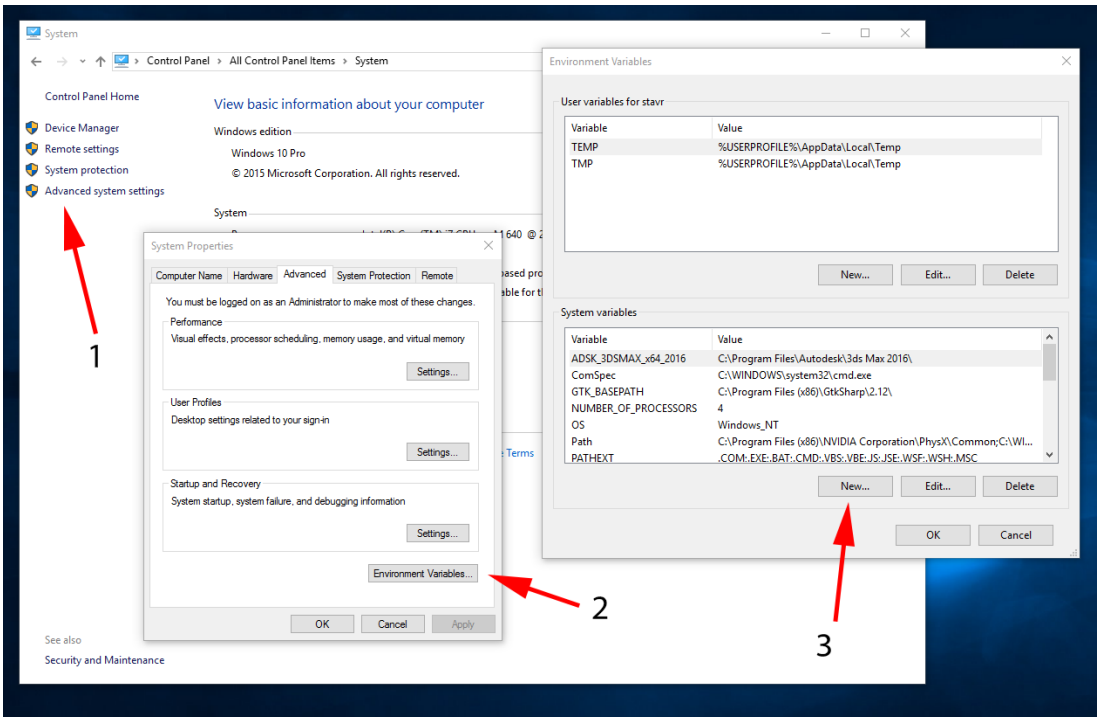
- FaceTrackingAPI_NC3.2
- Pthreads
- VideoWrapper
- Vrpn
- VRPNWrapper

Το επόμενο βήμα είναι να περάσουμε μεταβλητές συστήματος στον υπολογιστή μας ώστε να γνωρίζει που είναι αποθηκευμένα τα παραπάνω προγράμματα και φάκελοι. Επιλέγουμε το Start Menu των

Windows και γράφουμε στην αναζήτηση System ή μπορούμε να μεταβούμε στο Control Panel και να επιλέξουμε από εκεί το εικονίδιο System.



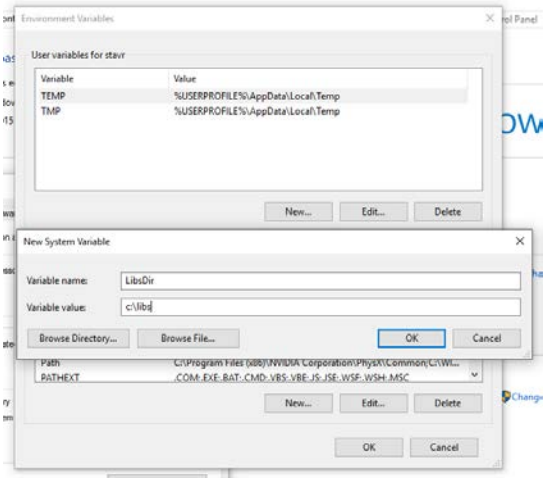
Εφόσον έχουμε ανοίξει το **System** επιλέγουμε την συντόμευση **Advanced System Setting** και μετά από το αναδυόμενο παράθυρο που θα εμφανιστεί επιλέγουμε την συντόμευση **Environment Variables**. Στην νέα καρτέλα που θα ανοίξει στην υποσημείωση που αναγράφει **System variables** κάνουμε κλικ στο **New** και περνάμε τις καινούριες μεταβλητές όπως δείχνει η εικόνα που ακολουθεί.



Οι καινούριες μεταβλητές συστήματος (System Variables) θα πρέπει να έχουν δύο διακριτικά, την ονομασία (name) της μεταβλητής και την τιμή (value), δηλαδή την διαδρομή που βρίσκονται τα βοηθητικά προγράμματα που χρειαζόμαστε όπως διακρίνεται στην ακόλουθη λίστα.

- Variable name = LibsDir, Variable value = C:\libs
- Variable name = SM_API_CPP_WRAPPERS, Variable value = C:\libs\FaceTrackingAPI_NC3.2\Wrappers\C++
- Variable name = SM_API_PATH, Variable value = C:\libs\FaceTrackingAPI_NC3.2\API
- Variable name = VIDEOWRAPPER, Variable value = C:\libs\VideoWrapper

Στην παρακάτω εικόνα σαν παράδειγμα περνάμε την πρώτη μεταβλητή με την ονομασία LibsDir και την διαδρομή C:\libs.



Τώρα που έχουμε περάσει όλες τις μεταβλητές συστήματος καλό είναι να κάνουμε επανεκκίνηση του υπολογιστή μας, ώστε το λειτουργικό σύστημα (Windows) να περιλάβει τις νέες αυτές μεταβλητές στις βιβλιοθήκες του.

Μετά την επανεκκίνηση του λειτουργικού συστήματος μεταβαίνουμε στον φάκελο C:\libs\vrpn και ανοίγουμε το αρχείο .gitmodules με ένα Text Editor όπως το Notepad++. Θα δούμε στο αρχείο αυτό το παρακάτω κείμενο το οποίο μας δίνει πληροφορίες για επιπλέον προγράμματα που χρειαζόμαστε.

[submodule "submodules/hidapi"]

path = submodules/hidapi

url = https://github.com/vrpn/hidapi.git

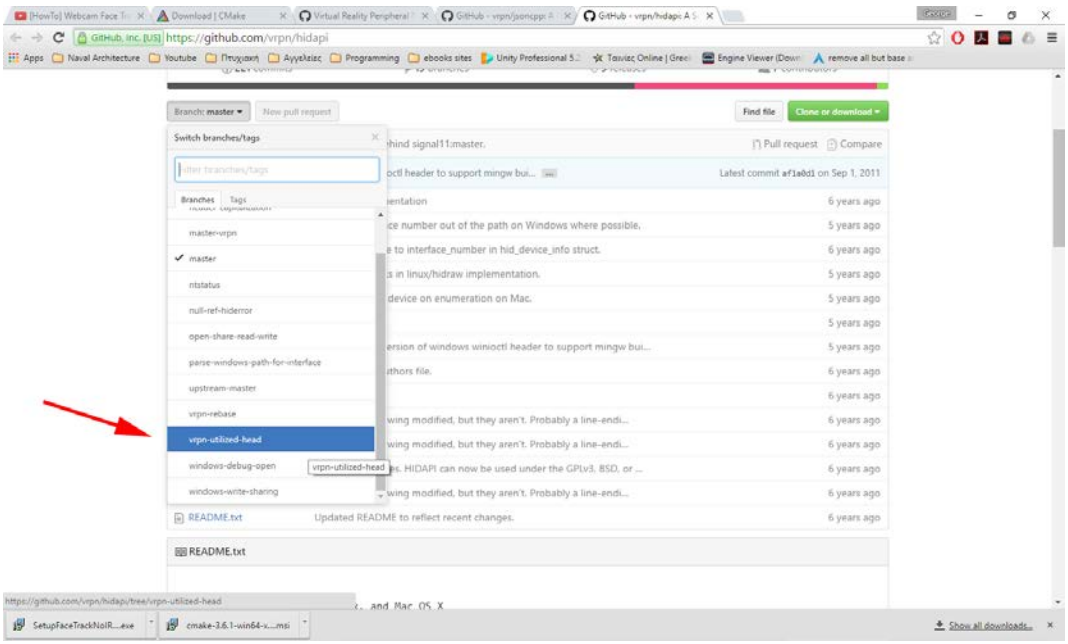
branch = vrpn-utilized-head

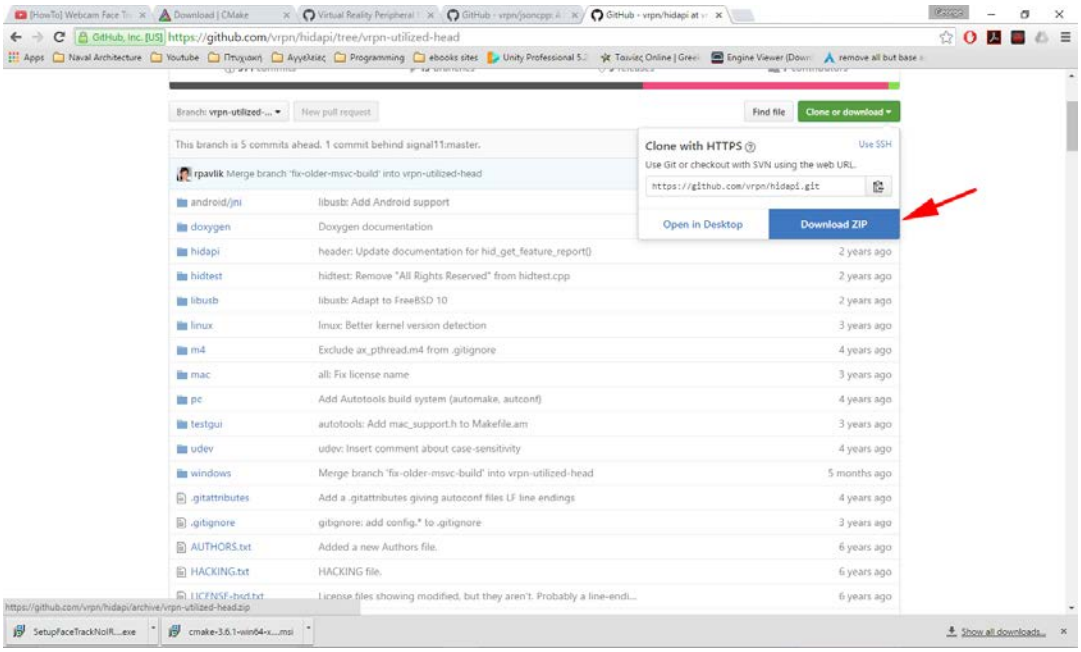
[submodule "submodules/jsoncpp"]

path = submodules/jsoncpp

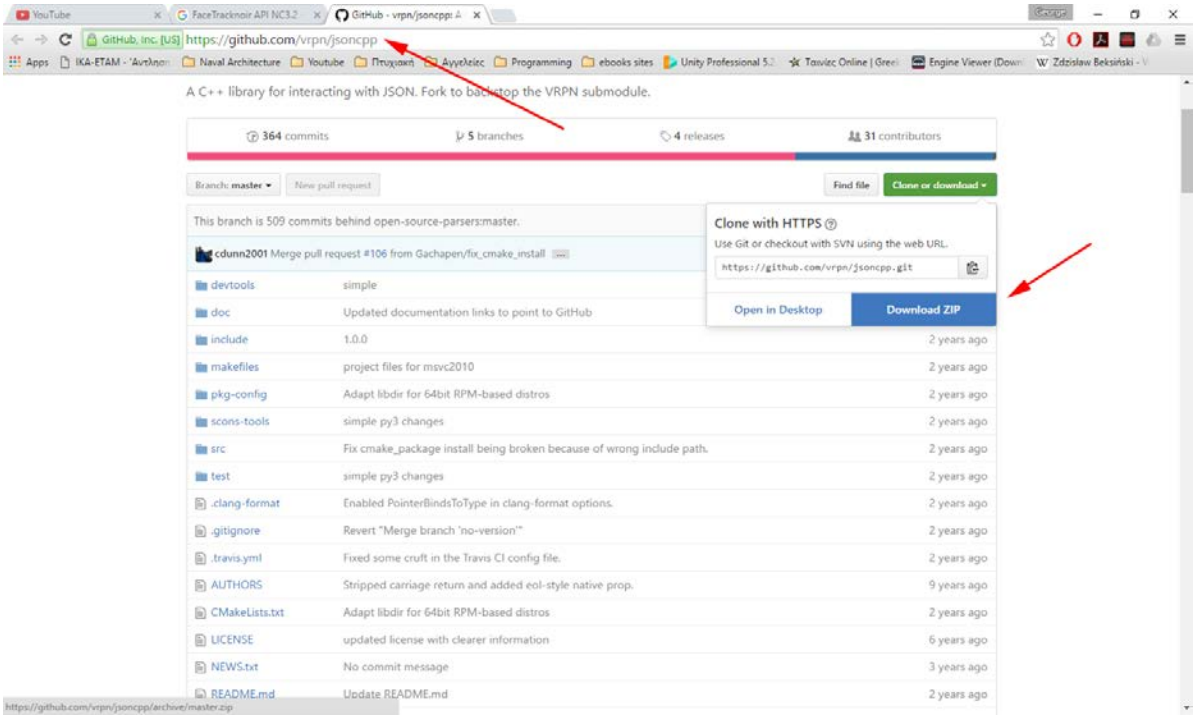
url = https://github.com/vrpn/jsoncpp.git

Παρατηρούμε να αναφέρεται ότι στο φάκελο submodules, ο οποίος βρίσκεται στην διαδρομή C:\libs\vrpn\submodules, πρέπει να εισάγουμε δυο νέους φακέλους με την ονομασία hidapi και jsoncpp. Στους φακέλους αυτούς πρέπει να περιέχονται τα νέα βοηθητικά προγράμματα τα οποία μπορούμε να κατεβάσουμε από το διαδίκτυο. Συγκεκριμένα στην ιστοσελίδα https://github.com/vrpn/hidapi.git επιλέγουμε το branch vrpn-utilized-head και κατεβάζουμε ένα συμπιεσμένο αρχείο με την ονομασία hidapi-vrpn-utilized-head.zip όπου στην συνέχεια το κάνουμε αποσυμπίεση στην διαδρομή C:\libs\vrpn\submodules οπότε και δημιουργείται ένας νέος φάκελος με διαδρομή C:\libs\vrpn\submodules\hidapi.

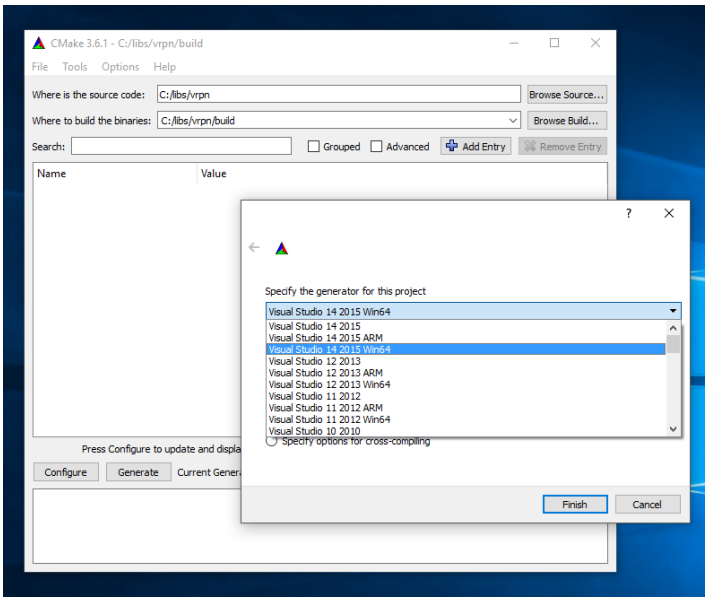




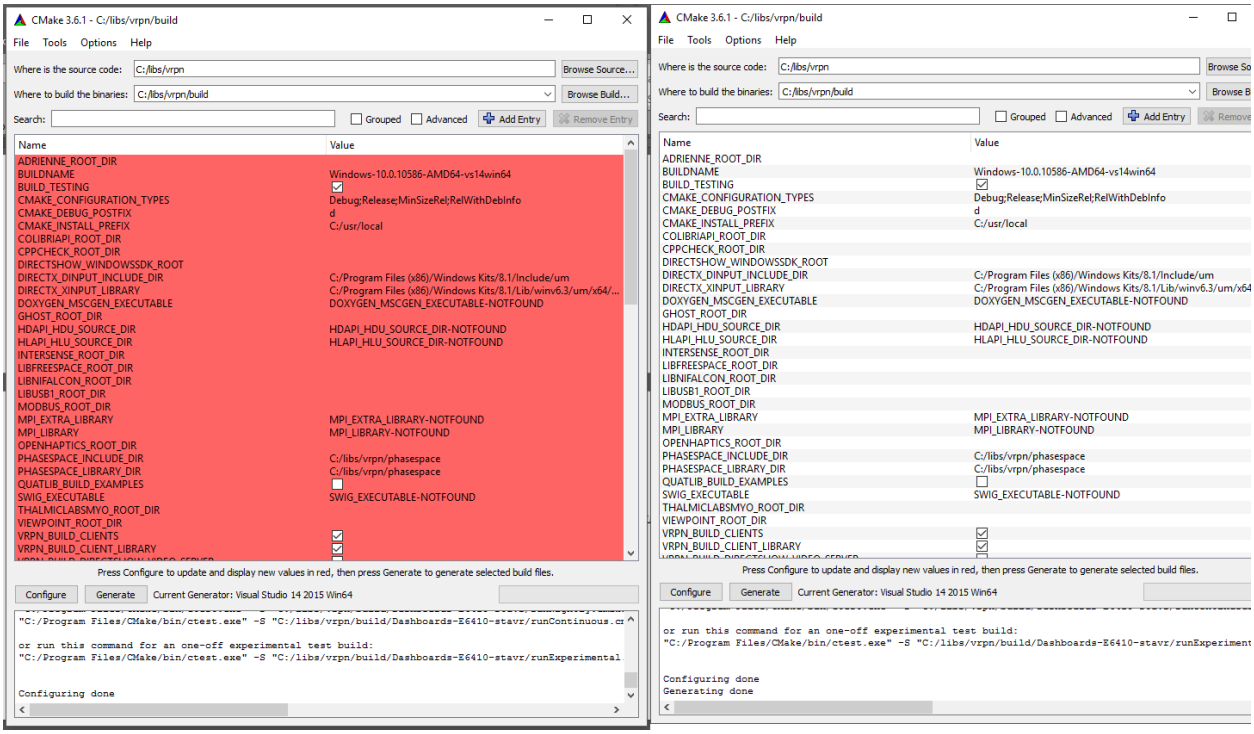
Στην συνέχεια από την ίδια ιστοσελίδα και τη διαδρομή https://github.com/vrpn/jsoncpp.git κατεβάζουμε από το Master Branch το συμπιεσμένο αρχείο jsoncpp-master.zip το οποίο κάνουμε αποσυμπίεση στην διαδρομή C:\libs\vrpn\submodules οπότε δημιουργείται ένας νέος φάκελος με διαδρομή C:\libs\vrpn\submodules\jsoncpp.



Επόμενο βήμα είναι ανοίξουμε το πρόγραμμα **Cmake** και να επιλέξουμε για την διαδρομή **Where is the source code** τον φάκελο **C:/libs/vrpn** ενώ για την διαδρομή **Where to build the binaries** τον φάκελο **C:/libs/vrpn/build** όπου build είναι ένας φάκελος που έχουμε δημιουργήσει στην διαδρομή **C:/libs/vrpn**. Στην συνέχεια επιλέγουμε **Configure** και από το αναδυόμενο μενού επιλέγουμε **Visual Studio 14 2015 Win64** για να κτίσουμε τον VRPN Server για αρχιτεκτονική επεξεργαστή 64bit. Προσοχή χρειάζεται το γεγονός ότι το Visual Studio 2015 μπορεί να μην έχει εγκαταστημένο C++ Compiler οπότε το Cmake θα μας εμφανίσει σφάλμα για την επιλογή C++ Compiler. Για να λύσουμε το πρόβλημα αυτό ανοίγουμε το Visual Studio και κάνουμε ένα νέο Project C++ οπότε το πρόγραμμα μας παραπέμπει να κατεβάσουμε τον C++ Compiler και να τον κάνουμε εγκατάσταση στο σύστημα μας.

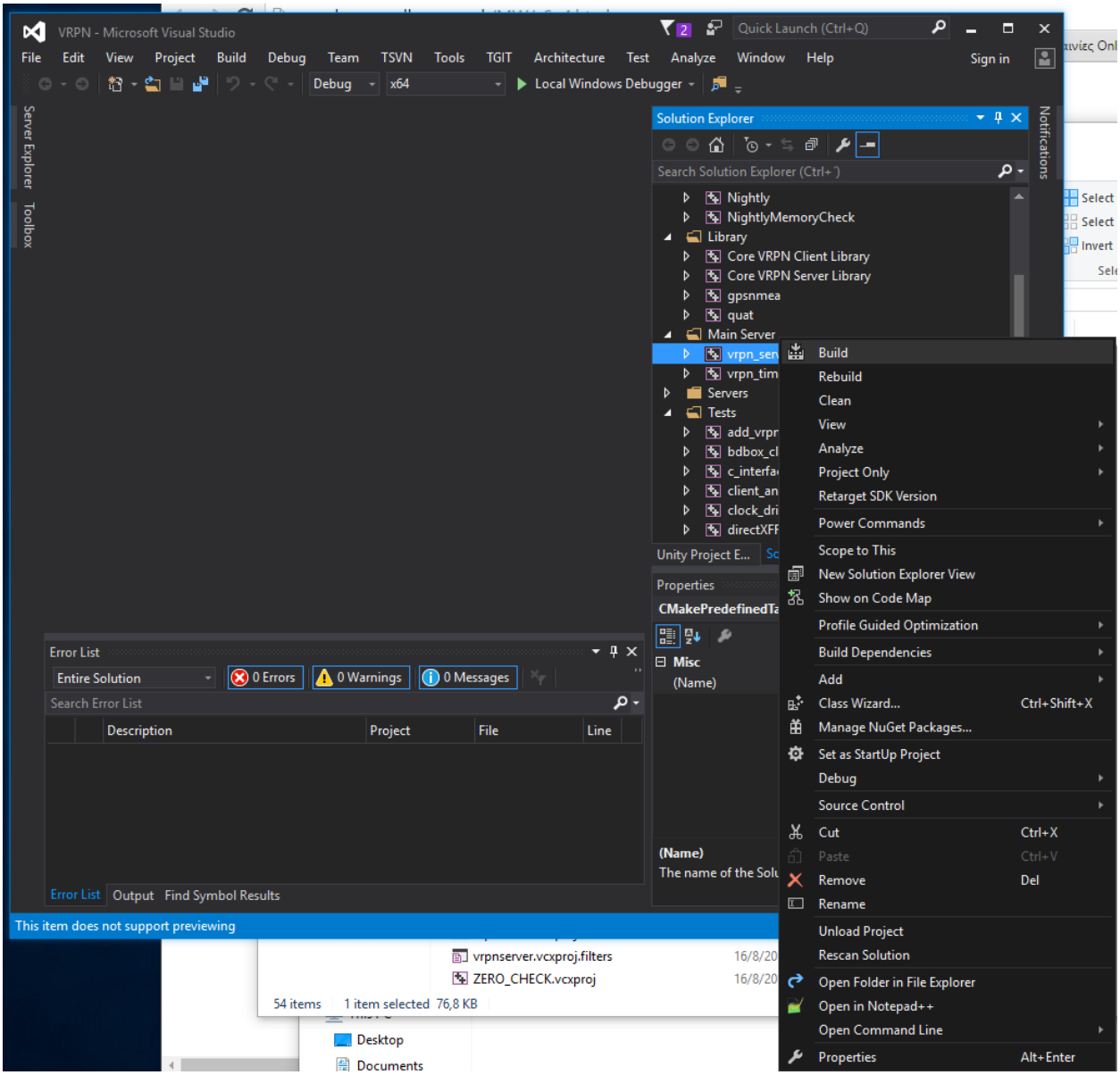


Έπειτα επιλέγουμε Generate δύο φορές ώστε το κόκκινο χρώμα του interface του Cmake να γίνει άσπρο.



Κλείνουμε το Cmake και μεταβαίνουμε στην διαδρομή C:\libs\vrpn\build όπου έχουμε κτίσει το project για να γίνει compile με την βοήθεια του Visual Studio. Κάνοντας διπλό κλικ στο αρχείο VRPN.sln ανοίγουμε το Project του VRPN στο Visual Studio. Δεξιά στο παράθυρο του Visual Studio και συγκεκριμένα στο Solution Explorer κάνουμε δεξί κλικ στα binaries files και επιλέγουμε Build ώστε τελικά

να φτιάξουμε εκτελέσιμα αρχεία *.exe τα οποία θα τρέξουμε για να εκκινήσουμε τον VRPN Server, αλλά και άλλα βοηθητικά προγράμματα, όπως το vrpn_print_devices.exe, τα οποία μας βοηθούν στον οπτικό έλεγχο καλής λειτουργίας των περιφερειακών συσκευών.



Συγκεκριμένα κτίσαμε τα παρακάτω προγράμματα.

- **vrpn_print_devices**
- **vrpn_print_performance**
- **vrpn_print_messages**
- **vrpn_server**

Τα εκτελέσιμα αρχεία για τα προγράμματα **vrpn_print_devices.exe**, **vrpn_print_performance.exe**, **vrpn_print_messages.exe** θα τα βρούμε στην διαδρομή

- **C:\libs\vrpn\build\client_src\Debug**

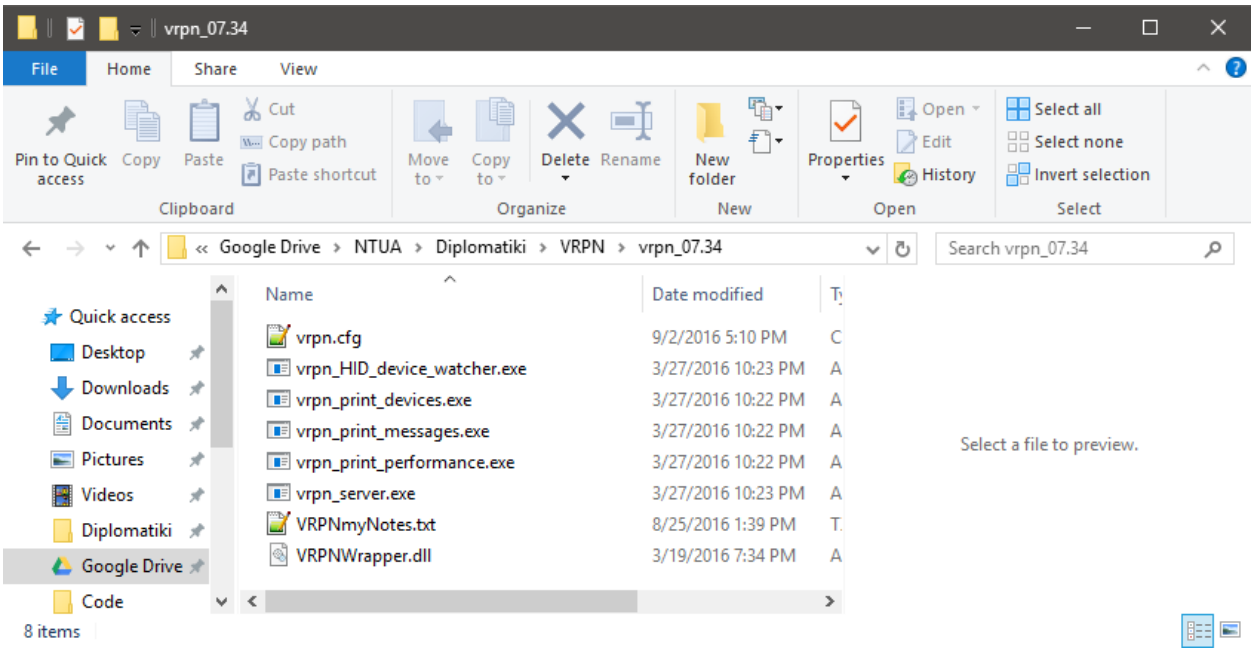
ενώ το εκτελέσιμο αρχείο **vrpn_server.exe** θα το βρούμε στην διαδρομή

- **C:\libs\vrpn\build\server_src\Debug**

Τέλος σημαντικό είναι και το αρχείο **vrpn.cfg** το οποίο βρίσκεται στην διαδρομή

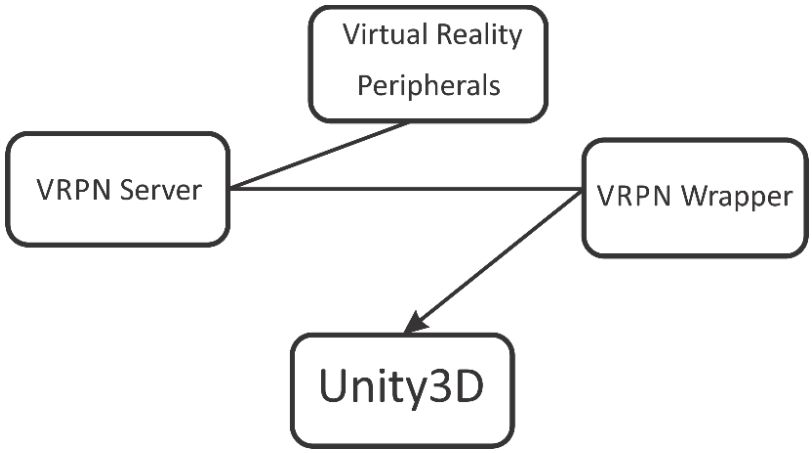
- **C:\libs\vrpn\build\server_src**

Επιλέγουμε τα αρχεία αυτά και τα αντιγράφουμε (copy) σε έναν οποιοδήποτε φάκελο στον σκληρό μας δίσκο όπως διακρίνεται στην παρακάτω εικόνα.

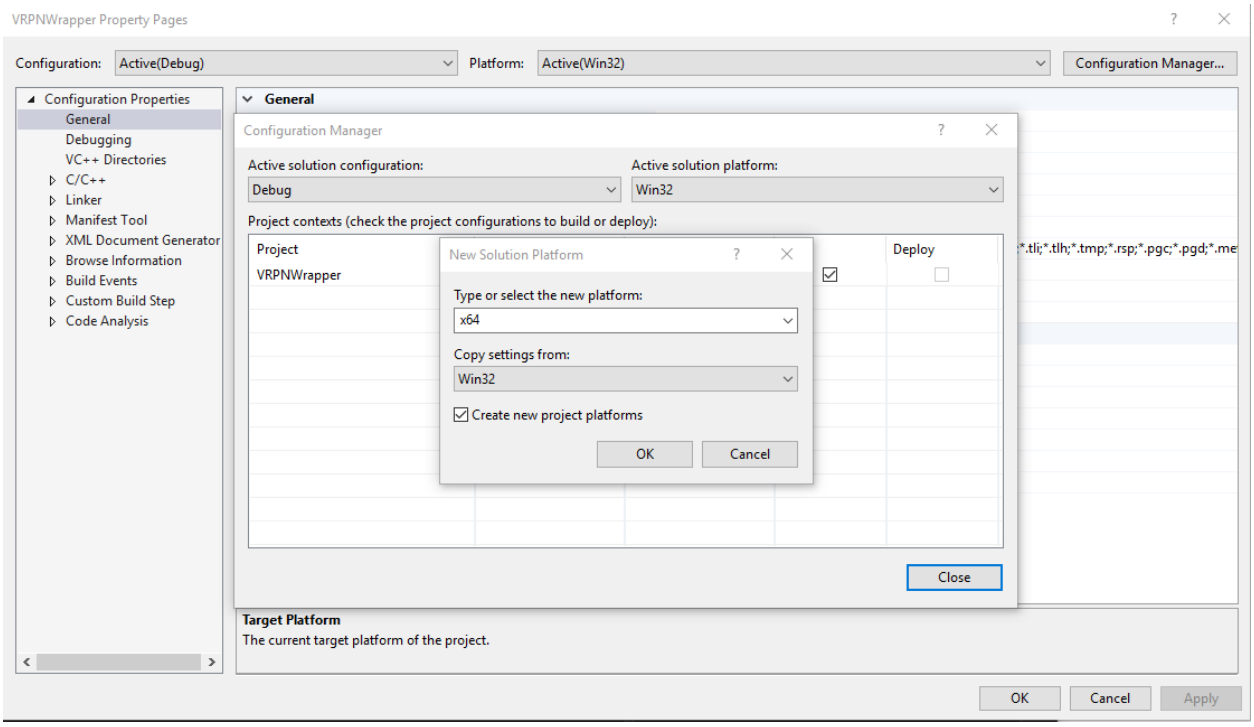


2.2.VRPNWrapper

Τέλος**,** θα πρέπει να κτίσουμε το VRPNWrapper, συγκεκριμένα θα δημιουργήσουμε ένα αρχείο dll με ονομασία VRPNWrapper.dll για αρχιτεκτονική επεξεργαστή 64bit το οποίο είναι υπεύθυνο για την διασύνδεση των περιφερειακών συσκευών εικονικής πραγματικότητας με το πρόγραμμα Unity3D διαμέσου του VRPNServer.



Στον αρχικό φάκελο C:\libs μεταβαίνουμε στην διαδρομή C:\libs\VRPNWrapper\VRPNPlugin\Win32 και ανοίγουμε το αρχείο VRPNPlugin.sln με το Visual Studio. Στην καρτέλα Solution Explorer βλέπουμε το Project με την ονομασία VRPNWrapper, κάνουμε δεξί κλικ και επιλέγουμε Properties. Στην επιλογή Configuration αφήνουμε το Active(Debug) ενώ στον Configuration Manager θα πρέπει να επιλέξουμε για Active Solution Platform το New και μετά X64 ενώ θα πρέπει να αφήσουμε την επιλογή Copy Settings From Win32 όπως διακρίνεται στην εικόνα που ακολουθεί ώστε να κτιστεί το VRPNWrapper.dll για αρχιτεκτονική επεξεργαστή 64Bit.

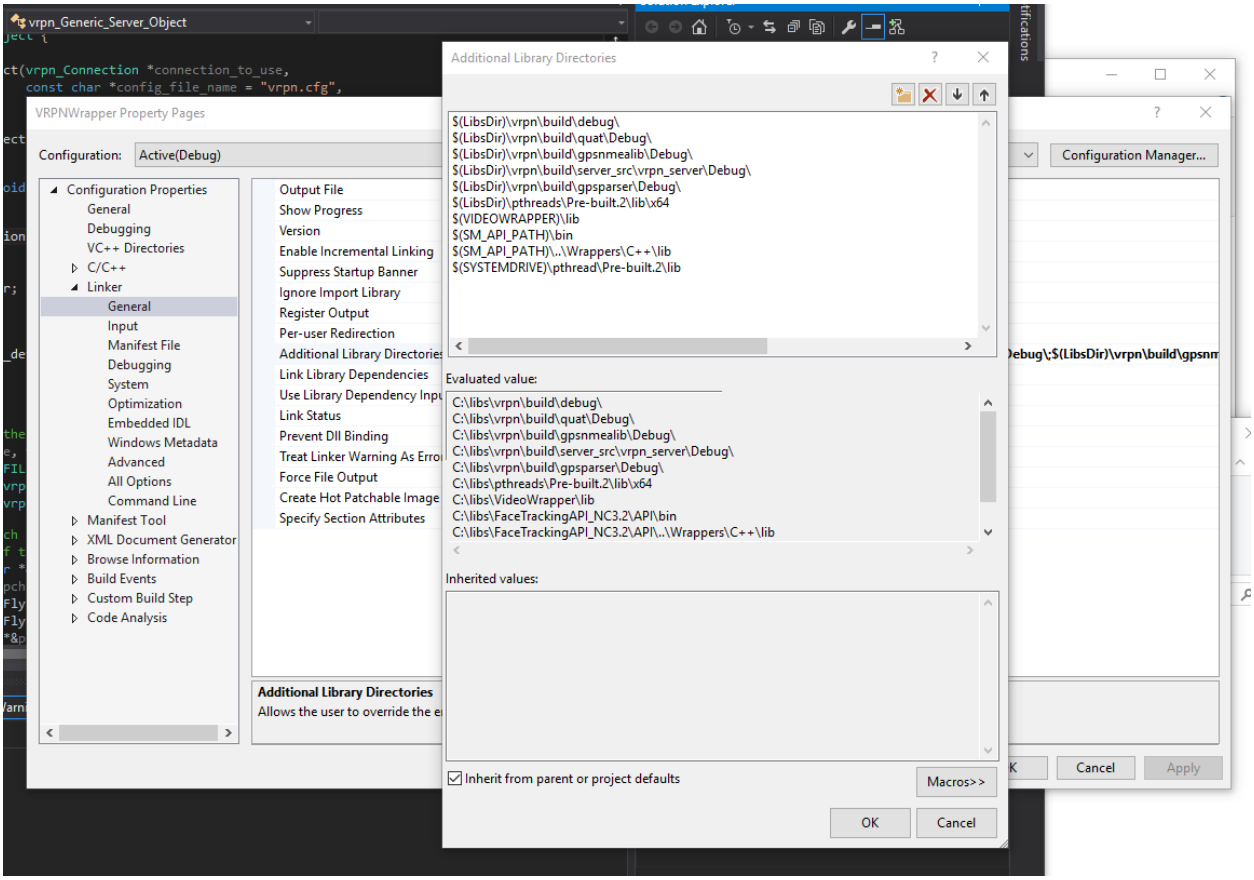


Το επόμενο βήμα που πρέπει να προσέξουμε είναι στο αναδιπλούμενο μενού με την ονομασία Linker επιλέγοντας Input και Additional Dependencies να έχουμε αναγραφόμενες τις παρακάτω βιβλιοθήκες

- **vrpnserverd.lib**
- **quatd.lib**
- **smft32.lib**
- **smftcpp.lib**
- **pthreadVC2.lib**
- **SharedMem.lib**
- **gpsnmead.lib**
- **SetupApi.lib**

Ενώ στο υπομενού Linker -> General -> Additional Library Directories πρέπει να είναι επιλεγμένη η διαδρομή

- $(LibsDir)\pthreads\Pre-built.2\lib\x64



Τέλος στο Linker -> Advanced και συγκεκριμένα στο Target Machine πρέπει να αναφέρει την αρχιτεκτονική 64Bit

- MachineX64(/Machine:X64)

Εφόσον ικανοποιούνται αυτές οι προϋποθέσεις στο αρχικό μενού του Visual Studio κάνουμε δεξί κλικ στο Project VRPNWrapper και επιλέγουμε Build. Το αρχείο VRPNWrapper.dll που δημιουργείται, βρίσκεται στην διαδρομή :

- C:\libs\VRPNWrapper\VRPNPlugin\Win32\build\debug

Τέλος για να ολοκληρώσουμε την κατασκευή του VRPNWrapper και να το χρησιμοποιήσουμε σαν ολοκληρωμένο πρόγραμμα επέκτασης (Plugin) στο Unity3D θα πρέπει να έχουμε και τα ακόλουθα επτά αρχεία τα οποία είναι προγράμματα C#.

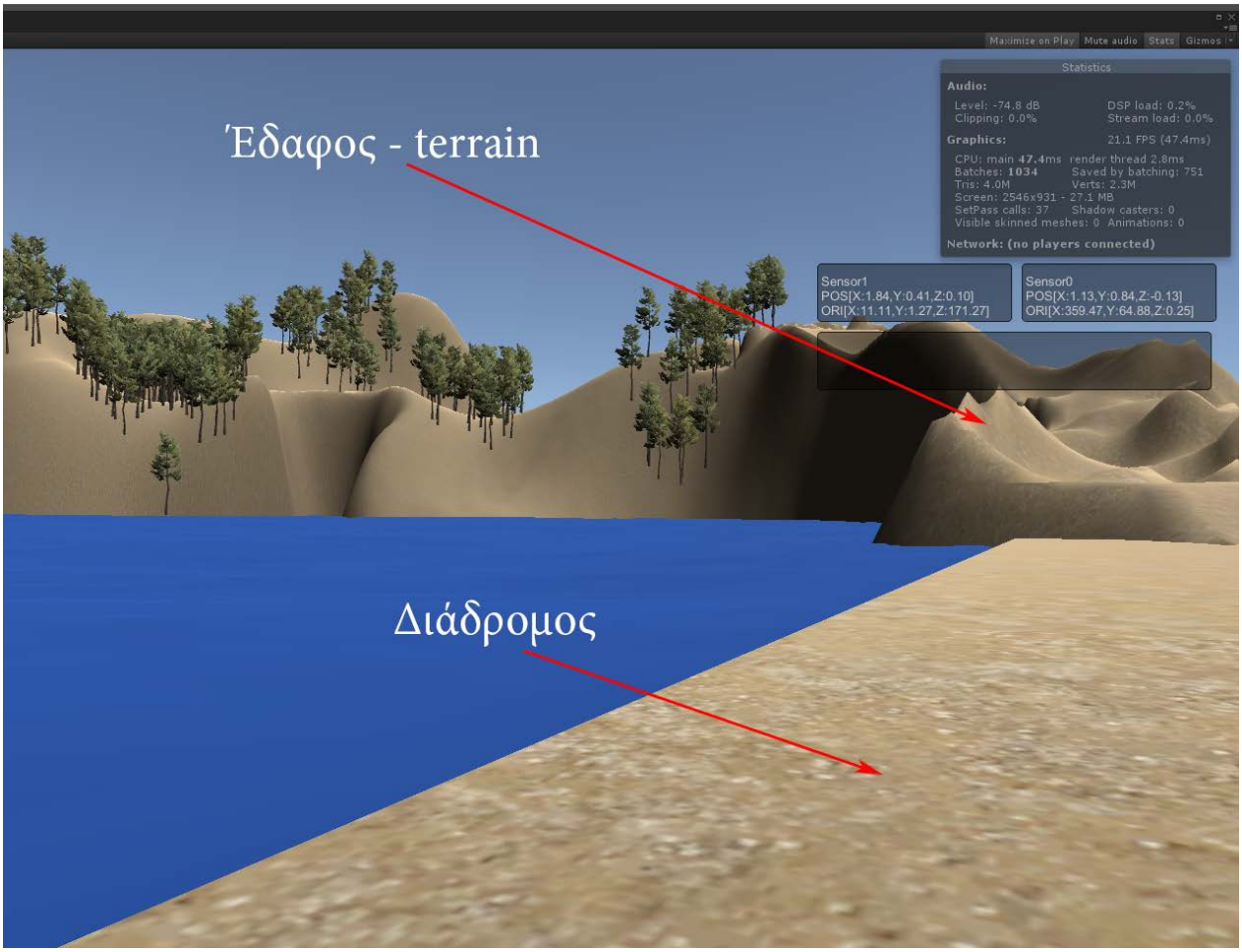
- **Plugins.cs**
- **VRPNAnalog.cs**
- **VRPNButton.cs**
- **VRPNDeviceConfig.cs**
- **VRPNManager.cs**
- **VRPNTracker.cs**
- **VRPNTrackerCalibrate.cs**

3. Δημιουργία του εικονικού κόσμου

3.1. Εισαγωγή

Στην αρχή της υλοποίησης του εικονικού κόσμου χρησιμοποιούνται πρωτόγονα αντικείμενα (Primitive Objects) όπως είναι ο κύβος, η σφαίρα, η πυραμίδα και ο κύλινδρος, τα οποία μας βοηθούν στον αρχικό έλεγχο του κώδικα. Για να είναι όμως πειστικός ο εικονικός κόσμος στον χρήστη θα πρέπει μελλοντικά να δοθεί προσοχή στην λεπτομέρεια των γεωμετριών που θα εισάγουμε. Το εικονικό περιβάλλον λοιπόν θα πρέπει να αποτελείται από περίπλοκες γεωμετρίες και εικόνες οι οποίες πλησιάζουν όσον το δυνατόν περισσότερο την πραγματικότητα ώστε να μην αποσπάται εύκολα η προσοχή του χρήστη από εξωτερικά ερεθίσματα.

Στο Project που αναπτύχθηκε για την παρούσα διπλωματική εργασία δημιουργήσαμε ένα εικονικό περιβάλλον το οποίο αποτελείται από ένα έδαφος (terrain) εμπλουτισμένο με δέντρα και μικρούς λόφους. Οι λόφοι διαβρέχονται περίγυρα από θάλασσα ενώ διάδρομος οδηγεί σε βιομηχανικό κτήριο στο οποίο βρίσκεται δικύλινδρος κινητήρας.

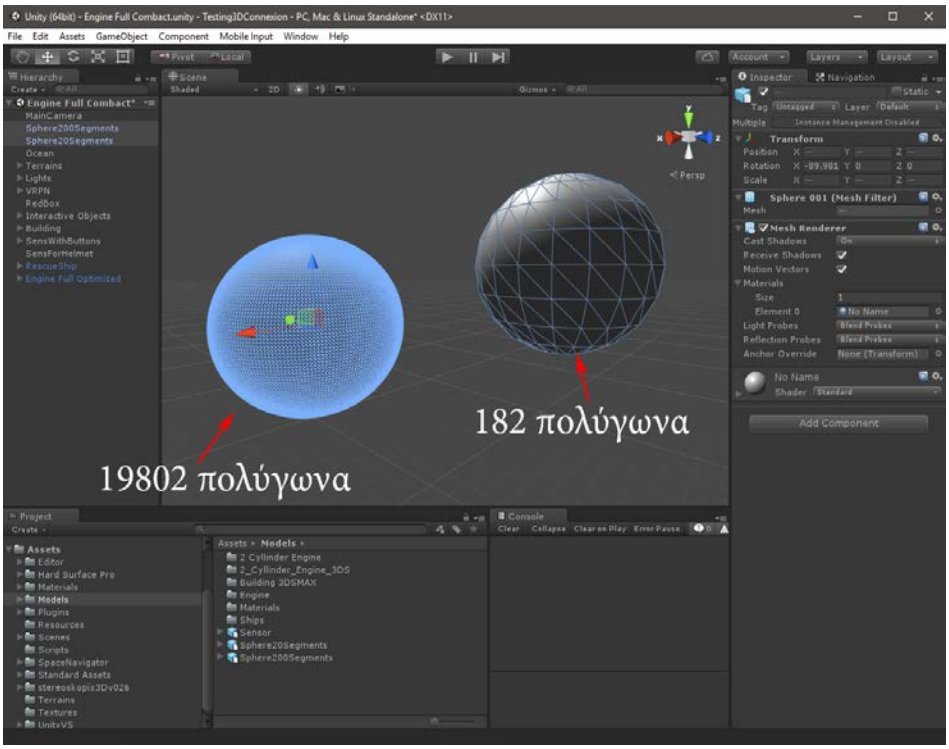


Τα προγράμματα που χρησιμοποιήθηκαν για την κατασκευή του εικονικού κόσμου αλλά και την διαχείριση και βελτιστοποίηση των αντικειμένων είναι τα :

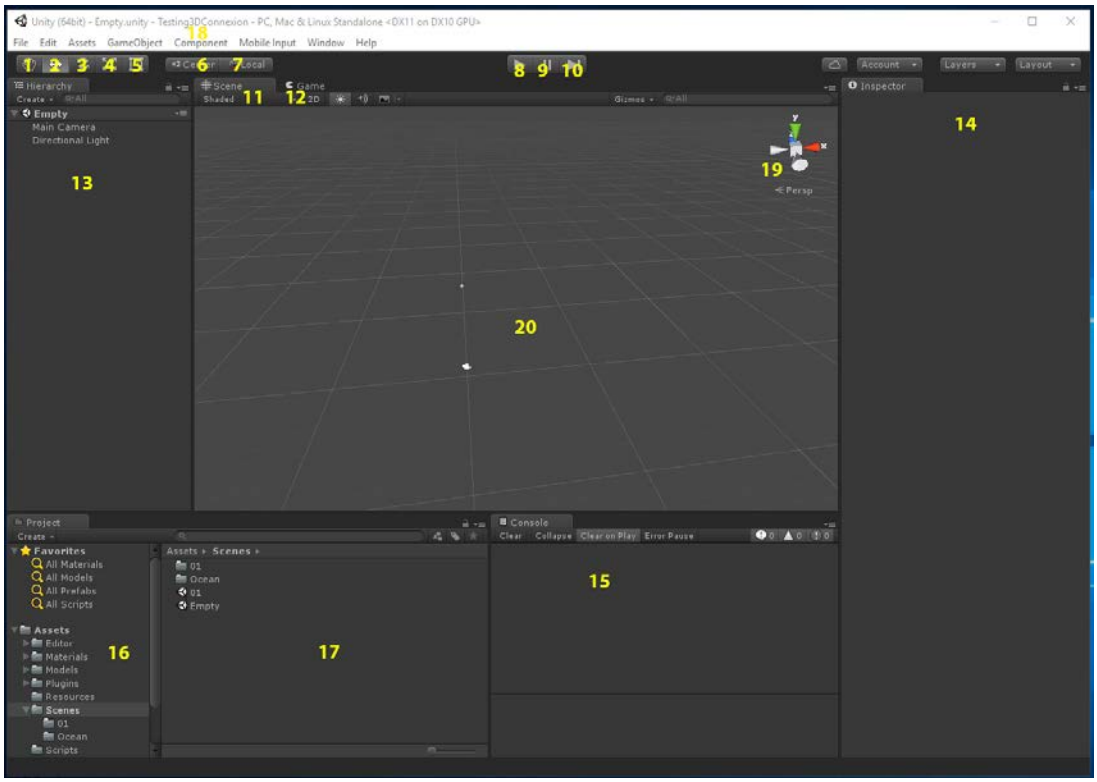
- Unity3D, τρισδιάστατη μηχανή παιχνιδιών (Game Engine).
- Autodesk 3DStudioMax, πρόγραμμα δημιουργίας και διαχείρισης τρισδιάστατων γεωμετριών.

3.2.Unity3D

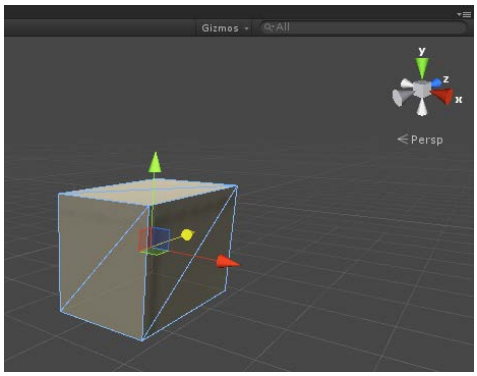
Το Unity3D είναι μηχανή παιχνιδιών (Game Engine) η οποία χρησιμοποιεί για την γραφική απεικόνιση τον αντικειμένων πολύγωνα (polygons), σε αντίθεση με τα CAD προγράμματα τα οποία χρησιμοποιούν NURBS. Η ένωση των πολυγώνων μεταξύ τους μας δίνει την οπτική ψευδαίσθηση ενός συμπαγούς γεωμετρικού αντικειμένου ενώ ο αριθμός τους αποτελεί κριτήριο για την ποιότητα της επιφάνειας των αντικειμένων αυτών. Όσο μεγαλύτερος είναι ο αριθμός των πολυγώνων τόσο πιο ομαλή θα είναι και η επιφάνεια και το αντίθετο. Στην εικόνα που ακολουθεί βλέπουμε δύο σφαίρες η μία έχει 19802 πολύγωνα ενώ η άλλη 182 κάτι το οποίο έχει αντίκτυπο στην ποιότητα της επιφάνεια.



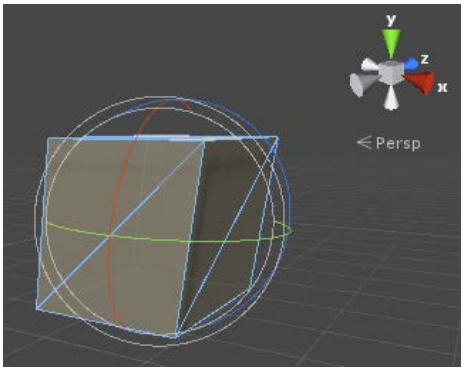
Την πρώτη φορά που ανοίγουμε το Unity3D βλέπουμε ένα αρκετά " καθαρό " γραφικό περιβάλλον το οποίο μας δίνει την δυνατότητα να εργαστούμε άμεσα και ταχύτατα. Στην εικόνα που ακολουθεί, γίνεται εν συντομία περιγραφή των βασικών μενού και παραθύρων του προγράμματος ώστε ο αναγνώστης να γνωρίσει τις βασικές λειτουργίες και την φιλοσοφία του προγράμματος.



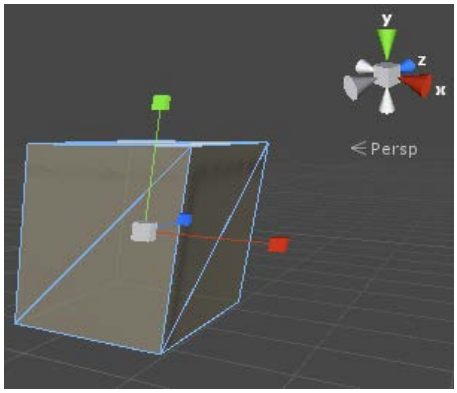
1. Pan, μας επιτρέπει την μετακίνηση σε δύο διαστάσεις του κύριου παραθύρου (20), σε σχέση με τα αντικείμενα του εικονικού κόσμου, όταν βρισκόμαστε στην καρτέλα Scene δηλαδή σε λειτουργία επεξεργασίας (Edit Mode).
2. Translate, κατά την επιλογή ενός αντικειμένου με αριστερό κλικ πάνω του και την επιλογή Translate μας δίνεται η δυνατότητα μετακίνησης σε τρείς διαστάσεις του αντικειμένου αυτού.



3. Rotate, κατά την επιλογή ενός αντικειμένου με αριστερό κλικ πάνω του και την επιλογή Rotate μας δίνεται η δυνατότητα περιστροφής σε τρείς διαστάσεις.

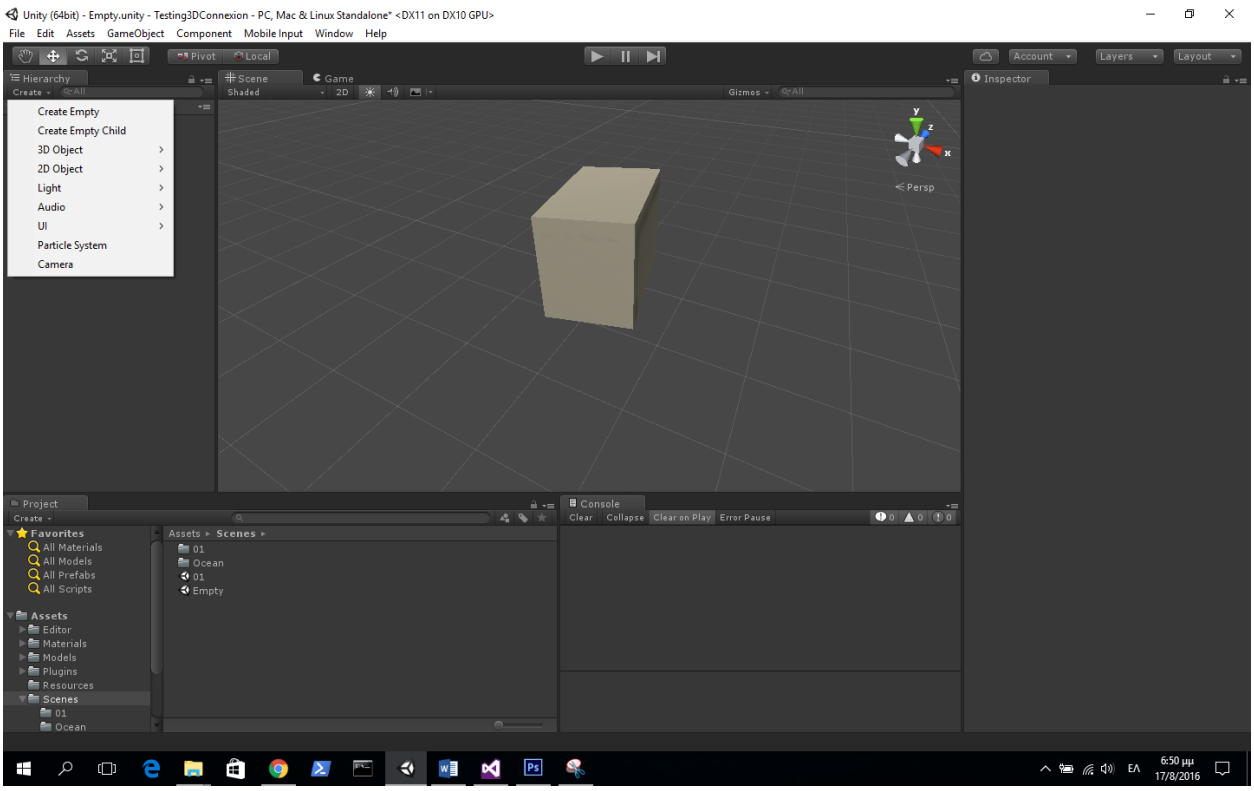


4. Scale, κατά την επιλογή ενός αντικειμένου με αριστερό κλικ πάνω του και την επιλογή Scale μας δίνεται η δυνατότητα μεταβολής των διαστάσεων του αντικειμένου.

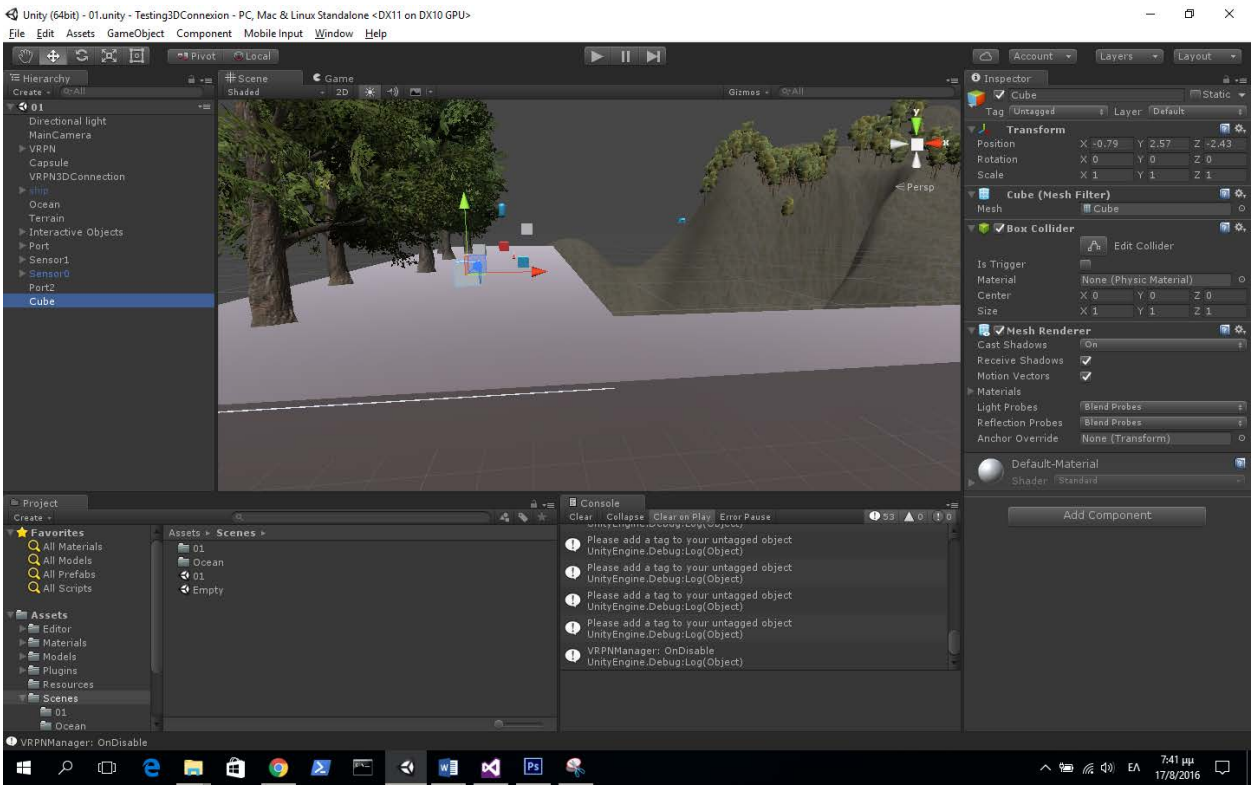


5. 2D Scale, έχει τις ίδιες ιδιότητες με το Scale η διαφορά του είναι ότι η μεταβολή των διαστάσεων γίνεται σε επίπεδο και όχι στον τρισδιάστατο χώρο.

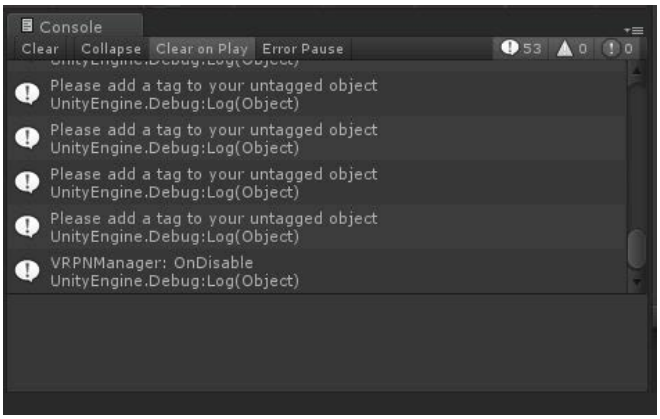
6. Center - Pivot, το εργαλείο λαβής (handle tool) τοποθετείται στο κέντρο του αντικειμένου (Center) ή στο κέντρο του τρισορθογώνιου συστήματος αξόνων (Pivot Point), τα οποία μπορεί να συμπίπτουν μπορεί και όχι.
7. Local - Global, η μετακίνηση των αντικειμένων γίνεται αναφορικά με το κέντρο όγκου τους (Local) ή σύμφωνα με το κέντρο του εικονικού κόσμου (Global).
8. Play, με το πρώτο πάτημα του κουμπιού ξεκινάει η εκτέλεση της εφαρμογής μας ενώ στο δεύτερο πάτημα σταματάει.
9. Pause, γίνεται μερικώς παύση της εφαρμογής.
10. Step, με το πάτημα του κουμπιού η εφαρμογή μας μεταβαίνει στο επόμενο καρέ (frame). Είναι ένα χρήσιμο εργαλείο ώστε να μπορούμε να έχουμε οπτικό έλεγχο της εξέλιξης διάφορων μεταβατικών καταστάσεων στην εφαρμογή μας και λειτουργεί εφόσον έχουμε πρώτα πατήσει το κουμπί Play.
11. Scene, είναι το παραθυρικό περιβάλλον κατά το οποίο μπορούμε να επεξεργαστούμε την εφαρμογή μας σε κάθε στάδιο (Edit Mode).
12. Game, είναι το παραθυρικό περιβάλλον κατά το οποίο γίνεται εκτέλεση της.
13. Hierarchy, είναι η καρτέλα όπου έχουμε οπτική επαφή με όλα τα αντικείμενα των οποίων μπορούμε να αλλάξουμε σειρά και όνομα, να δημιουργήσουμε βασικές γεωμετρίες (2D και 3D Objects), φωτισμό (Light), ηχητικές πηγές (Audio), κάμερες (Camera), σύστημα σωματιδίων (Particle System) για διάφορα οπτικά εφέ και τέλος διαδραστικό περιβάλλον χρήστη (User Interface, UI).



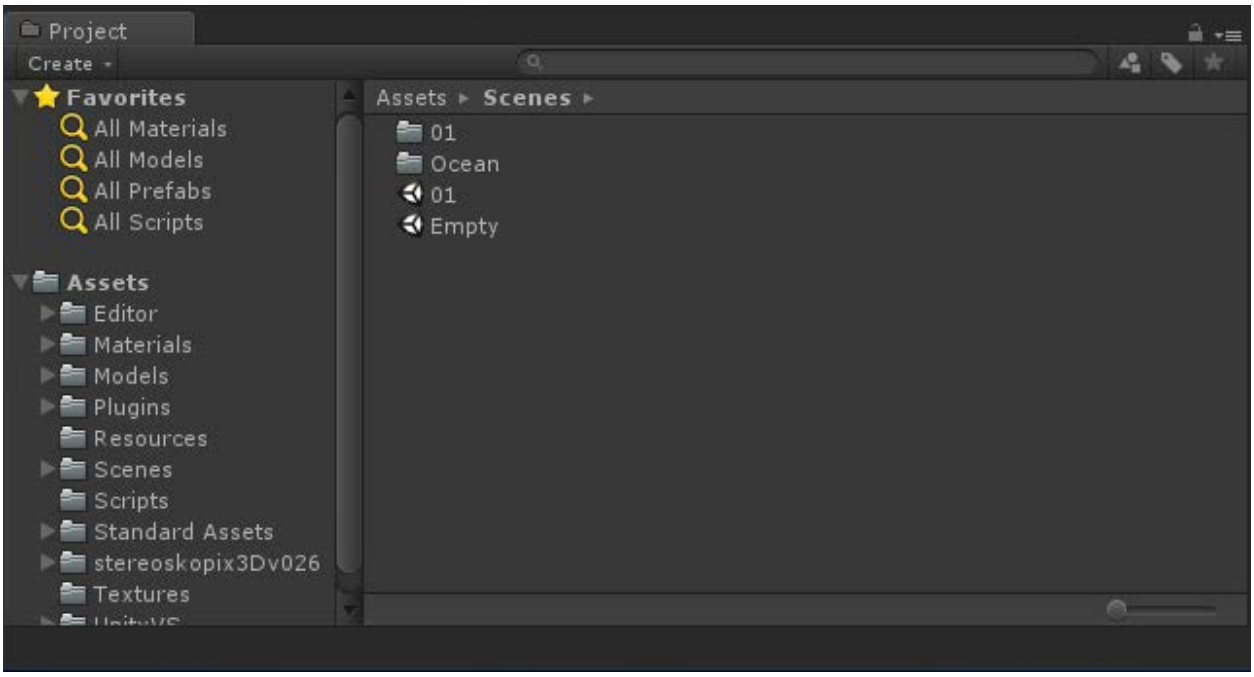
14. Inspector, η καρτέλα αυτή είναι “ ενεργή ”, αλλάζει το μενού της ανάλογα με το αντικείμενο που έχουμε επιλέξει και μας δίνει πληροφορίες για τις ιδιότητές του. Στην παρακάτω εικόνα έχουμε επιλέξει ένα κύβο (Cube) και στην καρτέλα Inspector διακρίνονται όλα τα “ συστατικά ” του (Components).



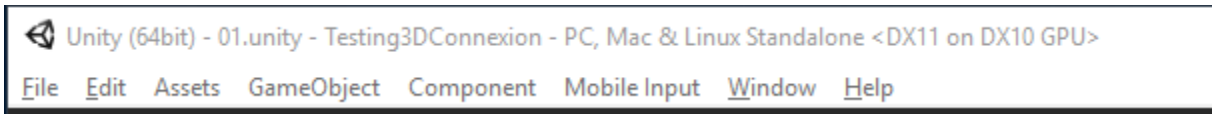
15. Console, στην κονσόλα λαμβάνουμε πληροφορίες που αφορούν την εφαρμογή μας σε μορφή κειμένου.



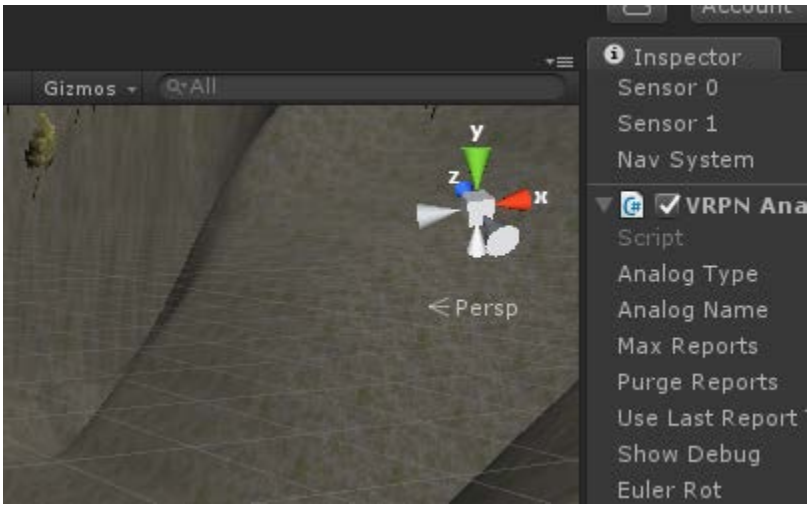
16. Project, η καρτέλα αυτή περιέχει τον κύριο φάκελο Assets όπου μέσα του περιέχονται όλοι οι υποφάκελοι και τα αρχεία της εφαρμογής μας.
17. Project details, έχουμε λεπτομερή οπτικό έλεγχο των επιλεγμένων αρχείων, επίσης με δεξί κλικ μέσα στο παράθυρο αυτό μπορούμε να δημιουργήσουμε νέα αρχεία.



18. Main Menu, είναι το κυρίως μενού στο οποίο βρίσκουμε όλες τις επιμέρους εφαρμογές του Unity3D συγκεντρωμένες.



19. Navigation Gizmo, είναι βοηθητική πυξίδα που μας επιτρέπει τον οπτικό έλεγχο σε 360 μοίρες της σκηνής (Scene) στην οποία εργαζόμαστε.

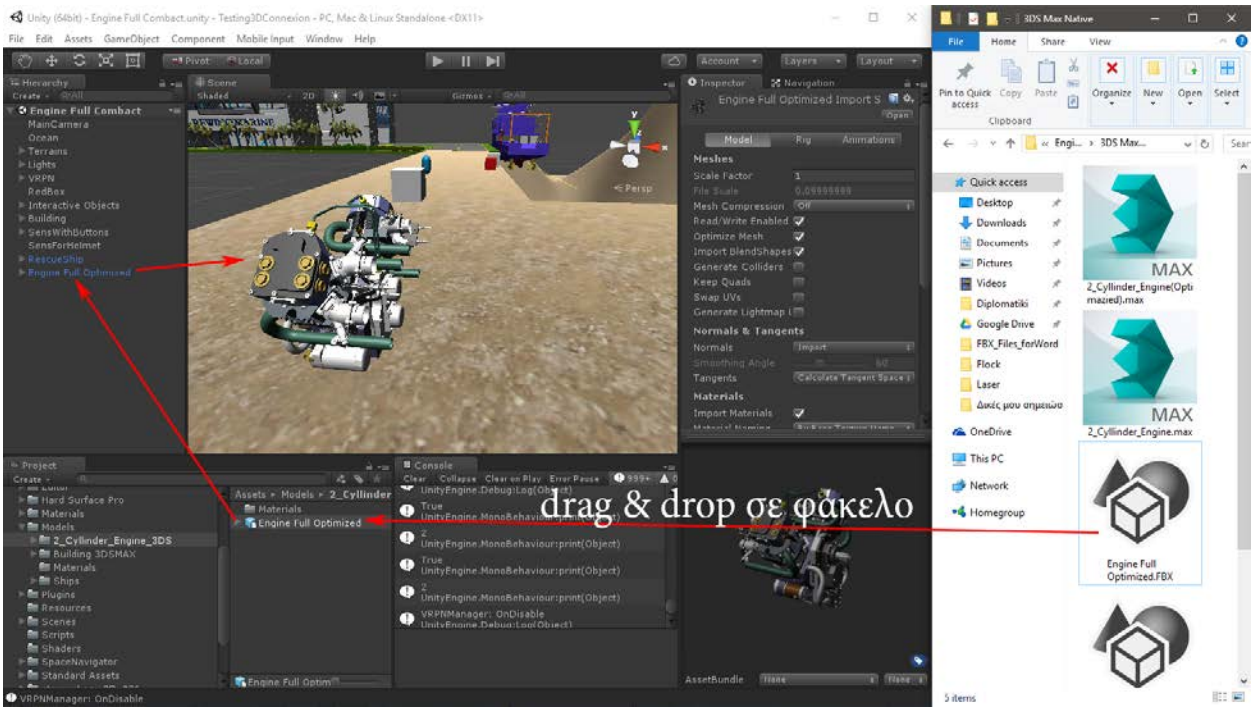


20. Main Window, είναι το κυρίως παράθυρο στο οποίο εργαζόμαστε.

Στο Unity3D μπορούμε να εισάγουμε απλές γεωμετρίες και να δημιουργήσουμε έδαφος (Terrain) με εργαλεία που παρέχονται από το ίδιο το πρόγραμμα όπως αναφέραμε παραπάνω. Η απαίτηση για πιο περίπλοκα αντικείμενα μας αναγκάζει να κάνουμε εισαγωγή μοντέλων μέσω προγραμμάτων CAD εφόσον έχει γίνει κατάλληλη μετάφρασή τους σε αρχείο τύπου *.FBX. Έχοντας δημιουργήσει τα αρχεία που χρειαζόμαστε, δημιουργούμε για την οργάνωσή τους, με δεξί κλικ ένα φάκελο στο Project window του Unity3D και του δίνουμε την ονομασία Models. Τα μοντέλα που θα εισάγουμε θα περιέχονται το κάθε ένα σε ξεχωριστό δικό του υποφάκελο οπότε δημιουργούμε άλλους τρείς φακέλους όσα είναι και τα μοντέλα μας, με τις ονομασίες :

- 2_Cyllinder_Engine_3DS
- Building 3DSMAX
- Ships

Στην συνέχεια κάνουμε drag & drop τα αρχεία *.FBX από τον File Manager των windows στους αντίστοιχους φακέλους που δημιουργήσαμε. Το Unity3D επεξεργάζεται τα αρχεία αυτά, ενώ για να φανούν στο παράθυρο Scene πρέπει να τα κάνουμε drag & drop στο παράθυρο Hierarchy.



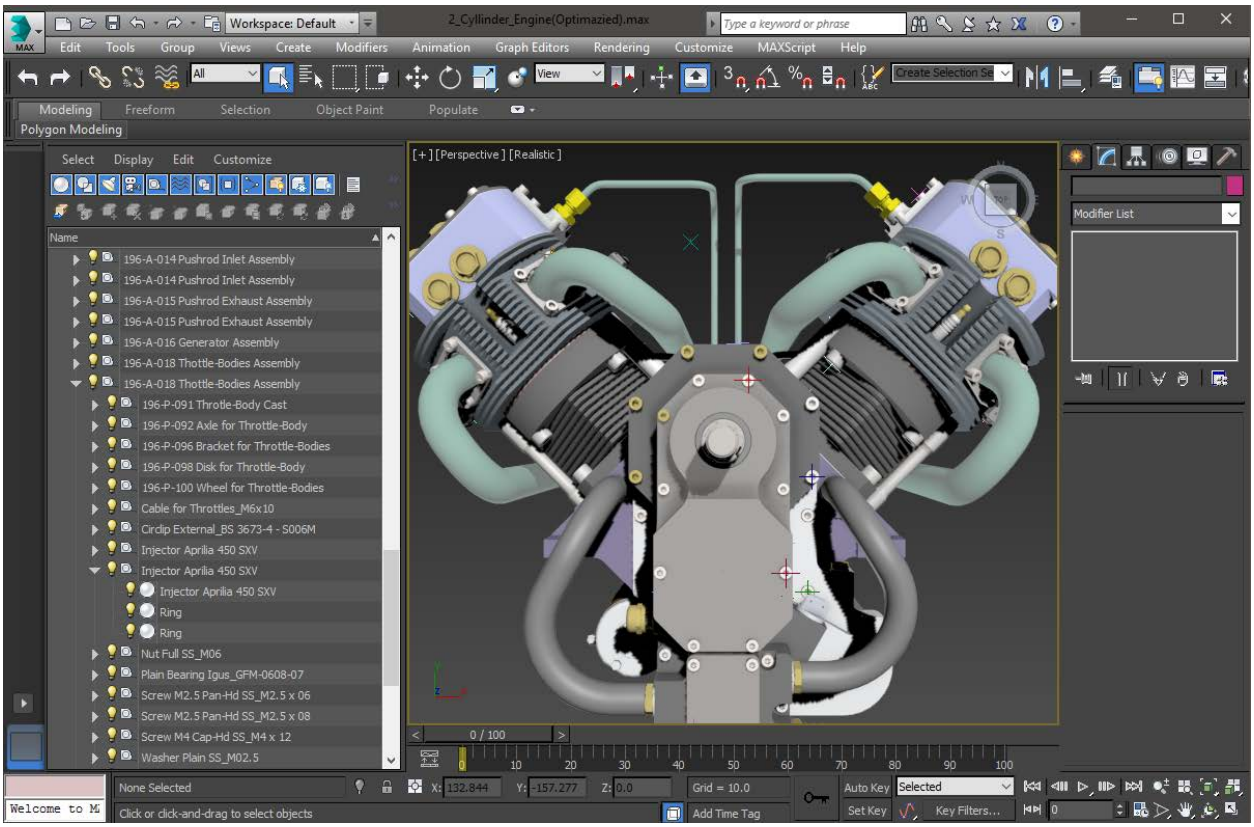
Μετά την εισαγωγή των γεωμετριών, το Unity3D μας επιτρέπει την μετακίνηση, περιστροφή αλλά και την αλλαγή των διαστάσεών τους ώστε να διαμορφώσουμε τον εικονικό κόσμο όπως εμείς θέλουμε.

3.3.3D Studio Max

Το 3D Studio Max είναι ένα πρόγραμμα της εταιρίας Autodesk το οποίο χρησιμοποιείται για δημιουργία τρισδιάστατων μοντέλων, την κινηματική απεικόνιση και τον φωτορεαλισμό τους. Οι δυνατότητες του προγράμματος είναι αρκετά περίπλοκες, εμείς βέβαια θα επικεντρωθούμε μόνο σε αυτές που μας ενδιαφέρουν. Στο 3D Studio Max μπορούμε να κάνουμε εισαγωγή γεωμετριών από διάφορα προγράμματα CAD και εξαγωγή σε μορφή αρχείου *.FBX, ως παράδειγμα εισάγουμε το μοντέλο του δικύλινδρου κινητήρα με ονομασία **engine-v-twin-4-valve-heads** το οποίο έχουμε κατεβάσει από την

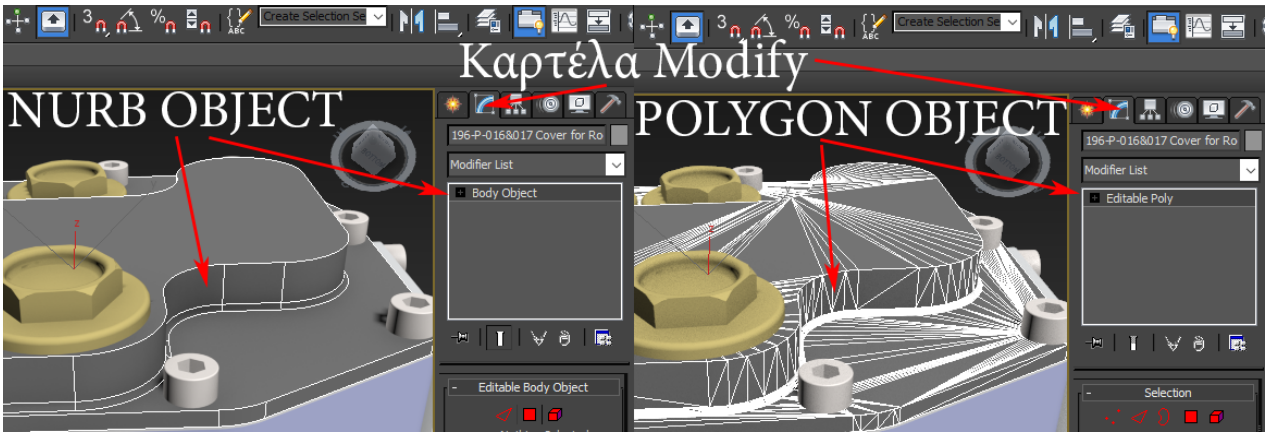
ιστοσελίδα https://grabcad.com/ και συγκεκριμένα από τον χρήστη Aled J Taylor το οποίο είναι εγγενές αρχείο του προγράμματος Solidworks.

Κατά την εισαγωγή του παραπάνω αρχείου το 3D Studio Max κάνει τους απαραίτητους μετασχηματισμούς, μέσω του υπο-προγράμματος Autodek Direct Connect, ώστε να μεταφράσει την γεωμετρία σε μορφή κατάλληλη για περαιτέρω επεξεργασία μέσα από το δικό του περιβάλλον.

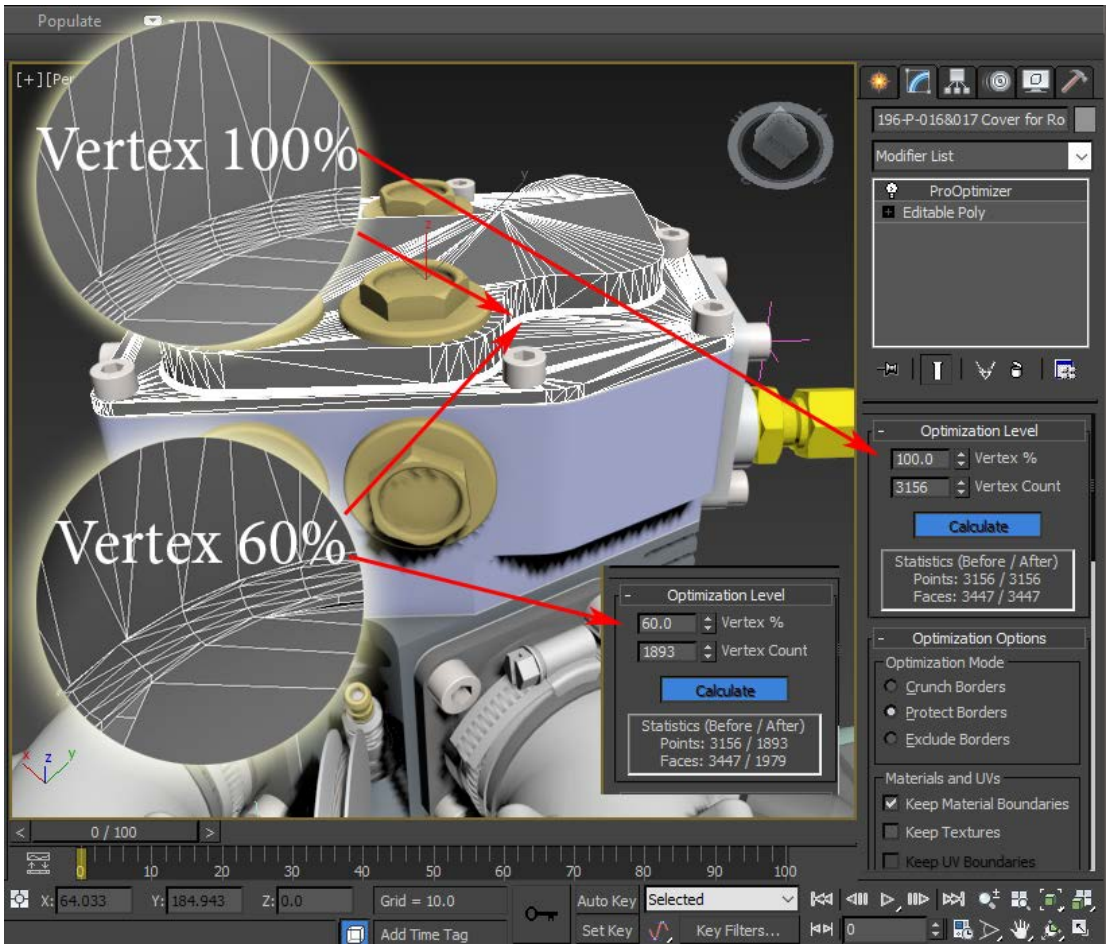


Το επόμενο βήμα είναι να επεξεργαστούμε τα επιμέρους δομικά στοιχεία που αποτελούν τον κινητήρα**.** Συγκεκριμένα μας ενδιαφέρει να κάνουμε βελτιστοποίηση στον αριθμό των πολυγώνων και να αλλάξουμε τον προσανατολισμό των συστημάτων αναφοράς (Pivot Points).

Σαν παράδειγμα για την βελτιστοποίηση του αριθμού των πολυγώνων επιλέγουμε το καπάκι της κεφαλής των βαλβίδων ενός από τους κυλίνδρου και την καρτέλα Modify στην οποία αναφέρεται το Body Object. Με δεξί κλικ στην ονομασία Body Object επιλέγουμε Editable Poly οπότε αλλάζει ο τρόπος αναπαράστασης του στοιχείου, μετατρέπεται δηλαδή από Nurbs (Non-uniform rational B-spline) σε Polygons όπως διακρίνεται και στην εικόνα που ακολουθεί.

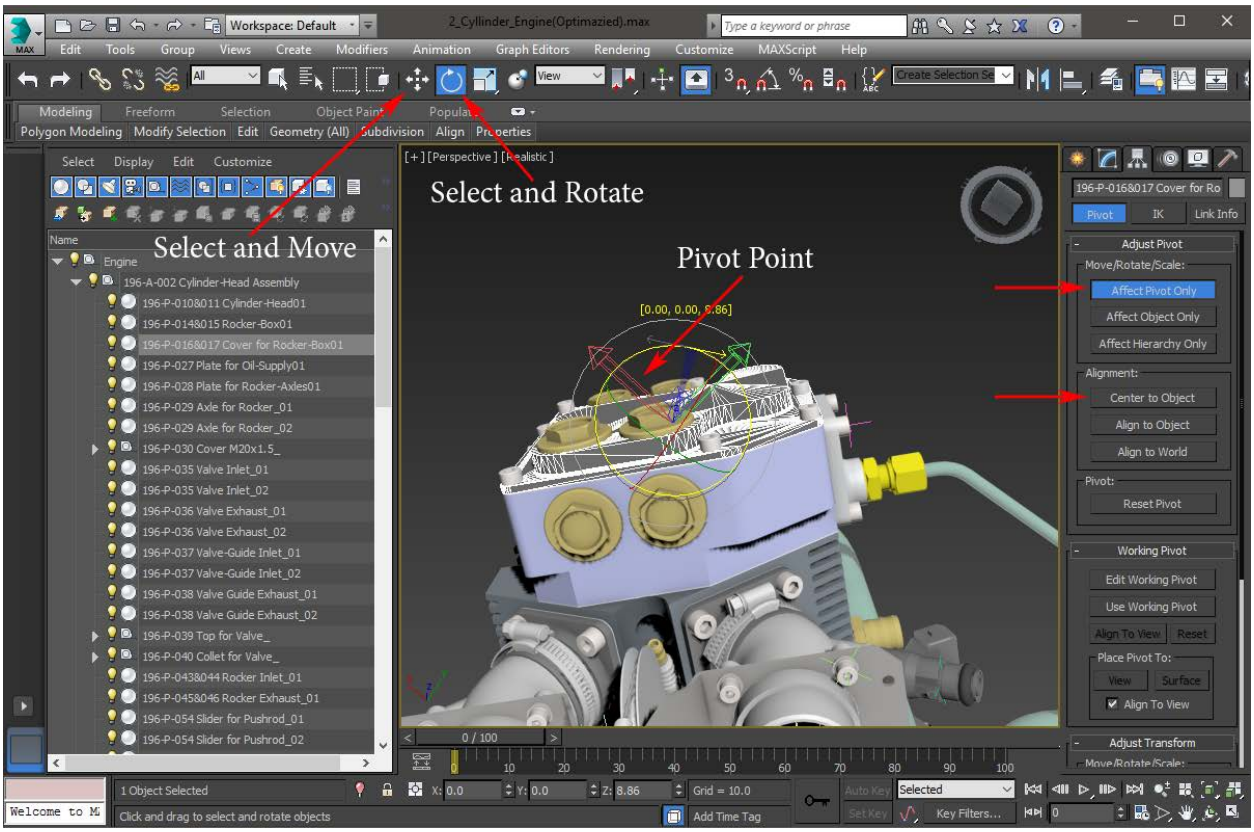


Στην συνέχεια από τον αναδυόμενο κατάλογο Modifier List επιλέγουμε ProOptimizer και κάνουμε κλικ στο κουμπί Calculate ώστε να μας υπολογίσει τον αριθμό των σημείων και επιφανειών από τα οποία αποτελείται η μετασχηματισμένη πλέον γεωμετρία του καπακιού. Αν δεν χρειαζόμαστε μεγάλη λεπτομέρεια στην απεικόνιση των αντικειμένων μπορούμε να μεταβάλλουμε το ποσοστό % στο μενού Optimization Level προσέχοντας ταυτόχρονα να μην μειώσουμε αισθητά των αριθμό και χάσουμε γεωμετρική λεπτομέρεια. Στην παρακάτω εικόνα έχουμε κάνει μείωση 40% των πολυγώνων δηλαδή από 3156 έχουμε κάνει μείωση σε 1893 πολύγωνα έτσι μετά από μεγέθυνση σε κάποια περιοχή μπορούμε να διακρίνουμε τις αλλαγές στην ποιότητα της επιφάνειας μεταξύ των δύο τιμών.



Η αλλαγή του αριθμού των πολυγώνων σε μια επιφάνεια έχει άμεσο αντίκτυπο στην τελική επίδοση του προγράμματος Unity3D. Ο κινητήρας που επεξεργαζόμαστε αποτελείται από περίπου 1060 δομικά στοιχεία τα οποία με την σειρά τους διαμορφώνονται από δεκάδες πολύγωνα, έτσι καταλαβαίνουμε πως ο αριθμός των πολυγώνων μπορεί να αντιστοιχεί σε δεκάδες εκατομμύρια για μια γεωμετρική συναρμογή όπως ο κινητήρας μας. Η χρήση πολλών τέτοιων γεωμετριών σε ένα πρόγραμμα που αποδίδει τα γραφικά σε πραγματικό χρόνο, όπως είναι το Unity3D, καταλαβαίνουμε πως απαιτεί μεγάλη υπολογιστική ισχύ από τον κύριο επεξεργαστή CPU αλλά και από την μονάδα επεξεργασίας γραφικών GPU, έτσι πρέπει να ακολουθούμε την τακτική της βελτιστοποίησης σχεδόν σε όλες τις περίπλοκες γεωμετρίες που θέλουμε να εισάγουμε για την δημιουργία του εικονικού κόσμου.

Το σύστημα αναφοράς (Pivot Point) κάθε δομικού στοιχείου του κινητήρα είναι άλλο ένα σημαντικό κομμάτι που πρέπει να επεξεργαστούμε μέσω του 3D Studio Max. Η αλλαγή του συστήματος αυτού μας βοηθάει στην μετακίνηση και περιστροφή των αντικειμένων όταν αυτά εισαχθούν στο Unity3D. Σαν παράδειγμα θα χρησιμοποιήσουμε και πάλι το καπάκι της κεφαλής των βαλβίδων. Επιλέγουμε λοιπόν το ανωτέρω δομικό στοιχείο του κινητήρα και μεταβαίνουμε στην καρτέλα Hierarchy του 3D Studio Max. Στο μενού Adjust Pivot επιλέγουμε Affect Pivot Only ενώ στο Alignment κάνουμε κλικ στην επιλογή Center to Object. Έτσι έχουμε μετακινήσει το σύστημα αναφοράς (Pivot Point) στο κέντρο όγκου του αντικειμένου. Το επόμενο βήμα, αν είναι απαραίτητο, είναι να μετακινήσουμε ή και να περιστρέψουμε το Pivot Point με την βοήθεια των εργαλείων Select and Move και Select and Rotate όπως διακρίνεται στην παρακάτω εικόνα.



4. Η χρηστικότητα του VRPNServer και VRPNWrapper στο Unity3D

4.1. VRPNServer

Έχοντας πλέον κτίσει όλα τα επιμέρους προγράμματα του VRPNServer είμαστε σε θέση να δούμε την χρηστικότητά του και τον τρόπο εφαρμογής και διασύνδεσης με το Unity3D. Όπως προαναφέραμε ο VRPNServer αποτελείται από τα παρακάτω τέσσερα προγράμματα και ένα αρχείο με την ονομασία **vrpn.cfg** τα οποία έχουμε τοποθετήσει σε έναν φάκελο στον σκληρό μας δίσκο.

- **vrpn_print_devices.exe**
- **vrpn_print_performance.exe**
- **vrpn_print_messages.exe**
- **vrpn_server.exe**

Πρέπει σε αυτό το σημείο να σημειώσουμε ότι η διαδρομή που βρίσκεται ο φάκελος αυτός δεν θα πρέπει να περιέχει ελληνικούς χαρακτήρες διότι υπάρχει η πιθανότητα να μην δουλέψει σωστά ή και καθόλου η εκτέλεση των προγραμμάτων.

Πρώτη μας δουλειά λοιπόν είναι να ανοίξουμε το αρχείο **vrpn.cfg** με την βοήθεια του Notepad.exe των Windows ή με ένα πιο εξελιγμένο πρόγραμμα όπως το Notepad.exe ++ το οποίο μπορούμε να κατεβάσουμε δωρεάν και να εγκαταστήσουμε από την διεύθυνση https://notepad-plus-plus.org/. Στις αρχικές γραμμές του αρχείου αυτού βλέπουμε την έκδοση του VRPNServer, στην περίπτωσή μας αναφέρεται η έκδοση 7.34 και συγκεκριμένα **# vrpn.cfg SAMPLE for VRPN version 07.34**. Διαβάζοντας τις οδηγίες χρήσης καταλαβαίνουμε ότι πρόκειται για το αρχείο διαχείρισης των εξωτερικών συσκευών εικονικής πραγματικότητας με την βοήθεια του οποίου μπορούμε να ενεργοποιήσουμε, απενεργοποιήσουμε καθώς και να ρυθμίσουμε τις παραμέτρους και τον τρόπο επικοινωνίας του υπολογιστή μας με τις συσκευές αυτές.

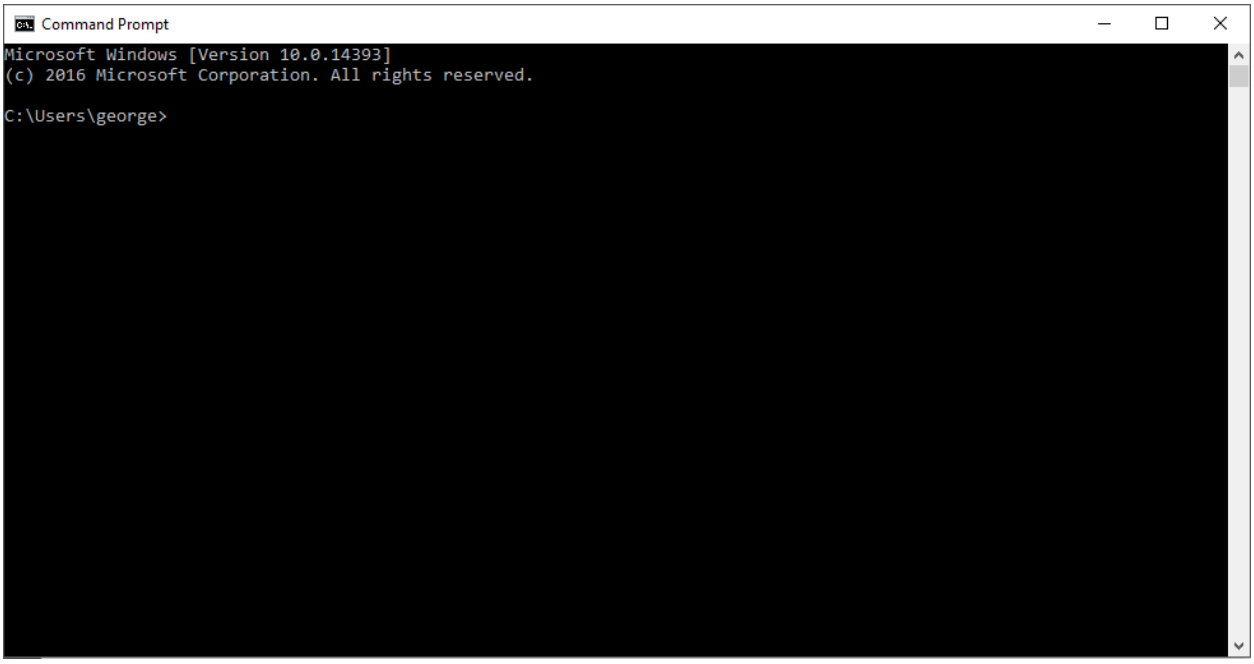
Σαν παράδειγμα θα αναφέρουμε τις ρυθμίσεις που πρέπει να κάνουμε για το ποντίκι και το πληκτρολόγιο, κοινές συσκευές που διαθέτει ο καθένας μας οι οποίες επιτρέπουν έναν αρχικό έλεγχο σωστής λειτουργία του VRPNServer. Στις γραμμές του αρχείου **vrpn.cfg** 2070 έως 2080 έχουμε αναφορά για την συσκευή του ποντικιού ενώ στις γραμμές 2081 έως 2090 για το πληκτρολόγιό μας. Για την επικοινωνία λοιπόν του VRPNServer με το ποντίκι και το πληκτρολόγιο αφαιρούμε μπροστά από την εκάστοτε γραμμή το σύμβολο της δίεσης **#** όπως διακρίνεται στο κείμενο που ακολουθεί με τα κόκκινα γράμματα.

```
##############################################################################
# Open the mouse as an analog and button devices.  There is an implementation
# under Windows and another under Linux (using GPM).  There are two analog
# channels, reporting in the range [0..1] as the mouse moves across the screen.
# There are 3 button channels: left, middle, right.
#
# There is one argument:
#       char    name_of_this_device[]
```

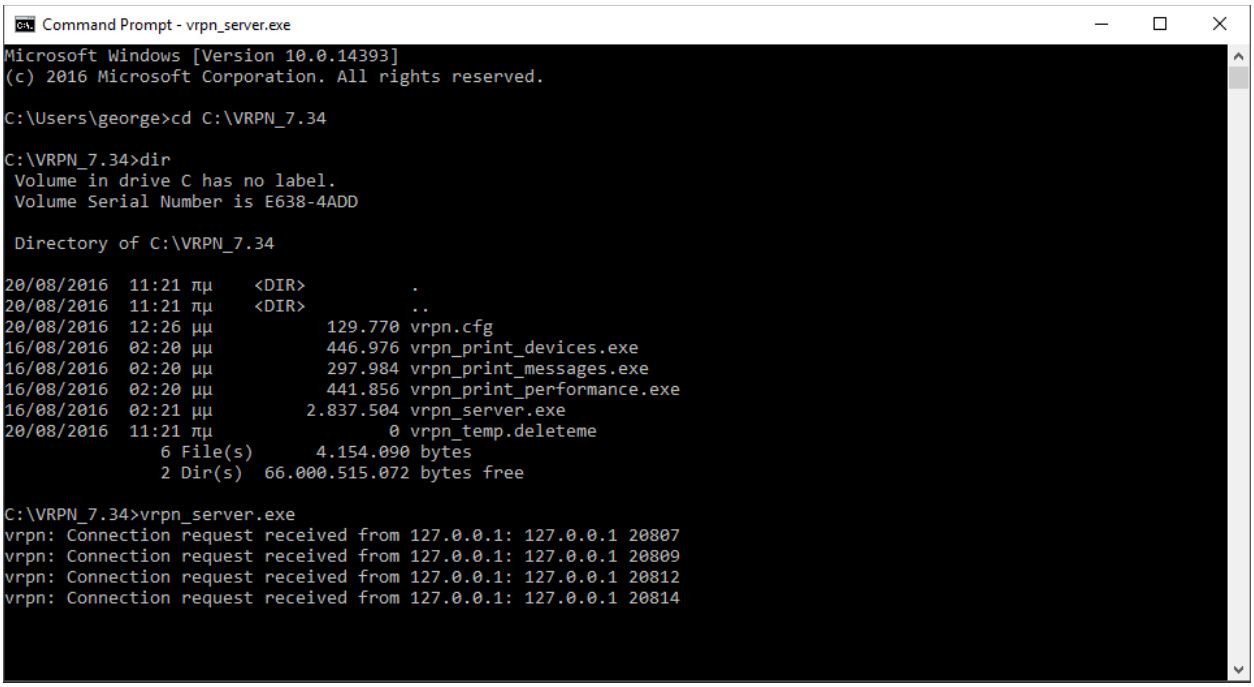
```
vrpn_Mouse          Mouse0

###############################################################################

# Open the keyboard as a button device.  There is an implementation

# under Windows.  There are 256 buttons, and they are triggered according

# to the scan code that they represent.

#

# There is one argument:

#         char        name_of_this_device[]

vrpn_Keyboard       Keyboard0
```

Τώρα είμαστε σε θέση να ξεκινήσουμε το κυρίως πρόγραμμα **vrpn_server.exe** αλλά και τα υπόλοιπα βοηθητικά προγράμματα **vrpn_print_devices.exe**, **vrpn_print_messages.exe**, **vrpn_print_performance.exe** του VRPNServer.

Πρώτο βήμα είναι να ανοίξουμε ένα πρόγραμμα των Windows με την ονομασία cmd.exe (command prompt).



Στην συνέχεια μεταβαίνουμε μέσω του **cmd.exe** στην διαδρομή του φακέλου στον οποίο έχουμε τοποθετήσει όλα τα προαναφερθέντα αρχεία με την εντολή " **cd C:\VRPN_7.34** ". Τρέχουμε την εντολή " **dir** " ώστε να διαβάσουμε των κατάλογο των αρχείων και κάνουμε αντιγραφή και επικόλληση το όνομα του **vrpn_server.exe** πατώντας στην συνέχεια enter για την εκτέλεσή του. Θα εμφανιστούν διάφορα μηνύματα τα οποία ανανεώνονται όταν δεχτούν δεδομένα από τις συσκευές τα οποία μας δίνουν χρήσιμες πληροφορίες για την κατάσταση που βρίσκεται ο VRPNServer αλλά και οι συσκευές οι ίδιες.

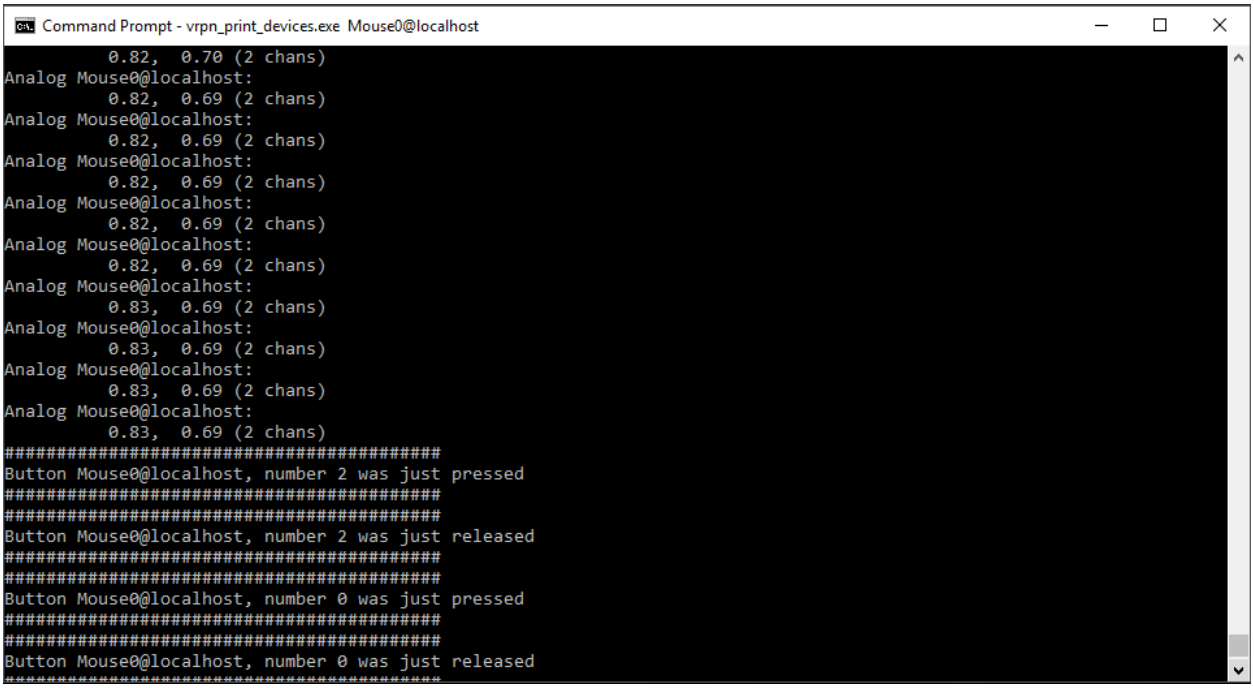


Όπως διακρίνεται στην παραπάνω εικόνα έχει γίνει επιτυχής έναρξη του VRPNServer ο οποίος στην συγκεκριμένη περίπτωση τρέχει τοπικά στον υπολογιστή κάτι που αντιλαμβανόμαστε από το μήνυμα και συγκεκριμένα από την αναφορά της διεύθυνσης **127.0.0.1 (localhost)**. Για τερματισμό του προγράμματος απλά κλείνουμε το παράθυρο **cmd.exe** ή πατάμε ” **ctrl & c** ”.

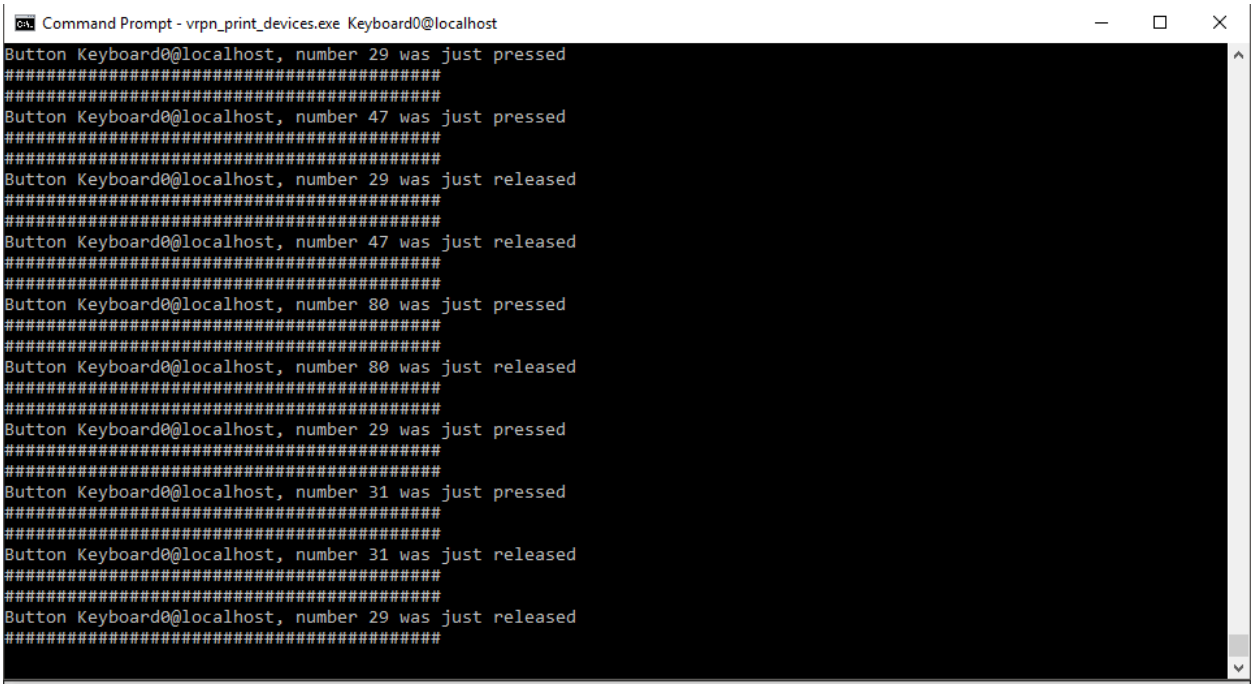
Ο VRPNServer δεν περιορίζεται στο να εκτελείται μόνο σε τοπικό επίπεδο (localhost), μπορεί να τρέξει και σε έναν απομακρυσμένο υπολογιστή αρκεί να είναι συνδεδεμένος σε δίκτυο και να είναι γνωστή και προσβάσιμη η μοναδική διεύθυνση (IP) του από τους άλλους υπολογιστές.

Για έλεγχο της επικοινωνίας της συσκευής με τον VRPNServer χρησιμοποιούμε το πρόγραμμα **vrpn_print_devices.exe** και πάλι με την βοήθεια του **cmd.exe**. Ανοίγουμε λοιπόν ένα νέο τερματικό (cmd.exe) και μεταβαίνουμε στην διαδρομή “ **cd C:\VRPN_7.34** ” εκτελώντας αυτή τη φορά την εντολή “ **vrpn_print_devices.exe Mouse0@127.0.0.1** ” ή “ **vrpn_print_devices.exe Mouse0@localhost**”. Όπως καταλαβαίνουμε υποδηλώνουμε στο πρόγραμμα **” vrpn_print_devices.exe ”** να μας υποδείξει δεδομένα για την συσκευή με ονομασία **Mouse0** η οποία είναι συνδεδεμένη στον τοπικό υπολογιστή μας με διεύθυνση **@127.0.0.1** ή **@localhost**. Αν θέλαμε να λάβουμε δεδομένα για συσκευή η οποία βρίσκεται σε απομακρισμένο υπολογιστή συνδεδεμένο στο δίκτυό μας ο οποίος επικοινωνεί με τον VRPNServer θα έπρεπε να δώσουμε εντολή “ **vrpn_print_devices.exe Mouse0@192.168.1.12** ” όπου η διεύθυνση **192.168.1.2** δεν είναι τίποτα άλλο από την διεύθυνση (IP) του υπολογιστή που εκτελείται ο διαχειριστής VRPNServer.

Στην παρακάτω εικόνα αναγράφονται πληροφορίες που αφορούν το ποντίκι του υπολογιστή μας. Συγκεκριμένα τα νούμερα αντιστοιχούν στην θέση του κέρσορα στην οθόνη, με “ **1.00, 0.00** ” να είναι οι συντεταγμένες στην πάνω δεξιά γωνία ενώ “ **0.00, 1.00** ” είναι οι συντεταγμένες στην κάτω αριστερή γωνία. Επίσης παρέχονται πληροφορίες για το ποντίκι ότι είναι αναλογική (**Analog**) συσκευή με δύο κανάλια (**2 chans**) καθώς και ο αριθμός του πλήκτρου του ποντικιού όπου πατήθηκε (**number 2 was just pressed**).



Με ακριβώς τον ίδιο τρόπο αν τρέξουμε την εντολή " **vrpn_print_devices.exe Keyboard0@127.0.0.1** " θα λάβουμε δεδομένα που αφορούν το πληκτρολόγιό μας, όπως φαίνεται στην εικόνα που ακολουθεί.

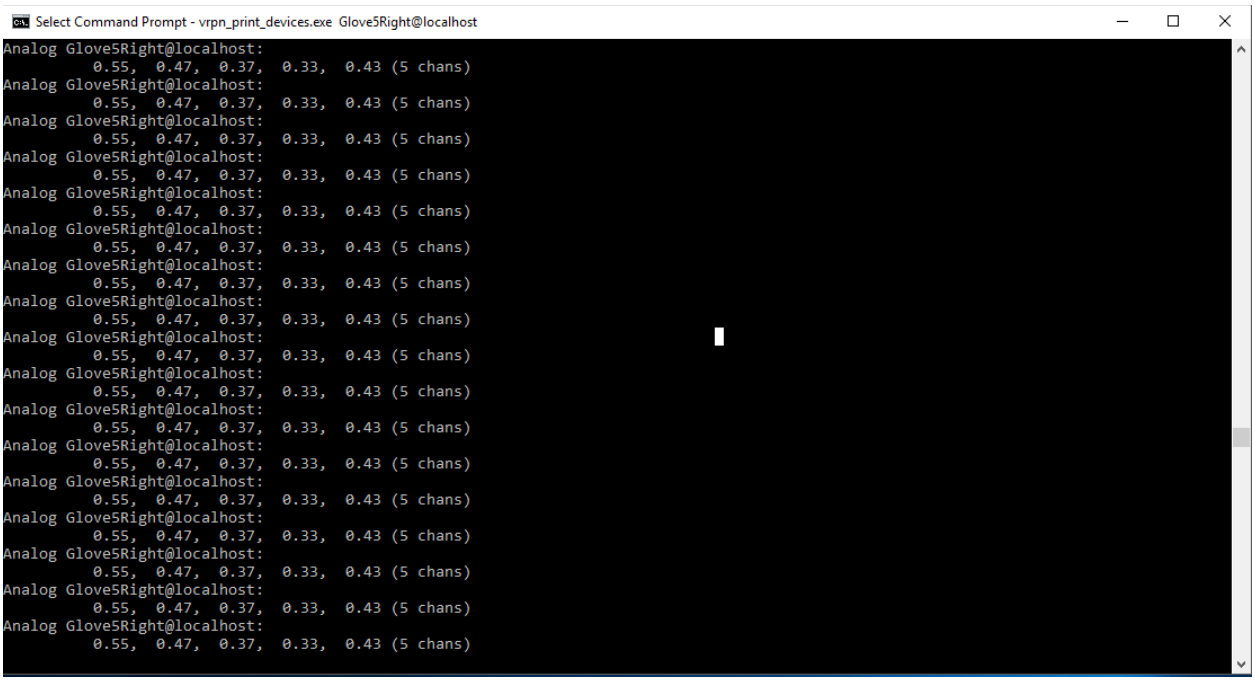


Για πιο περίπλοκες συσκευές, όσον αφορά την κατασκευή αλλά και τον τρόπο επικοινωνίας τους με τον υπολογιστή, το πρόγραμμα **vrpn_print_devices.exe** μας δίνει περισσότερες πληροφορίες. Στο εργαστήριο της σχολής μας έχουμε γάντια που χρησιμοποιούνται στην εικονική πραγματικότητα κατασκευασμένα από την εταιρία http://www.5dt.com/ τα οποία συνδέονται σε θύρα USB είτε ασύρματα μέσω της τεχνολογίας Bluetooth και διαθέτουν 5 ή 16 αισθητήρες, αναλόγως το μοντέλο.

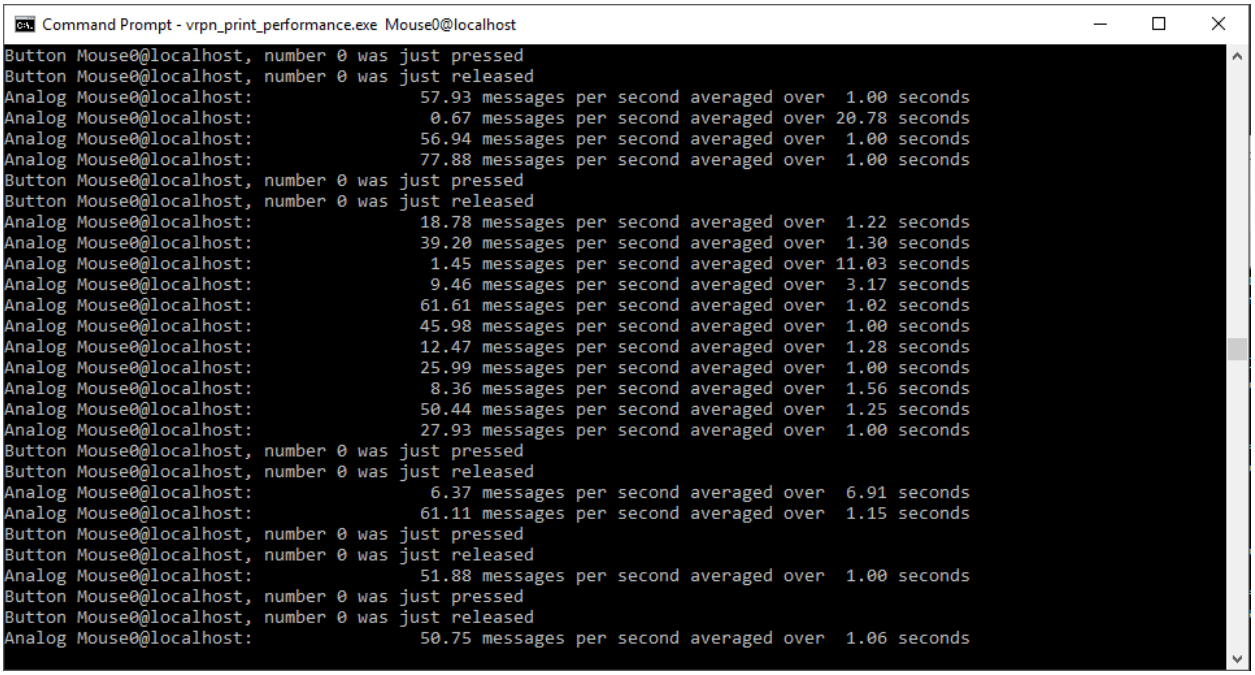
Για την ενεργοποίηση του δεξιού γαντιού στο αρχείο **vrpn.cfg** μεταβαίνουμε στην γραμμή **2275** και την ξεσχολιάζουμε αφαιρώντας το σύμβολο **#** από το **vrpn_Analog_5dtUSB_Glove5Right Glove5Right**.

```
##############################################################################
# 5DT DataGlove "Ultra" USB/USB Wireless support (based on HID)
#
# Reports 5 or 14 sensors' raw values as analogs 0-4 or 0-13, in range 0.0 - 1.0
# Note that your code will probably need to perform some scaling/calibration:
# see vrpn_Analog_5dtUSB.h for more info
#
# Four device types as shown in examples below: the server will connect to the
# first device available of that type.
#
# For serial (non-"Ultra") gloves, see vrpn_5dt and vrpn_5DT16
#
# Arguments:
#   char    name_of_this_device[]

vrpn_Analog_5dtUSB_Glove5Right Glove5Right
#vrpn_Analog_5dtUSB_Glove5Left Glove5Left
#vrpn_Analog_5dtUSB_Glove14Right Glove14Right
#vrpn_Analog_5dtUSB_Glove14Left Glove14Left

##############################################################################
```

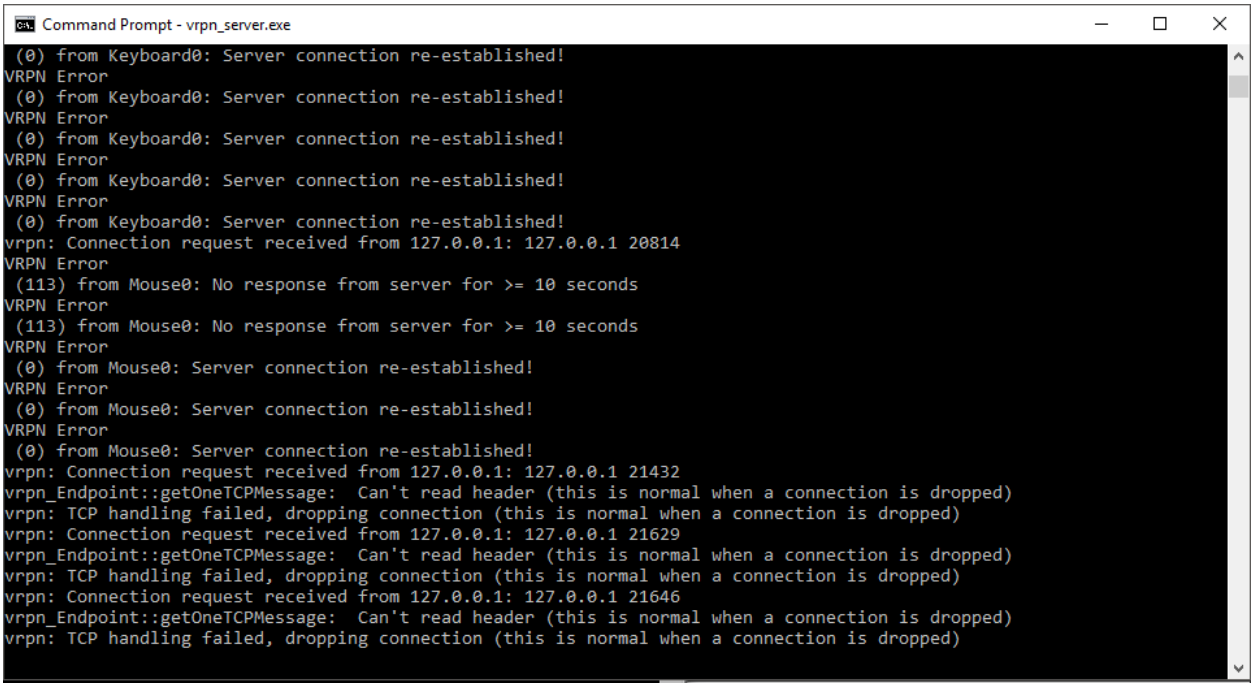
Αν τρέξουμε μέσω του **cmd.exe** την εντολή " **vrpn_print_devices.exe Glove5Right@localhost** " απεικονίζονται στο παράθυρο του τερματικού τα δεδομένα που λαμβάνουμε τα οποία μας δηλώνουν πως έχουμε μια αναλογική συσκευή με 5 κανάλια ή αισθητήρες (5 chans).



Τα προγράμματα **vrpn_print_performance.exe**, **vrpn_print_messages.exe** χρησιμοποιούνται με τον ίδιο τρόπο. Ως παράδειγμα εκτελώντας την εντολή " **vrpn_print_performance.exe Mouse0@localhost** " λαμβάνουμε δεδομένα για την απόδοση του ποντικιού, παίρνουμε δηλαδή πληροφορίες χρονικής υστέρηση που αφορούν την εκτέλεση διάφορων ενεργειών τις οποίες πραγματοποιούμε με την συγκεκριμένη συσκευή.



Τέλος πρέπει να σημειωθεί ότι αν τρέχουμε ταυτόχρονα πολλές εντολές μέσω της εφαρμογής **cmd.exe** που αφορούν τον VRPNServer μπορούν και να μην διακοπούν όλες, όταν θέλουμε να μεταβάλλουμε κάποιες ρυθμίσεις στις συσκευές μας, παρά μόνο αυτές του **vrpn_server.exe** και των συσκευών που θέλουμε να κάνουμε τις αλλαγές. Με το πέρας τον αλλαγών αν τρέξουμε ξανά την εντολή " **vrpn_server.exe** " ο VRPNServer έχει την ικανότητα να ξανασυνδεθεί με τις εν ενεργεία συσκευές όπως φαίνεται και από το μήνυμα " **Server connection-re-established** ".



4.2.VRPNWrapper

4.2.1.Εισαγωγή

Σε αυτό το κεφάλαιο θα μιλήσουμε για την εφαρμογή VRPNWrapper και πώς επιτυγχάνεται η συνδεσμολογία με τον VRPNServer και το Unity3D. Ο VRPNWrapper δεν είναι τίποτα άλλο από ένα πρόγραμμα επέκτασης (Plugin) του Unity3D και στόχος του είναι να πετύχει την επικοινωνία μεταξύ του Unity3D συνδέοντάς το με τον VRPNServer και τις εξωτερικές συσκευές εικονικής πραγματικότητας όπως είναι τα προαναφερθέντα γάντια της εταιρίας 5DT αλλά και το ηλεκτρομαγνητικό σύστημα παρακολούθησης αισθητήρων έξι βαθμών ελευθερίας (6 Degree Of Freedom) Flock Of Birds. Ένας σύντομος κατάλογος των συσκευών που υποστηρίζονται από το VRPNWrapper είναι ο ακόλουθος :

- vrpn_3DConnexion_Navigator,
- vrpn_3DConnexion_SpaceBall5000,
- vrpn_3DConnexion_SpaceExplorer,
- vrpn_3DConnexion_SpaceMouse,
- vrpn_3DConnexion_Traveler,
- vrpn_3DMicroscribe,
- vrpn_5dt,
- vrpn_5dt16,
- vrpn_Tracker_Flock,

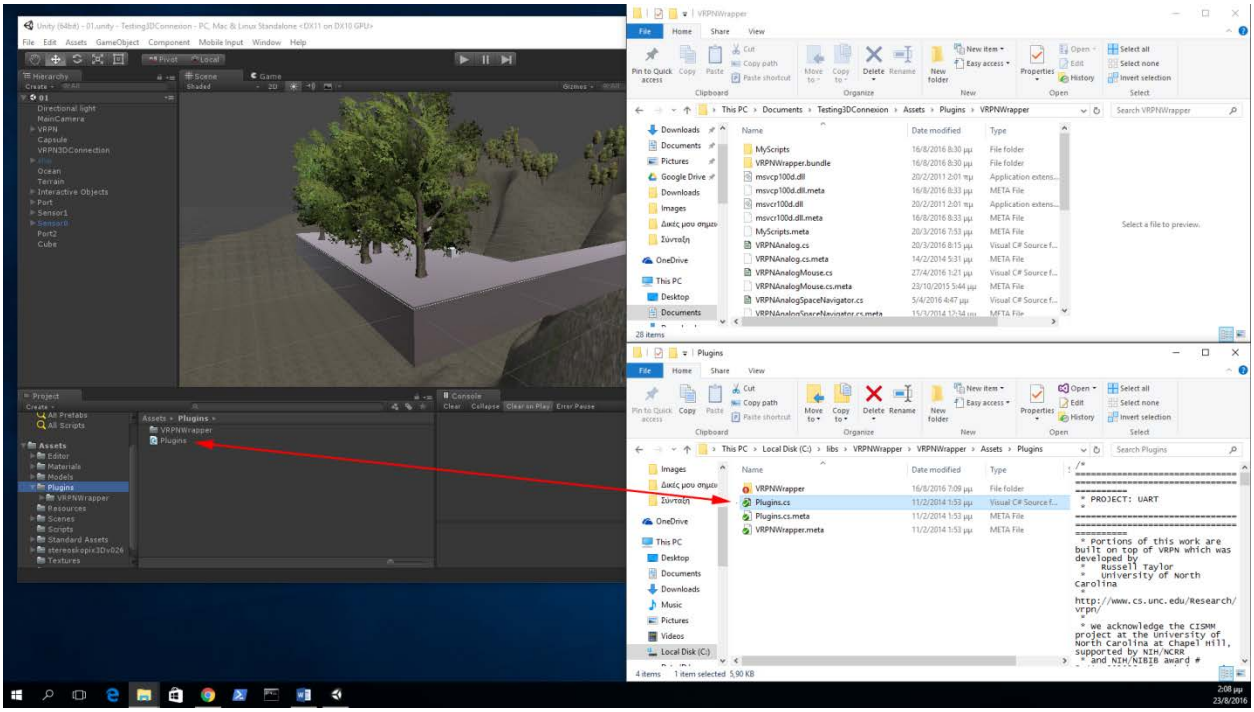
- vrpn_Tracker_Flock_Parallel,
- vrpn_Tracker_GPS,vrpn_JoyFly,
- vrpn_Joylin,
- vrpn_Joystick,
- vrpn_Joywin32,
- vrpn_Keyboard,

4.2.2.Εφαρμογή του VRPNWrapper

Κατά την έναρξη το Unity3D μας παραπέμπει να δημιουργήσουμε ένα καινούριο project επιλέγοντας το κουμπί New σε μια συγκεκριμένη διαδρομή στον σκληρό μας δίσκο. Επιλέγουμε λοιπόν το όνομα του project και την θέση του σκληρού μας δίσκου να είναι το Testing3DConnexion και C:\Users\george\Documents αντίστοιχα. Ο κύριος φάκελος λοιπόν στον οποίο βρίσκεται το όλο project βρίσκεται στην διαδρομή C:\Users\george\Documents\Testing3DConnexion.

Εφόσον εκκινήσει το πρόγραμμα στο Project window του Unity3D κάνουμε δεξί κλικ και δημιουργούμε ένα φάκελο με την ονομασία **Plugins** στον οποίο κάνουμε drag and drop το αρχείο C# **Plugins.cs** από τον File Manager των Windows στον File Manager του Unity3D. Μέσα στον φάκελο αυτό φτιάχνουμε άλλον ένα φάκελο με την ονομασία VRPNWrapper στον οποίο εισάγουμε πάλι με την μέθοδο drag and drop το αρχείο **VRPNWrapper.dll** και τα παρακάτω αρχεία C#.

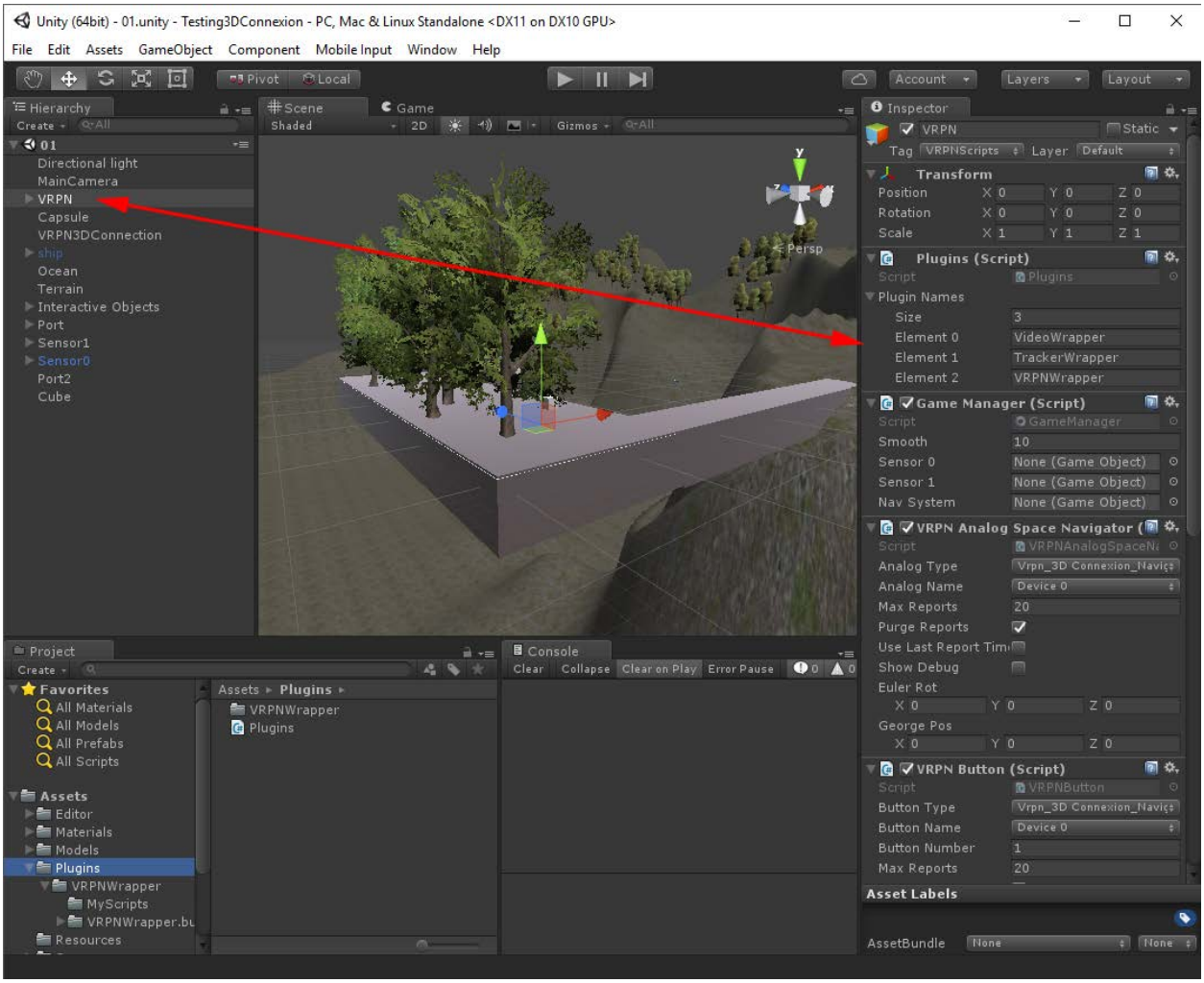
- **VRPNAnalog.cs**
- **VRPNButton.cs**
- **VRPNDeviceConfig.cs**
- **VRPNManager.cs**
- **VRPNTracker.cs**
- **VRPNTrackerCalibrate.cs**



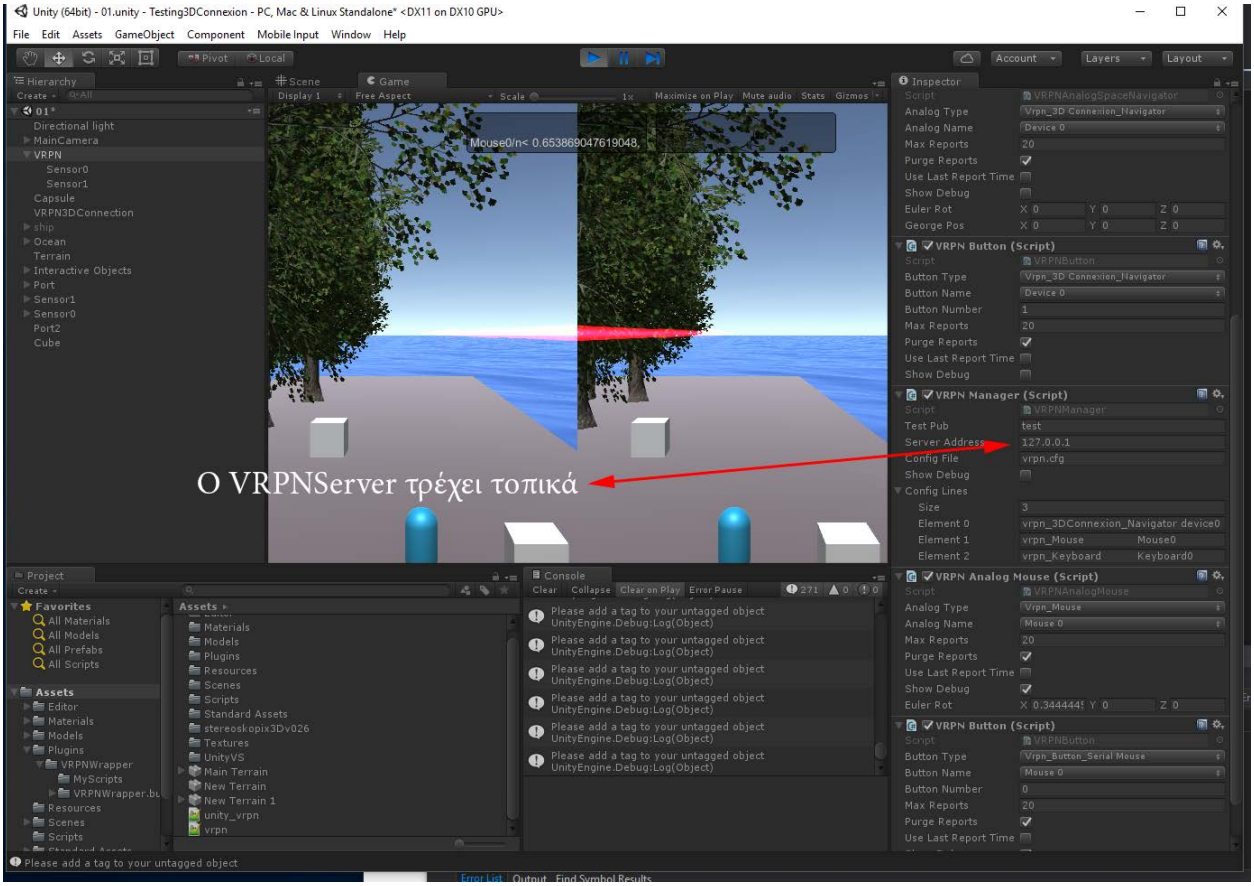
Επόμενο βήμα είναι να κάνουμε δεξί κλικ στο Hierarchy window να δημιουργήσουμε ένα κενό αντικείμενο (Create Empty) και να του δώσουμε την ονομασία VRPN ώστε να έχει μια σχετική ονοματολογία με τον VRPNServer για να μπορούμε να το ξεχωρίσουμε από τα μελλοντικά αντικείμενα (Game Objects) που θα εισάγουμε. Στο κενό αντικείμενο VRPN κάνουμε drag & drop μέσα από τον File Manager του Unity3D όλα τα παρακάτω αρχεία scripts C#:

- **VRPNAnalog.cs**
- **VRPNButton.cs**
- **VRPNDeviceConfig.cs**
- **VRPNManager.cs**
- **VRPNTracker.cs**
- **VRPNTrackerCalibrate.cs**
- **Plugins.cs**

Επιλέγοντας από το Hierarchy window το αντικείμενο VRPN θα δούμε στο Inspector windows όλα τα επιμέρους συστατικά του (Components) τα οποία σαν διακριτικό σε παρένθεση έχουν το είδος τους**,** το οποίο στην συγκεκριμένη περίπτωση είναι πρόγραμμα C# (Script).



Ο VRPNServer μπορεί να λειτουργήσει εντός του Unity3D, να κάνει δηλαδή αυτόματα εκκίνηση πατώντας το κουμπί Play, ή μπορούμε να τον εκκινήσουμε εξωτερικά εμείς. Στην πρώτη περίπτωση θα πρέπει στο VRPN αρχείο που φτιάξαμε και συγκεκριμένα στο **VRPNManager.cs** script που έχουμε επικολλήσει να αφήσουμε κενό το πεδίο **Server Address** ενώ στην δεύτερη έχουμε άλλες δύο υποπεριπτώσεις. Στην πρώτη υποπερίπτωση γίνεται εκκίνηση του VRPNServer μέσω του προγράμματος **cmd.exe** δίνοντας την εντολή " **vrpn_server.exe** ", από τον ίδιο υπολογιστή που τρέχει και το Unity3D, ενώ στο πεδίο **Server Address** του **VRPNManager.cs** script δίνουμε την διεύθυνση **127.0.0.1**. Στην δεύτερη υποπερίπτωση γίνεται εκκίνηση του VRPNServer σε έναν απομακρυσμένο υπολογιστή συνδεδεμένο σε τοπικό δίκτυο, πάλι με την βοήθεια του **cmd.exe**, όμως στο πεδίο **Server Address** του **VRPNManager.cs** script προσέχουμε να δώσουμε την διεύθυνση (IP) του υπολογιστή αυτού.



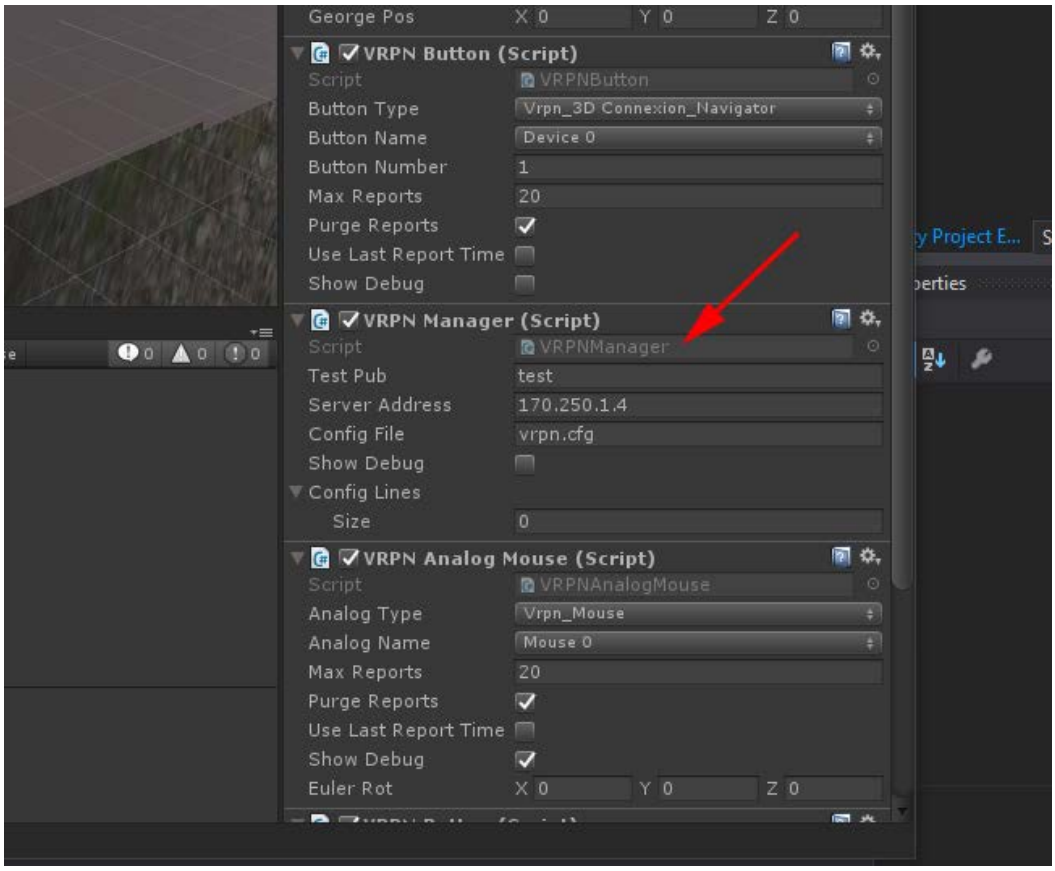
5. Διαχείριση συσκευών

5.1. Εισαγωγή

Οι εξωτερικές συσκευές που θέλουμε να συνδέσουμε στο Unity3D πρέπει να αντιμετωπιστούν με ξεχωριστό τρόπο από τον VRPNWrapper. Υπάρχουν τρείς κατηγορίες συσκευών οι οποίες διαχωρίζονται, ανάλογα με την συνδεσμολογία, επικοινωνία με τον υπολογιστή και κατασκευή τους. Έχουμε λοιπόν συσκευές οι οποίες είναι ικανές να εντοπίσουν στον χώρο την θέση και τον προσανατολισμό αισθητήρων και ονομάζονται **VRPNTracker_NameOfTheDevice**, συσκευές οι οποίες έχουν πλήκτρα (buttons) και δουλεύουν σαν διακόπτες ώστε να εκτελούν εντολές True – False οι οποίες ονομάζονται **VRPNButton_NameOfTheDevice** και αναλογικές συσκευές οι οποίες μας δίνουν δεδομένα σε πραγματικό χρόνο με την ονομασία **VRPNAnalog_NameOfTheDevice**.

Ο παραπάνω διαχωρισμός των συσκευών δεν σημαίνει ότι αυτές δεν είναι ικανές να λειτουργήσουν με πολλαπλούς τρόπους. Έχουμε λοιπόν συσκευές, όπως το απλό ποντίκι του υπολογιστή μας αλλά και το 3DConnexion Space Navigator, τις οποίες μπορούμε να δηλώσουμε σαν **VRPNAnalog** και σαν **VRPNButton** ταυτόχρονα χρησιμοποιώντας δύο ξεχωριστά C# scripts τα οποία θα έχουν διαφορετική χρηστικότητα και εφαρμογή το καθένα.

Για να " δούμε " λοιπόν τις συσκευές αυτές στο Unity3D θα πρέπει να επέμβουμε στο script **VRPNManager.cs**. Κάνουμε διπλό κλικ στο component VRPNManager (script) στο πεδίο που φαίνεται στην εικόνα που ακολουθεί και ανοίγουμε το αρχείο με ένα IDE Editor όπως το Visual Studio ή το MonoDevelop.



Στον κώδικα εισάγουμε τρείς νέες μεταβλητές απαριθμητού τύπου τις οποίες δηλώνουμε ως δημόσιες (public) με ονομασίες **public enum Tracker_Types, public enum Button_Types, public enum Analog_Types** όπου μέσα στις μεταβλητές αυτές εισάγουμε ονομασίες συσκευών που υποστηρίζονται από τον VRPNServer όπως διακρίνεται στις ακόλουθες γραμμές.

- public enum Tracker_Types

  {

  vrpn_3DConnexion_Navigator,

  vrpn_5dt,

  vrpn_5dt16,

  vrpn_Tracker_Flock,

  vrpn_Tracker_Flock_Parallel

  };

- public enum Button_Types

  {

  vrpn_3DConnexion_Navigator,

  vrpn_Button_5DT_Server,

  vrpn_Button_NI_DIO24,

  vrpn_Button_SerialMouse,

  vrpn_WiiMote,

  vrpn_XInputGamepad

  };

- public enum Analog_Types

  {

  vrpn_3DConnexion_Navigator,

  vrpn_Mouse,

  vrpn_Keyboard,

  vrpn_WiiMote,

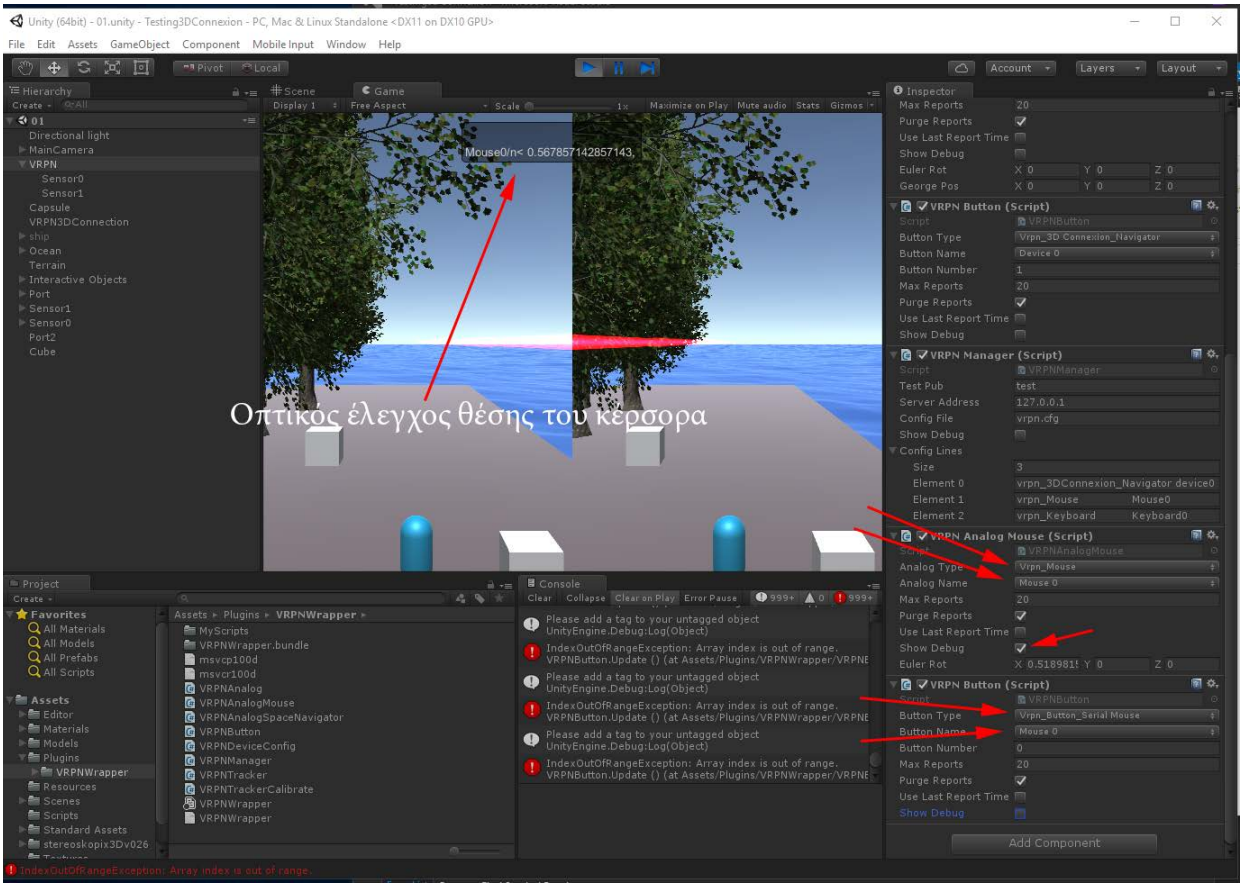
  vrpn_XInputGamepad

  };

Με αυτό τον τρόπο μας δίνεται η δυνατότητα να επιλέξουμε τις συσκευές μας μέσα από τα script components τα οποία έχουμε ενσωματώσει στο άδειο αντικείμενο VRPN ώστε να μπορέσουν να επικοινωνήσουν με τον VRPNServer και το Unity3D, μια διαδικασία την οποία θα δούμε στο επόμενο κεφάλαιο για κάθε συσκευή ξεχωριστά.

5.2.Διαχείριση του ποντικού

Για την απλή περίπτωση του ποντικιού μας επιλέγουμε το component **VRPNAnalogMouse.cs** όπου στο πεδίο **Analog Type** δηλώνουμε **Vrpn_Mouse** ενώ στο πεδίο **Analog Name** το **Mouse0**. Επίσης στο component **VRPNButton.cs** στο πεδίο **Button Type** επιλέγουμε το **Vrpn_Button_SerialMouse** ενώ στο πεδίο **Button Name** ξανά το **Mouse0**. Τέλος για οπτικό έλεγχο του ποντικιού μας, ότι δέχεται και λαμβάνει δεδομένα, μαρκάρουμε στο **VRPNAnalogMouse.cs** το **Show Debug** οπότε κατά την εκτέλεση του προγράμματος (Play) εμφανίζεται ένα ορθογώνιο πλαίσιο στο Game window του Unity3D το οποίο μας πληροφορεί υπό την μορφή κειμένου και αριθμών για την θέση του κέρσορα στην οθόνη του υπολογιστή μας.



5.3.Διαχείριση του 3DConnexion Space Navigator

Η συσκευή 3DConnexion Space Navigator δεν είναι τίποτα άλλο από ένα ποντίκι το οποίο χρησιμοποιείται σε 3D Cad προγράμματα μηχανολογικού σχεδιασμού όπως το Solidworks και το Autodesk Inventor αλλά και σε άλλες εφαρμογές Modeling Sculpting όπως το 3DSMax και το Maya τις οποίες χρησιμοποιεί η βιομηχανία των ηλεκτρονικών παιχνιδιών και του animation.

3D Connexion Space Navigator

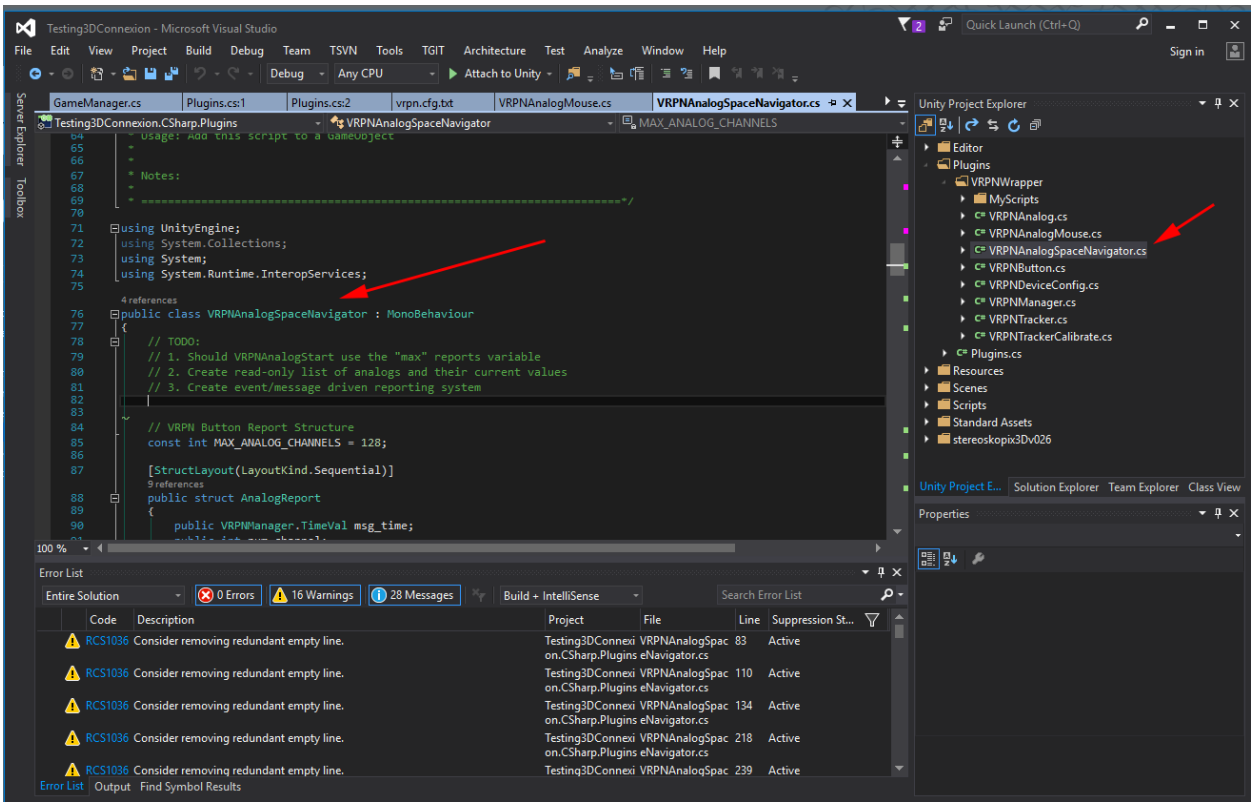
Η ιδιαιτερότητα της συσκευής αυτής, σε σχέση με τα άλλα ποντίκια, έγκειται στο γεγονός ότι δουλεύει στον τρισδιάστατο χώρο για την διαχείριση των μοντέλων μεταβάλλοντας την θέση και τον προσανατολισμό τους. Είναι δηλαδή μια συσκευή με έξι βαθμούς ελευθερίας (6DOF, 6 Degree Of Freedom) οπότε και αναμένουμε να μας δίνει στην έξοδό της τρία δεδομένα για την θέση και τρία δεδομένα για τον προσανατολισμό ενός τρισορθογώνιου συστήματος αξόνων.

Η σύνδεση της συσκευής με το Unity3D και τον VRPNServer γίνεται χρησιμοποιώντας αντικείμενα (Game Objects) και scripts γραμμένα σε γλώσσα Java και C#, όπου η τελευταία είναι και η προτεινόμενη. Συγκεκριμένα στο κενό αντικείμενο **VRPN** του Unity3D κάνουμε drag & drop δύο C# Scripts εφόσον πρώτα έχουμε αλλάξει την ονομασία του ενός. Το πρώτο Script είναι το **VRPNAnalog.cs** το οποίο μετονομάζουμε σε **VRPNAnalogSpaceNavigator.cs** για να μπορούμε να το ξεχωρίζουμε εύκολα όταν γίνει ενσωμάτωση στο άδειο αντικείμενο **VRPN** και το άλλο είναι το **VRPNMouse.cs**. Μεγάλη προσοχή πρέπει να δοθεί στο όνομα των αρχείων C# και στην ονομασία της κλάσης τους όπου επιβάλλεται να είναι ακριβώς ίδια και να μην περιέχονται αριθμοί, σύμβολα και κενά (space) διότι δεν θα δουλέψει ο κώδικας μας, έτσι λοιπόν από το αρχικό όνομα της κλάσης:

- **public class VRPNAnalog : MonoBehaviour**

γίνεται μετονομασία σε

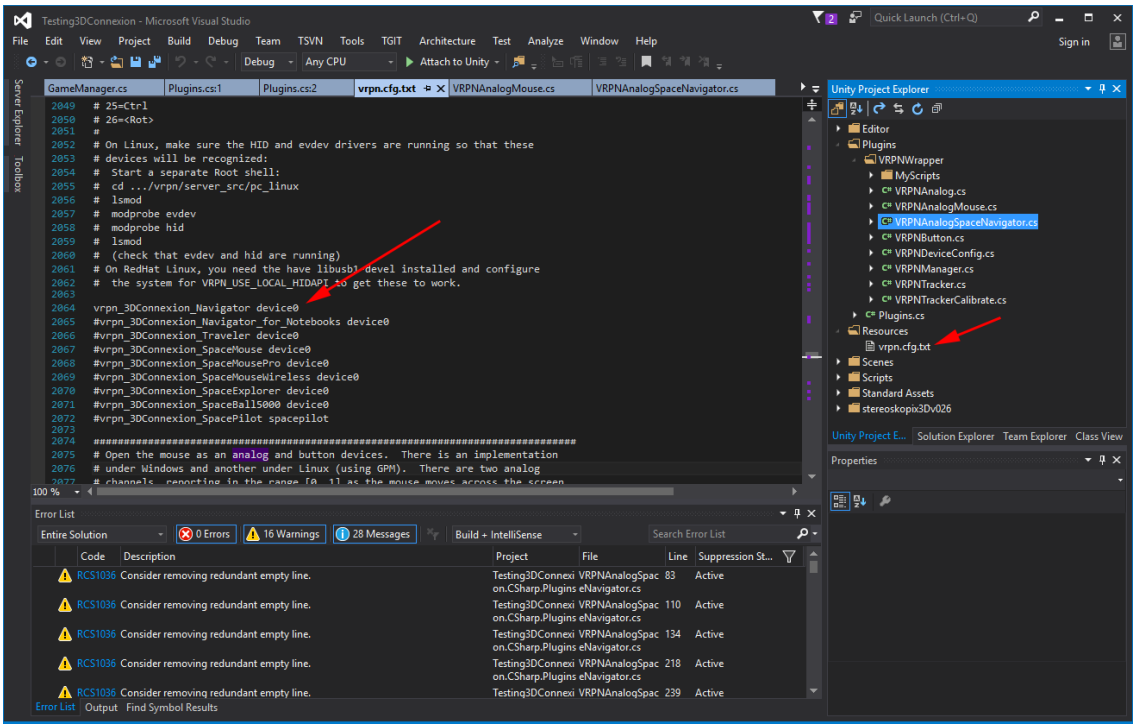
- **public class VRPNAnalogSpaceNavigator : MonoBehaviour**

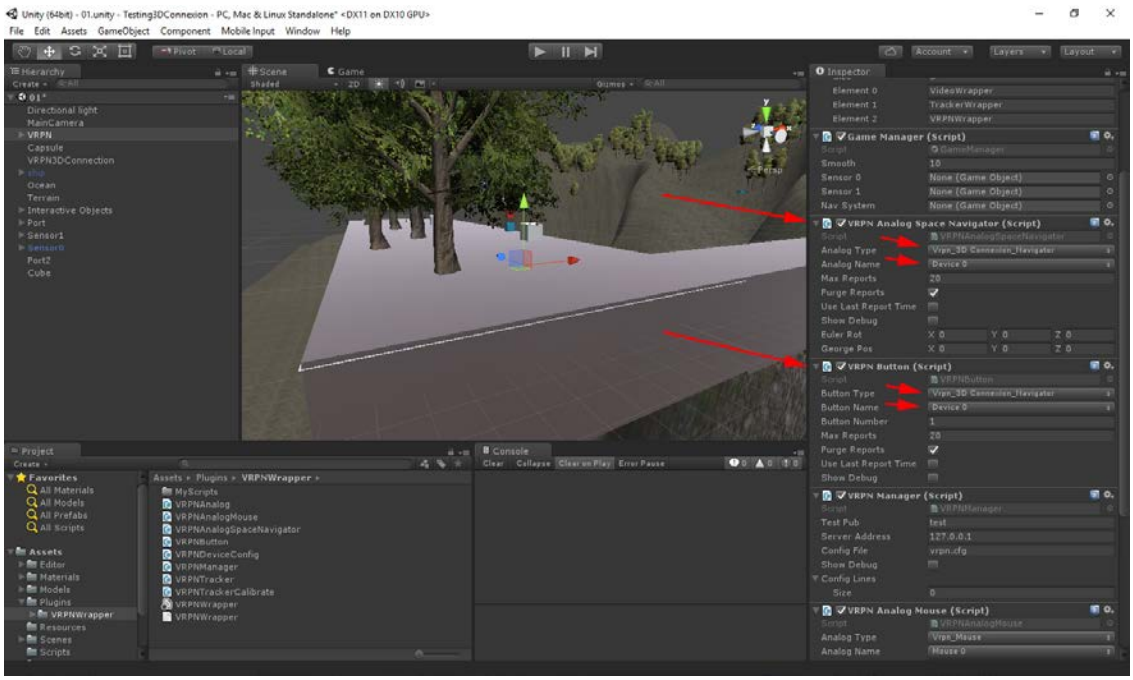


Δεν πρέπει να παραλείψουμε στο component **VRPNAnalogSpaceNavigator,** από το αναδυόμενο μενού του στο πεδίο **Analog Type** να επιλέξουμε το **Vrpn_3DConnexion_Navigator** και στο πεδίο **Analog Name** το **Device0**.

Το δεύτερο script όπως αναφέραμε είναι το **VRPNButton.cs** το οποίο κάνουμε δεύτερη φορά drag & drop στο άδειο αντικείμενο VRPN χωρίς να κάνουμε κάποια μετονομασία αν και θα μπορούσαμε, η διαφορά είναι ότι στα πεδία **Button Type** επιλέγουμε από το αναδυόμενο μενού το **Vrpn_3DConnexion_Navigator** και στο πεδίο **Button Name** το **Device0**.

Οι ονομασίες για τις συσκευές αυτές βρίσκονται στο αρχείο **vrpn.cfg** το οποίο καλό είναι να συμβουλευόμαστε προτού εργαστούμε με μια συσκευή. Για την συγκεκριμένη περίπτωση αν ανοίξουμε το αρχείο **vrpn.cfg** και μεταβούμε στην γραμμή **2064** θα δούμε να αναφέρει **vrpn_3DConnexion_Navigator device0**. Σημαντικό είναι και το αρχείο **VRPNDeviceConfig.cs** το οποίο βρίσκεται στην διαδρομή που έχουμε το Project και συγκεκριμένα στον φάκελο Assets\Plugins\VRPNWrapper**,** το οποίο μας δίνει μια λίστα με τις ονομασίες των συσκευών και χρησιμεύει στον κώδικα που θα γράψουμε.


Testing3DConnexion - Microsoft Visual Studio
File Edit View Project Build Debug Team TSVN Tools TGIT Architecture Test Analyze Window Help
Sign in
Debug
Any CPU
Attach to Unity
GameManager.cs
Plugins.cs:1
Plugins.cs:2
vrpn.cfg.txt
VRPNAnalogMouse.cs
VRPNAnalogSpaceNavigator.cs
2049 # 25=Ctrl
2050 # 26=<Rot>
2051 #
2052 # On Linux, make sure the HID and evdev drivers are running so that these
2053 # devices will be recognized:
2054 # Start a separate Root shell:
2055 # cd .../vrpn/server_src/pc_linux
2056 # lsmod
2057 # modprobe evdev
2058 # modprobe hid
2059 # lsmod
2060 # (check that evdev and hid are running)
2061 # On RedHat Linux, you need the have libusb1-devel installed and configure
2062 # the system for VRPN_USE_LOCAL_HIDAPI to get these to work.
2063
2064 vrpn_3DConnexion_Navigator device0
2065 #vrpn_3DConnexion_Navigator_for_Notebooks device0
2066 #vrpn_3DConnexion_Traveler device0
2067 #vrpn_3DConnexion_SpaceMouse device0
2068 #vrpn_3DConnexion_SpaceMousePro device0
2069 #vrpn_3DConnexion_SpaceMouseWireless device0
2070 #vrpn_3DConnexion_SpaceExplorer device0
2071 #vrpn_3DConnexion_SpaceBall5000 device0
2072 #vrpn_3DConnexion_SpacePilot spacepilot
2073
2074 ##
2075 # Open the mouse as an analog and button devices. There is an implementation
2076 # under Windows and another under Linux (using GPM). There are two analog
2077 # channels, reporting in the range [0, 1] as the mouse moves across the screen
Unity Project Explorer
Editor
Plugins
VRPNWrapper
MyScripts
VRPNAnalog.cs
VRPNAnalogMouse.cs
VRPNAnalogSpaceNavigator.cs
VRPNButton.cs
VRPNDeviceConfig.cs
VRPNManager.cs
VRPNTracker.cs
VRPNTrackerCalibrate.cs
Plugins.cs
Resources
vrpn.cfg.txt
Scenes
Scripts
Standard Assets
stereoskopix3Dv026
Unity Project E... Solution Explorer Team Explorer Class View
Properties
Error List
Entire Solution
0 Errors
16 Warnings
28 Messages
Build + IntelliSense
Search Error List
Code Description Project File Line Suppression St...
RCS1036 Consider removing redundant empty line. Testing3DConnexion.CSharp.Plugins VRPNAnalogSpaceNavigator.cs 83 Active
RCS1036 Consider removing redundant empty line. Testing3DConnexion.CSharp.Plugins VRPNAnalogSpaceNavigator.cs 110 Active
RCS1036 Consider removing redundant empty line. Testing3DConnexion.CSharp.Plugins VRPNAnalogSpaceNavigator.cs 134 Active
RCS1036 Consider removing redundant empty line. Testing3DConnexion.CSharp.Plugins VRPNAnalogSpaceNavigator.cs 218 Active
RCS1036 Consider removing redundant empty line. Testing3DConnexion.CSharp.Plugins VRPNAnalogSpaceNavigator.cs 239 Active
Error List Output Find Symbol Results


Unity (64bit) - 01.unity - Testing3DConnexion - PC, Mac & Linux Standalone* <DX11 on DX10 GPU>
File Edit Assets GameObject Component Mobile Input Window Help
Hierarchy
Directional light
MainCamera
VRPN
Capsule
VRPN3DConnection
Ocean
Terrain
Interactive Objects
Port
Sensor1
Port2
Cube
Scene
Game
Inspector
Game Manager (Script)
VRPN Analog Space Navigator (Script)
VRPN Button (Script)
VRPN Manager (Script)
VRPN Analog Mouse (Script)
Project
Console
Favorites
Assets
Plugins
VRPNWrapper
MyScripts
VRPNAnalog
VRPNAnalogMouse
VRPNAnalogSpaceNavigator
VRPNButton
VRPNDeviceConfig
VRPNManager
VRPNTracker
VRPNTrackerCalibrate
VRPNWrapper

Όπως αναφέραμε παραπάνω η συσκευή 3DConnexion Space Navigator έχει έξι βαθμούς ελευθερίας οπότε καταλαβαίνουμε πως θα έχει και έξι κανάλια επικοινωνίας με τον υπολογιστή από τα οποία δέχεται και λαμβάνει δεδομένα. Τα κανάλια (channels) από 0 έως 2 δηλαδή τα :

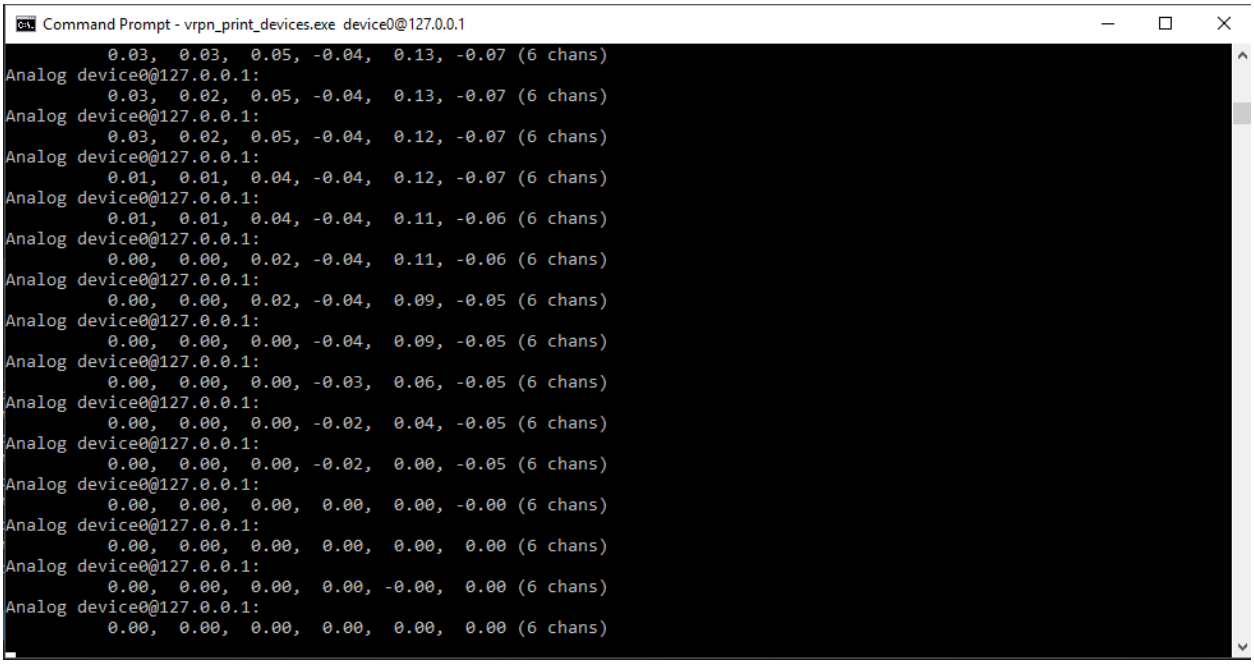
- channel[0]
- channel[1]
- channel[2]

χρησιμοποιούνται για να μεταβάλλουμε την θέση των αντικειμένων (Game Object) και αντιστοιχούν στην μεταφορά των αξόνων X, Y, Z, ενώ τα κανάλια από 3 έως 5 :

- channel[3]
- channel[4]
- channel[5]

αντιστοιχούν στην περιστροφή **περί** των αξόνων αυτών X, Y, Z.

Η εικόνα που ακολουθεί είναι η έξοδος της συσκευής 3DConnexion Space Navigator μέσω του προγράμματος **cmd.exe** όταν τρέξουμε την εντολή " **vrpn_print_devices.exe device0@127.0.0.1** ". Παρατηρούμε ότι τα τρία πρώτα αριθμητικά δεδομένα, από αριστερά προς τα δεξιά όπως βλέπουμε την εικόνα, αντιστοιχούν στην μετακίνηση των αξόνων αναφοράς ενώ τα επόμενα τρία αριθμητικά που ακολουθούν στις περιστροφές περί των αξόνων αυτών.



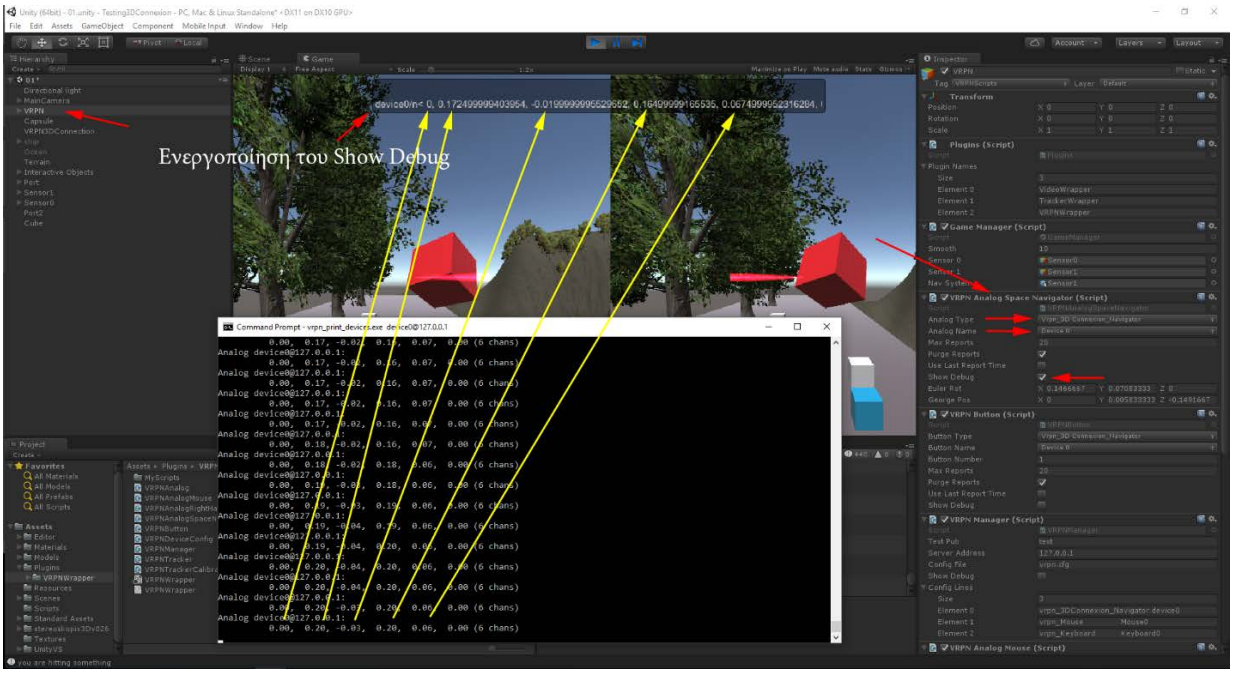
Τώρα που έχουμε φτιάξει το πρόγραμμα (script) **VRPNAnalogSpaceNavigator.cs** και είμαστε σίγουροι ότι η συσκευή μας επικοινωνεί με τον **VRPN Server** θα πρέπει να την συνδέσουμε με το Unity3D και να την χρησιμοποιήσουμε σε κάποιο αντικείμενο (Game Object).

Στο Unity3D λοιπόν έχουμε κάνει drag & drop το script **VRPNAnalogSpaceNavigator.cs** στο άδειο αντικείμενο **VRPN** επιλέγοντας στα πεδία **Analog Type** και **Analog Name** τα ονόματα **Vrpn_3DConnexion_Navigator** και **Device0** αντίστοιχα. Για να διαπιστώσουμε ότι η συσκευή αυτή λαμβάνει και στέλνει δεδομένα μέσα από το script **VRPNAnalogSpaceNavigator.cs** επιλέγουμε το **Purge Reports** και **Show Debug**, έτσι όταν πατήσουμε Play για να ξεκινήσει η εφαρμογή εμφανίζεται πλαίσιο με κείμενο στο οποίο ενημερώνονται τα δεδομένα σύμφωνα με την έξοδο της συσκευής.

Αν στο πλαίσιο εξόδου των δεδομένων δεν υπάρχει αρκετός χώρος για να εμφανιστούν όλες οι τιμές, μπορούμε να επέμβουμε στον κώδικα και να αλλάξουμε τις διαστάσεις και την θέση που κατέχει το πλαίσιο αυτό στο παράθυρο Play του Unity3D, συγκεκριμένα μπορούμε να μεταβάλλουμε τους αριθμούς (debug_xoffset + 10, 20, 600, 45) οι οποίοι διακρίνονται στις γραμμές του κώδικα που ακολουθεί.

```
void OnGUI()
    {
        if (ShowDebug)
        {
            GUI.skin.box.alignment = TextAnchor.LowerLeft;
            GUI.Box(new Rect(debug_xoffset + 10, 20, 600, 45), debug_text);
        }
    }
```

Το αποτέλεσμα μετά από κάποιες δοκιμασίες αλλαγής των αριθμητικών δεδομένων (debug_xoffset + 10, 20, 600, 45) φαίνεται στην παρακάτω εικόνα. Τα βέλη με κίτρινο χρώμα δείχνουν την αντιστοιχία των αποτελεσμάτων που λαμβάνουμε από το πρόγραμμα cmd.exe και το Show Debug στο Unity3D. Υπάρχει μια μικρή απόκλιση μεταξύ των δύο εφαρμογών και αυτό οφείλεται στην στρογγυλοποίηση των δεκαδικών ψηφίων που μας δίνει το cmd.exe, ενώ το Show Debug του Unity3D μας δίνει ακριβέστερη πληροφορία.



Επόμενο βήμα είναι να επέμβουμε στον κώδικα του **VRPNAnalogSpaceNavigator.cs** κάτι που θα αναλύσουμε στο κεφάλαιο 6.

5.4.Διαχείριση του Flock Of Birds

5.4.1.Γνωριμία με το σύστημα Flock Of Birds

Σε αυτό το κεφάλαιο θα αναλύσουμε την συσκευή Flock Of Birds της εταιρίας www.ascension-tech.com, τις δυνατότητές της και την συνδεσμολογία με τον VRPN Server και το Unity3D. Το Flock Of Birds είναι ένα ηλεκτρονικό μετρητικό σύστημα έξι βαθμών ελευθερίας (6DOF, Six Degree Of Freedom) και σκοπός του είναι να εντοπίζει την θέση και τον προσανατολισμό αισθητήρων οι οποίοι είναι συνδεδεμένοι σε ένα σύστημα πομπό-δέκτη. Κάθε αισθητήρας (δέκτης) είναι ικανός να μας δίνει 20 μέχρι 144 μετρήσεις το δευτερόλεπτο της θέσης και του προσανατολισμού του όταν βρίσκεται σε ένα πεδίο ± 4 πόδια (± 1.2192 μέτρα) μέγιστη απόσταση από τον πομπό, όμως δίνεται η δυνατότητα να αυξηθεί το εύρος του πεδίου αυτού στα ± 8 πόδια (± 2.4384 μέτρα) όταν χρησιμοποιούμε έναν προαιρετικό πομπό αύξησης του ηλεκτρομαγνητικού πεδίου ERT (Extended Range Transmitter).

ERT (Extended Range Transmitter)

Το σύστημα FOB (Flock Of Birds) εντοπίζει την θέση και τον προσανατολισμό των αισθητήρων παράγοντας ένα συνεχές (DC) παλμικό μαγνητικό πεδίο, με την βοήθεια του ERT, μέσα στο οποίο είναι εντοπίσιμοι οι αισθητήρες. Από τις μετρήσεις των χαρακτηριστικών μεγεθών του πεδίου εξάγουμε τα απαραίτητα δεδομένα που αφορούν την θέση και τον προσανατολισμό.

Ένα σύστημα FOB (Flock Of Birds) λοιπόν αποτελείται από μία ή περισσότερες ηλεκτρονικές συσκευές οι οποίες διαχωρίζονται σε :

- Σύστημα αισθητήρων με την ονομασία Flock Of Birds ή απλά Birds.
- Επεκτάσιμο σύστημα ελέγχου συσκευών, Extended Range Controller (ERC).
- Πομπό συνεχούς ηλεκτρομαγνητικού πεδίου, Extended Range Transmitter (ERT).

Κάθε συσκευή-αισθητήρας (Bird) καθώς και η συσκευή ERC έχουν δύο ανεξάρτητες θύρες επικοινωνίας. Η πρώτη χρησιμοποιείται για την επικοινωνία των συσκευών με τον ηλεκτρονικό υπολογιστή και φέρει την ονομασία RS-232C η οποία έχει την δυνατότητα λειτουργίας ως full duplex RS-232C ή ως half duplex RS422/485 και η δεύτερη για επικοινωνία μεταξύ των ίδιων των συσκευών Bird και ERC και έχει την ονομασία Fast Bird Bus (FBB).

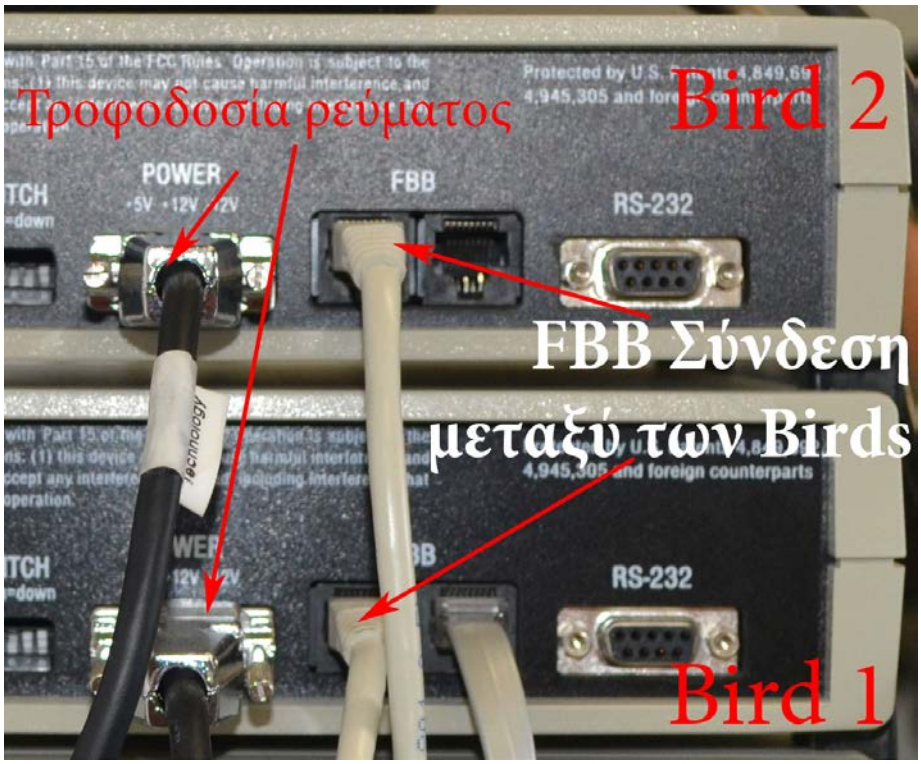
5.4.2.Συνδεσμολογία του Flock Of Birds

Οι ηλεκτρονικές συσκευές του FOB (Flock Of Birds) μπορούν να τοποθετηθούν οπουδήποτε, πρέπει όμως να αποφύγουμε την επαφή τους με άλλες ηλεκτρονικές συσκευές οι οποίες δεν έχουν μαγνητική θωράκιση και να μην επικαλύπτουμε το σύστημα εξαερισμού τους. Η κατασκευή τους επιτρέπει την τοποθέτηση της μίας συσκευής πάνω στην άλλη και αυτός είναι και ο προτεινόμενος τρόπος εφόσον εξασφαλίζει εξοικονόμηση χώρου και διευκολύνει στην συνδεσμολογία τους. Εξαίρεση αποτελεί ο πομπός ERT ο οποίος πρέπει να έχει μια σχετικά μεγάλη απόσταση με τις άλλες συσκευές, περίπου στο 1.5 μέτρο, και κατά προτίμηση τοποθετείται στο πάτωμα λόγω του μεγάλου βάρους του.

Ο πομπός ERT (Extended Range Transmitter) δέχεται ρεύμα μεγάλης τάσης με την βοήθεια του συνοδευτικού καλωδίου μέσω της μονάδας ERC (Extended Range Controller). Η σύνδεση γίνεται στην θέση XMTR 1 ενώ στην θέση XMTR 2 του ERC συνδέουμε προστατευτική γέφυρα για αποφυγή ηλεκτροπληξίας.

Από την παραπάνω συσκευή ERC το καλώδιο της εξόδου RS-232 συνδέεται στην σειριακή θύρα του υπολογιστή, για να μπορούμε να στείλουμε και να δεχτούμε δεδομένα, ενώ το καλώδιο της εξόδου FBB στην πρώτη συσκευή Bird.

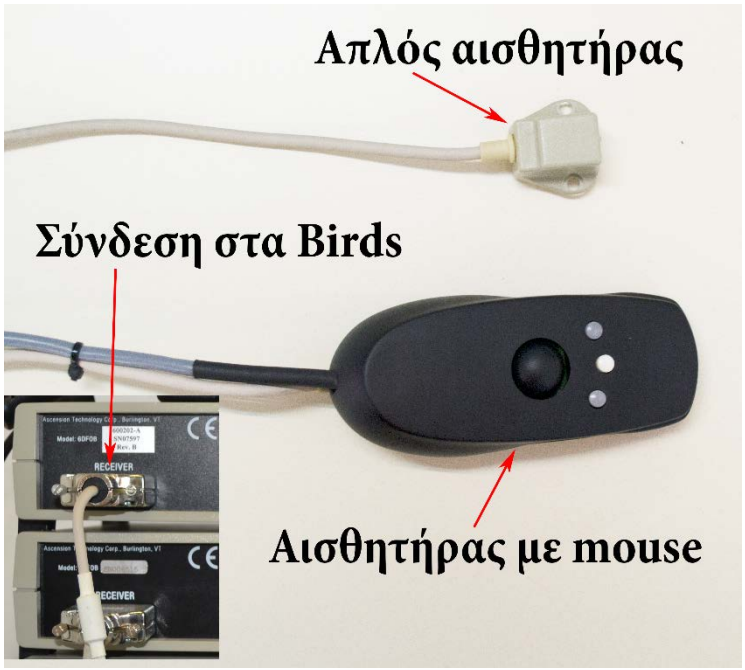
Στην πρώτη συσκευή Bird λοιπόν, στην οποία συνδέσαμε το ERC, συνδέουμε την επόμενη συσκευή Bird πάλι με το πρωτόκολλο επικοινωνίας FBB. Με την ίδια διαδικασία συνδέουμε μεταξύ τους όσες συσκευές Bird διαθέτουμε ώστε τελικά να έχουμε μια συστοιχία από αυτές.



Δεν πρέπει να παραλείψουμε να συνδέσουμε τα τροφοδοτικά ρεύματος των συσκευών Bird ενώ η συσκευή ERC λαμβάνει ρεύμα απευθείας από την πρίζα.

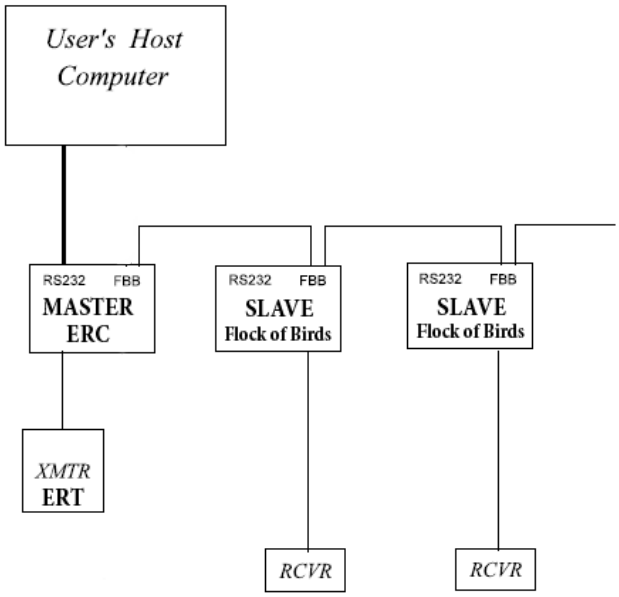
τροφοδοτικά ρεύματος των Birds

Τέλος στις συσκευές Birds συνδέουμε τους τερματικούς αισθητήρες στις θύρες με την ονομασία Receivers (RCVR). Ο ένας αισθητήρας που διαθέτει το εργαστήριο VR_LAB είναι απλός ενώ ο άλλος αισθητήρας έχει ενσωματωμένο ποντίκι το οποίο συνδέεται εκτός από την θύρα Receiver και με τον υπολογιστή σε νέα σειριακή θύρα, ξεχωριστή της σειριακής θύρας που συνδέσαμε τον ERC, και διαθέτει τρία κουμπιά και ημισφαιρικό joystick για την μετακίνηση του κέρσορα.



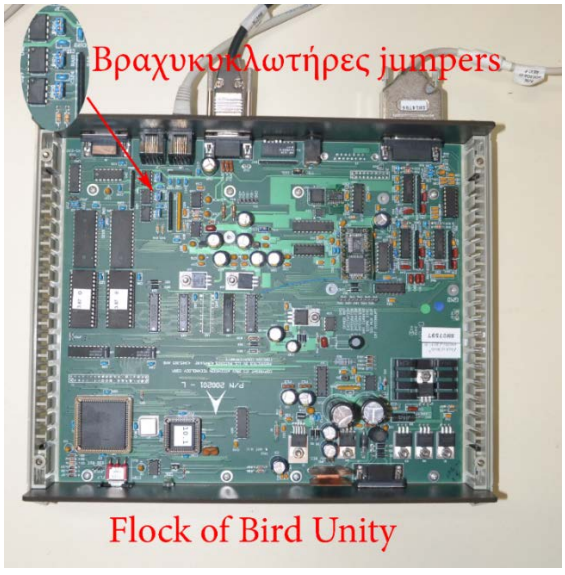
Τύποι Αισθητήρων

Στο παρακάτω σχήμα βλέπουμε την συνδεσμολογία που έχουμε ακολουθήσει στο εργαστήριο της σχολής μας. Έχουμε ρυθμίσει τον ERC να είναι η κύρια συσκευή ελέγχου (Master) και ακολουθούν οι συσκευές Birds ως δευτερεύουσες (Slave) συνδεδεμένες με το FBB ενώ η επικοινωνία του ERC με τον υπολογιστή γίνεται μέσω της σύνδεσης RS232 όπως προαναφέραμε.



5.4.3. Ρυθμίσεις των Jumpers και Dipswitch

Αν ανοίξουμε τις συσκευές FOB θα παρατηρήσουμε στην ηλεκτρονική πλακέτα αριθμημένους βραχυκυκλωτήρες (jumpers) με τους οποίους μας δίνεται η δυνατότητα να ρυθμίσουμε τον τρόπο διασύνδεσης των συσκευών μεταξύ τους αλλά και με τον υπολογιστή. Η εργοστασιακή ρύθμιση διασύνδεσης είναι με το πρωτόκολλο επικοινωνίας RS-232C, η ρύθμιση δηλαδή που χρησιμοποιούμε στην εργαστηριακή διάταξη για την εκπόνηση της διπλωματικής εργασίας.



Παρακάτω παραθέτουμε πίνακα με τις ρυθμίσεις των βραχυκυκλωτήρων, το σύμβολο * αφορά τις εργοστασιακές ρυθμίσεις.

Jumper	Function
3	1 – 2 connected, FBB/RS485 host control 1 – 2 not connected, RS232 host control *
9	1 – 2 connected, TTL Sync Signal enable 2 – 3 connected, CRT Sync Signal enabled *
10	1 – 2 connected, FBB CTS terminated 1 – 2 not connected, FBB CTS not terminated *
12	1 – 2 connected, FBB Receive Data enabled 2 – 3 connected, RS232 Receive Data enabled *
14	1 – 2 connected, FBB BIRD DATA terminated 1 – 2 not connected, FBB BIRD DATA not terminated *
15	1 – 2 connected, TTL single Sync input 2 – 3 connected, factory-supplied CRT Sync input cable*
16	1 – 2 connected, FBB HOST DATA terminated 1 – 2 not connected, FBB HOST DATA not terminated *
17	1 – 2 connected, FBB reset is enabled 1 – 2 not connected, FBB reset is not enabled *
18	1 – 2 connected, RS232 Request to Send signal will reset Bird 1 – 2 not connected, RS232 Request to Send signal will not reset Bird *

Για την απόδοση της διεύθυνσης των ηλεκτρονικών μονάδων του FOB χρησιμοποιούμε διακόπτες με την ονομασία Dipswitches. Οι συσκευές μπορούν να δουλέψουν με τρείς μεθόδους

- Normal Addressing mode (Dipswitches 1, 2, 3 on) χρησιμοποιείται όταν έχουμε μέχρι 14 συσκευές Birds,
- Expanded Addressing mode (Dipswitches 1, 2 on) χρησιμοποιείται όταν έχουμε παραπάνω από 14 συσκευές Birds και
- Super-Expanded Addressing mode χρησιμοποιείται όταν έχουμε παραπάνω από 30 συσκευές Birds.

Στο εργαστήριο της σχολής μας έχουμε δύο συσκευές Birds, μια συσκευή Extended Range Controller (ERC) καθώς και ένα πομπό ηλεκτρομαγνητικού πεδίου (ERT) οπότε η λειτουργία του FOB γίνεται σε Normal Addressing mode. Για να το πετύχουμε αυτό απενεργοποιούμε την συσκευή ERC κλείνοντάς την από τον διακόπτη ο οποίος βρίσκεται στο πίσω μέρος της ενώ τις συσκευές Birds τις βάζουμε σε STBY (stand by) mode και ενεργοποιούμε τους διακόπτες (dipswitches) 1, 2, 3 κατεβάζοντάς τους σε όλες τις συσκευές, δηλώνοντας έτσι το Baud Rate να είναι 115200. Πρέπει να σημειωθεί ότι το Baud Rate δηλώνει πόσες φορές ένα ψηφιακό σήμα αλλάζει από μηδέν σε ένα ή από ένα σε μηδέν μέσα σε ένα δευτερόλεπτο.

Στην συνέχεια με τους διακόπτες 4, 5, 6, 7 δηλώνουμε μια μοναδική διεύθυνση για κάθε συσκευή. Στην συσκευή ERC ενεργοποιούμε τον διακόπτη 7 οπότε αυτή η συσκευή θα πάρει την διεύθυνση :

- 1 = 0001 = off, off, off, on και θα γίνει Master

Στην συσκευή Bird που βρίσκεται ακριβώς από πάνω της ενεργοποιούμε τον διακόπτη 6 οπότε θα πάρει την διεύθυνση :

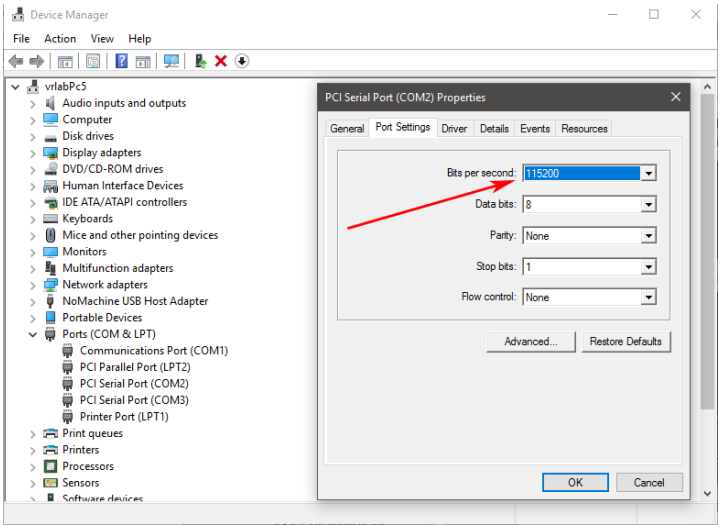
- 2 = 0010 = off, off, on, off ως Slave_1

Στην τελευταία συσκευή ενεργοποιούμε τους διακόπτες 6 και 7 οπότε θα πάρει την διεύθυνση :

- 3 = 0011 = off, off, on, on ως Slave_2.

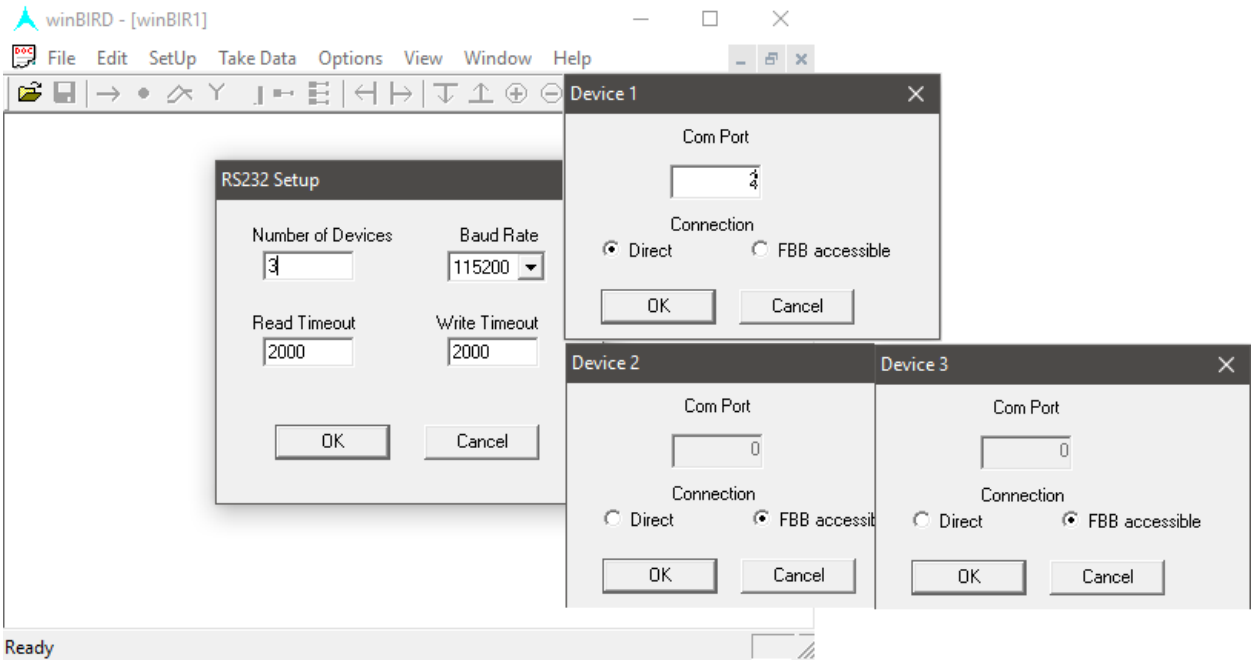
5.4.4.Έλεγχος σωστής λειτουργίας του FOB

Εφόσον έχουν γίνει σωστά οι ρυθμίσεις και η συνδεσμολογία του FOB ήρθε η ώρα να το βάλουμε σε λειτουργία. Στην συσκευή ERC τοποθετούμε τον διακόπτη, στο πίσω μέρος της, στην θέση **I (on)** ενώ στις συσκευές Bird από **STBY (StandBy)** βάζουμε τους διακόπτες, στο μπροστινό μέρος, στην θέση **FLY**. Οι μπροστινές λυχνίες θα αναβοσβήσουν τρείς φορές ένδειξη που δηλώνει ότι οι συσκευές είναι έτοιμες και επικοινωνούν μεταξύ τους. Το πρώτο πράγμα που πρέπει να δώσουμε προσοχή είναι η συνδεσμολογία του ERC με τον υπολογιστή μέσω της σειριακής θύρας RS232. Στον υπολογιστή της σχολής έχουμε κάνει εγκατάσταση κάρτα PCI η οποία μας δίνει δύο σειριακές εισόδους COM2 και COM3 τις οποίες έχουμε ρυθμίσει να έχουν Baud Rate ίδιο με αυτό του ERC δηλαδή 115200 όπως διακρίνεται στην παρακάτω εικόνα. Το ERC έχει συνδεθεί στην σειριακή θύρα COM2.

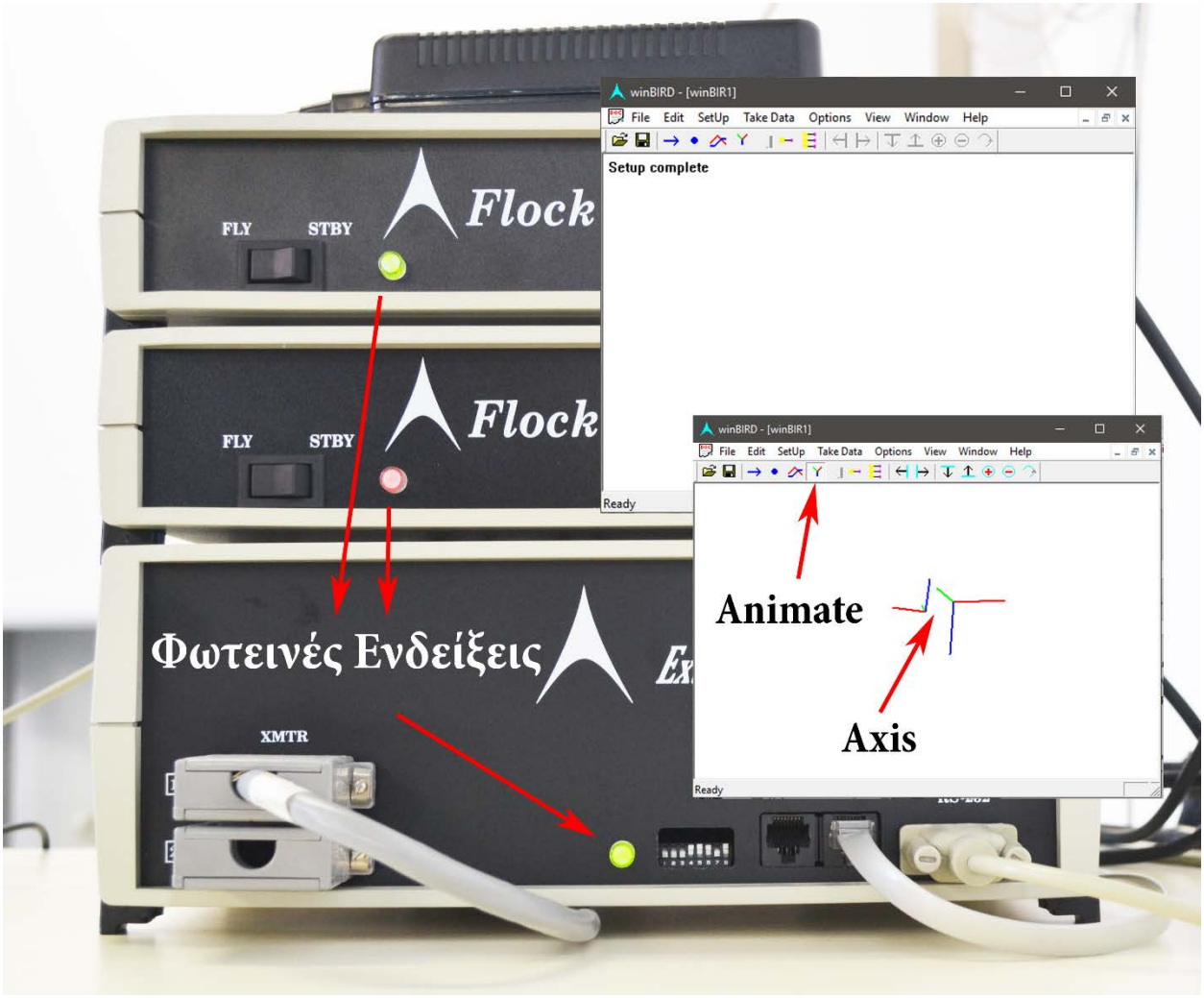


Baud Rate

Η εταιρία www.ascension-tech.com παρέχει λογισμικό με την ονομασία winBIRD.exe το οποίο κάνουμε εγκατάσταση στον υπολογιστή μας. Όταν ανοίξει το πρόγραμμα ανατρέχουμε στο κυρίως μενού και συγκεκριμένα στο SetUp -> RS232 και πληκτρολογούμε στο πεδίο Number of Devices τον αριθμό 3, όσες είναι και οι συσκευές μας, ενώ στο Baud Rate επιλέγουμε την τιμή 115200 και πατάμε ok. Στο επόμενο μενού με την ονομασία Device1 στο πεδίο Com Port βάζω τον αριθμό 2, όπου είναι και ο αριθμός της θύρας του υπολογιστή στην οποία έχουμε συνδέσει την συσκευή ERC η οποία είναι η Device1 εφόσον έχουμε δώσει την πρώτη διεύθυνση 0001. Κάνουμε κλικ στην επιλογή Direct γιατί η σύνδεση με τον υπολογιστή είναι RS232 και πατάμε ok. Στα επόμενα δύο μενού που μου εμφανίζει, δηλαδή στο Device2 και Device3, μαρκάρουμε την επιλογή FBB accessible και πατάμε ok.



Στο παράθυρο του winBIRD βλέπουμε το μήνυμα Setup complete ενώ στις συσκευές Flock of Birds και ERC οι φωτεινές ενδείξεις που βρίσκονται στο μπροστινό μέρος είναι συνεχώς αναμμένες κάτι το οποίο δηλώνει ότι η επικοινωνία μεταξύ τους αλλά και με τον υπολογιστή έχει επιτευχθεί. Για να έχω οπτικό έλεγχο της κίνησης των αισθητήρων στο winBIRD επιλέγω το εικονίδιο Animate το οποίο μου εμφανίζει, αναλόγως του αριθμού των αισθητήρων, γραφήματα τριών αξόνων χρώματος μπλε, πράσινο και κόκκινο.

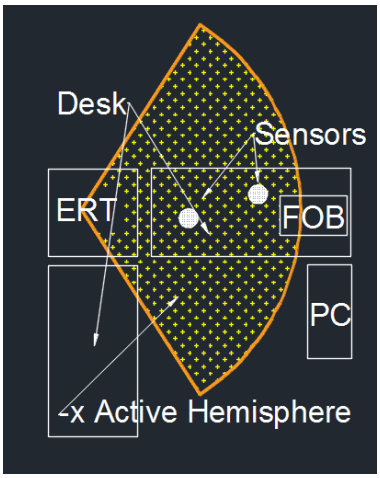


5.4.5.Λειτουργία του FOB μέσω VRPN Server

Ακολουθώντας την παραπάνω συνδεσμολογία και κάνοντας τον έλεγχο του συστήματος FOB είμαστε βέβαιοι για την σωστή λειτουργία του. Το επόμενο βήμα είναι η συνδεσμολογία με τον VRPN Server και το Unity3D μέσω του VRPN Wrapper.

Όσον αφορά τον VRPN Server ανοίγουμε το αρχείο **vrpn.cfg** και μεταβαίνουμε στην γραμμή που αναφέρεται σχετικά με την συσκευή μας Flock of Birds. Αν διαβάσουμε τις οδηγίες που δίνει το αρχείο **vrpn.cfg** βλέπουμε ότι η συσκευή δέχεται εφτά παραμέτρους που πρέπει να εισάγουμε :

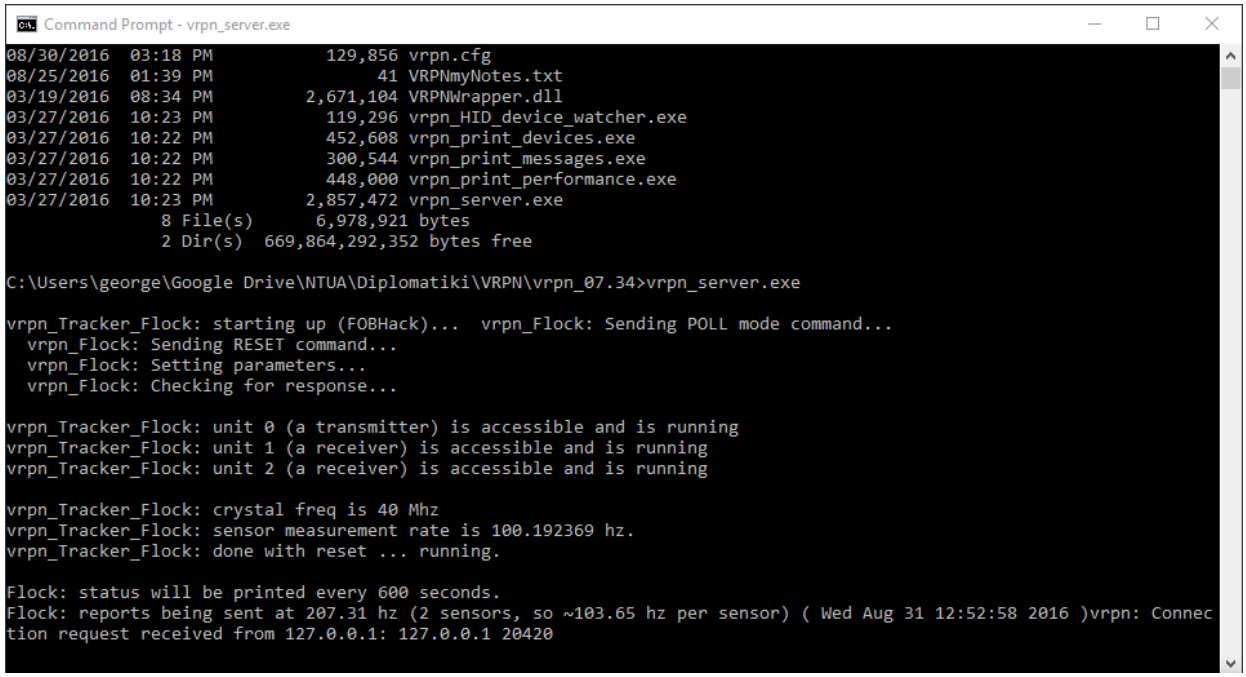
- **char name_of_this_device[]**, είναι η ονομασία της συσκευής για την οποία επιλέγουμε **Tracker0**.
- **int number_of_sensors**, είναι ο αριθμός των αισθητήρων οπότε επιλέγουμε **2**.
- **char name_of_serial_device[]**, είναι το όνομα της θύρας επικοινωνίας δηλαδή **COM2**.
- **int baud_rate_of_serial_device**, είναι το Baud Rate της συσκευής το οποίο είναι **115200**.
- **int invert_quaternion (0 = no, 1 = yes)**, επιλέγουμε να έχουμε αναστραμμένο ημισφαιρικό ηλεκτρομαγνητικό πεδίο ή όχι, εμείς επιλέγουμε **1**
- **char useERT (OPTIONAL; defaults to yes)**, εφόσον χρησιμοποιούμε την συσκευή ERT η επιλογή μας είναι **Y**
- **char[2] active_hemisphere (OPTIONAL; defaults to +z)**, εδώ επιλέγουμε με ποιο ημισφαιρικό ηλεκτρομαγνητικό πεδίο της συσκευής ERT θα δουλέψουμε. Αυτό εξαρτάται από την τοποθέτηση της συσκευής ERT, από την θέση και τον προσανατολισμό της, σε σχέση με την διάταξη και τον χώρο κίνησης των τερματικών αισθητήρων. Η δικιά μας επιλογή είναι **-x**, όμως πρέπει να αναφέρουμε ότι υπάρχει η δυνατότητα των επιλογών "+x", "-x", "+y", "-y", "+z", "-z". Για να γίνει κατανοητή η έννοια της χρήσης του ημισφαιρίου της συσκευής ERT παραθέτουμε σε κάτοψη την διάταξη που έχουμε στο εργαστήριο της σχολής όπου το πεδίο ενεργείας του ημισφαιρίου απεικονίζεται ως τμήμα κυκλικού δίσκου με κίτρινο χρώμα .



Το τελικό κείμενο στο αρχείο **vrpn.cfg** το οποίο δίνει τις πληροφορίες στον VRPN Server για τον τρόπο που θα διαχειριστεί την συσκευή FOB είναι το παρακάτω :

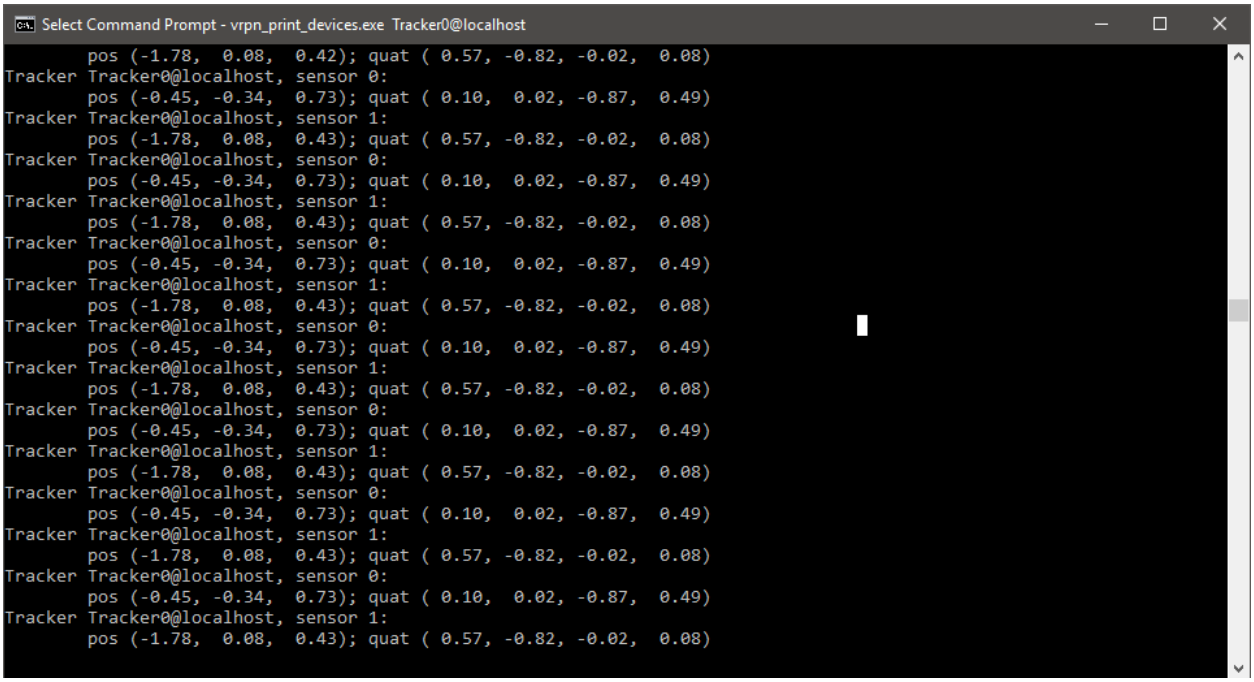
- **vrpn_Tracker_Flock Tracker0 2 COM2 115200 1 Y -x**

Εφόσον έχουμε τερματίσει την λειτουργία του FOB κλείνοντας το πρόγραμμα winBIRD ήρθε η στιγμή να ξεκινήσουμε και πάλι την συσκευή μέσω του **cmd.exe** με την εντολή " **vrpn_server.exe** ", δηλαδή χρησιμοποιώντας τον VRPN Server. Μετά από λίγα δευτερόλεπτα, εφόσον έχουμε δώσει την εντολή, η συσκευή θα ξεκινήσει και στο τερματικό θα εμφανιστούν μηνύματα που αφορούν την σωστή λειτουργία της αλλά και λοιπές πληροφορίες για τις περιφερειακές μονάδες, όπως ο αριθμός αισθητήρων, που είναι συνδεδεμένοι πάνω της κάτι το οποίο διακρίνεται στη παρακάτω εικόνα.



```
08/30/2016  03:18 PM           129,856 vrpn.cfg
08/25/2016  01:39 PM                41 VRPNmyNotes.txt
03/19/2016  08:34 PM         2,671,104 VRPNWrapper.dll
03/27/2016  10:23 PM           119,296 vrpn_HID_device_watcher.exe
03/27/2016  10:22 PM           452,608 vrpn_print_devices.exe
03/27/2016  10:22 PM           300,544 vrpn_print_messages.exe
03/27/2016  10:22 PM           448,000 vrpn_print_performance.exe
03/27/2016  10:23 PM         2,857,472 vrpn_server.exe
               8 File(s)      6,978,921 bytes
               2 Dir(s)  669,864,292,352 bytes free

C:\Users\george\Google Drive\NTUA\Diplomatiki\VRPN\vrpn_07.34>vrpn_server.exe

vrpn_Tracker_Flock: starting up (FOBHack)...  vrpn_Flock: Sending POLL mode command...
  vrpn_Flock: Sending RESET command...
  vrpn_Flock: Setting parameters...
  vrpn_Flock: Checking for response...

vrpn_Tracker_Flock: unit 0 (a transmitter) is accessible and is running
vrpn_Tracker_Flock: unit 1 (a receiver) is accessible and is running
vrpn_Tracker_Flock: unit 2 (a receiver) is accessible and is running

vrpn_Tracker_Flock: crystal freq is 40 Mhz
vrpn_Tracker_Flock: sensor measurement rate is 100.192369 hz.
vrpn_Tracker_Flock: done with reset ... running.

Flock: status will be printed every 600 seconds.
Flock: reports being sent at 207.31 hz (2 sensors, so ~103.65 hz per sensor) ( Wed Aug 31 12:52:58 2016 )vrpn: Connec
tion request received from 127.0.0.1: 127.0.0.1 20420
```

Αν επιθυμούμε να πάρουμε τα δεδομένα από τους αισθητήρες της συσκευής FOB πρέπει σε ένα ξεχωριστό cmd.exe να τρέξουμε την εντολή “ **vrpn_print_devices.exe Tracker0@127.0.0.1** ”. Το αποτέλεσμα της παραπάνω εντολής είναι η θέση $\text{pos} = (number, number, number)$ και $\text{quat} = (number, number\ , number, number)$ ο προσανατολισμός των δύο αισθητήρων **sensor0** και **sensor1**.



```
        pos (-1.78,  0.08,  0.42); quat ( 0.57, -0.82, -0.02,  0.08)
Tracker Tracker0@localhost, sensor 0:
        pos (-0.45, -0.34,  0.73); quat ( 0.10,  0.02, -0.87,  0.49)
Tracker Tracker0@localhost, sensor 1:
        pos (-1.78,  0.08,  0.43); quat ( 0.57, -0.82, -0.02,  0.08)
Tracker Tracker0@localhost, sensor 0:
        pos (-0.45, -0.34,  0.73); quat ( 0.10,  0.02, -0.87,  0.49)
Tracker Tracker0@localhost, sensor 1:
        pos (-1.78,  0.08,  0.43); quat ( 0.57, -0.82, -0.02,  0.08)
Tracker Tracker0@localhost, sensor 0:
        pos (-0.45, -0.34,  0.73); quat ( 0.10,  0.02, -0.87,  0.49)
Tracker Tracker0@localhost, sensor 1:
        pos (-1.78,  0.08,  0.43); quat ( 0.57, -0.82, -0.02,  0.08)
Tracker Tracker0@localhost, sensor 0:
        pos (-0.45, -0.34,  0.73); quat ( 0.10,  0.02, -0.87,  0.49)
Tracker Tracker0@localhost, sensor 1:
        pos (-1.78,  0.08,  0.43); quat ( 0.57, -0.82, -0.02,  0.08)
Tracker Tracker0@localhost, sensor 0:
        pos (-0.45, -0.34,  0.73); quat ( 0.10,  0.02, -0.87,  0.49)
Tracker Tracker0@localhost, sensor 1:
        pos (-1.78,  0.08,  0.43); quat ( 0.57, -0.82, -0.02,  0.08)
Tracker Tracker0@localhost, sensor 0:
        pos (-0.45, -0.34,  0.73); quat ( 0.10,  0.02, -0.87,  0.49)
Tracker Tracker0@localhost, sensor 1:
        pos (-1.78,  0.08,  0.43); quat ( 0.57, -0.82, -0.02,  0.08)
Tracker Tracker0@localhost, sensor 0:
        pos (-0.45, -0.34,  0.73); quat ( 0.10,  0.02, -0.87,  0.49)
Tracker Tracker0@localhost, sensor 1:
        pos (-1.78,  0.08,  0.43); quat ( 0.57, -0.82, -0.02,  0.08)
```

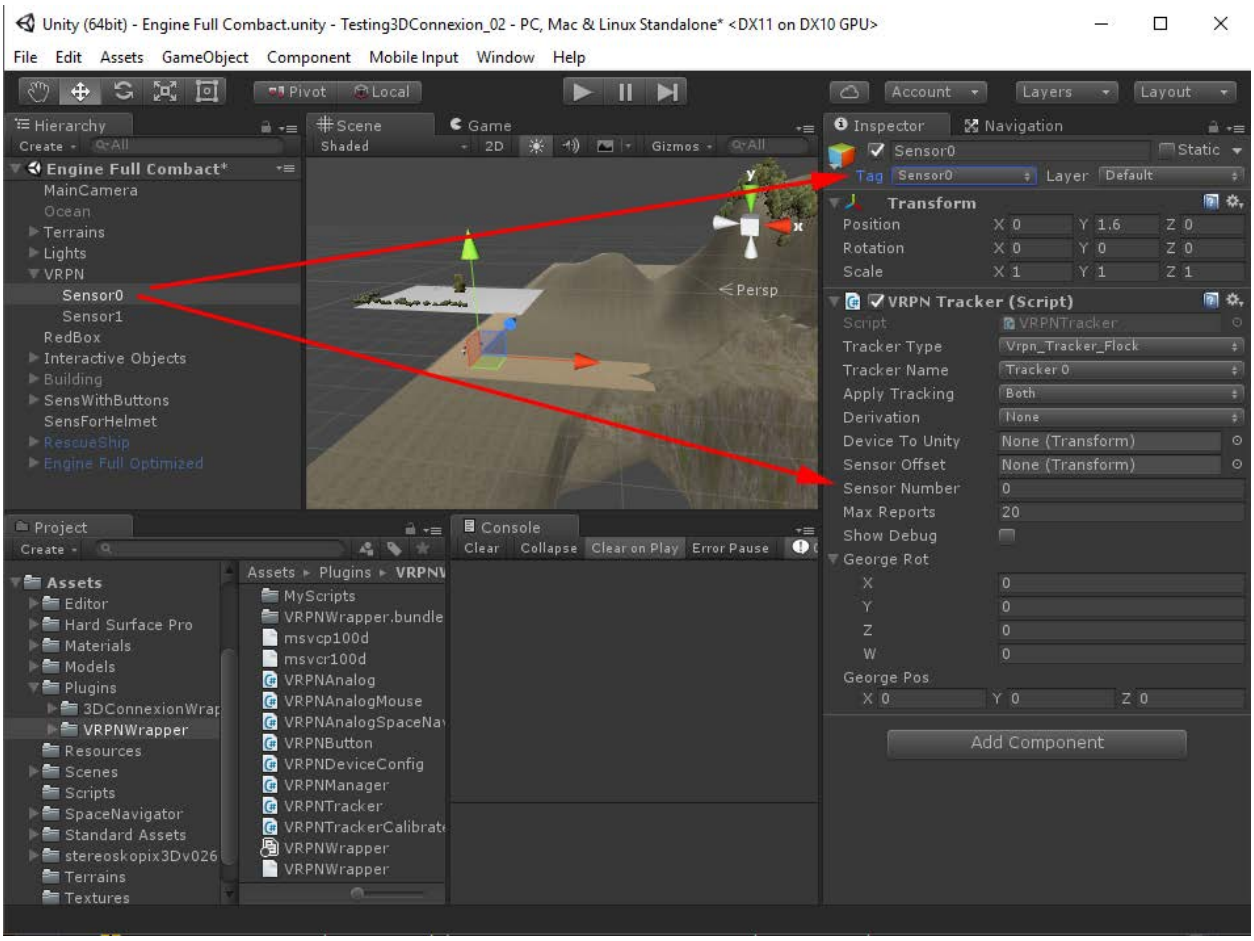
Παρατηρούμε ότι για τις περιστροφές των αισθητήρων η συσκευή κάνει χρήση της μεθόδου του τετραδονίου (Quaternions) η οποία έχει άμεση εφαρμογή στα τρισδιάστατα γραφικά υπολογιστών.

5.4.6.Σύνδεση του FOB με το Unity3D

Το επόμενο βήμα είναι η σύνδεση του συστήματος FOB με το Unity3D με την βοήθεια του VRPN Wrapper. Στο κενό αντικείμενο (empty Game Object) VRPN που έχουμε δημιουργήσει στο Unity3D εισάγουμε άλλα δύο κενά αντικείμενα με τις ονομασίες Sensor0 και Sensor1, τα οποία αφορούν τους αισθητήρες που διαθέτει το σύστημα FOB, κάνοντάς τα drag & drop πάνω του.

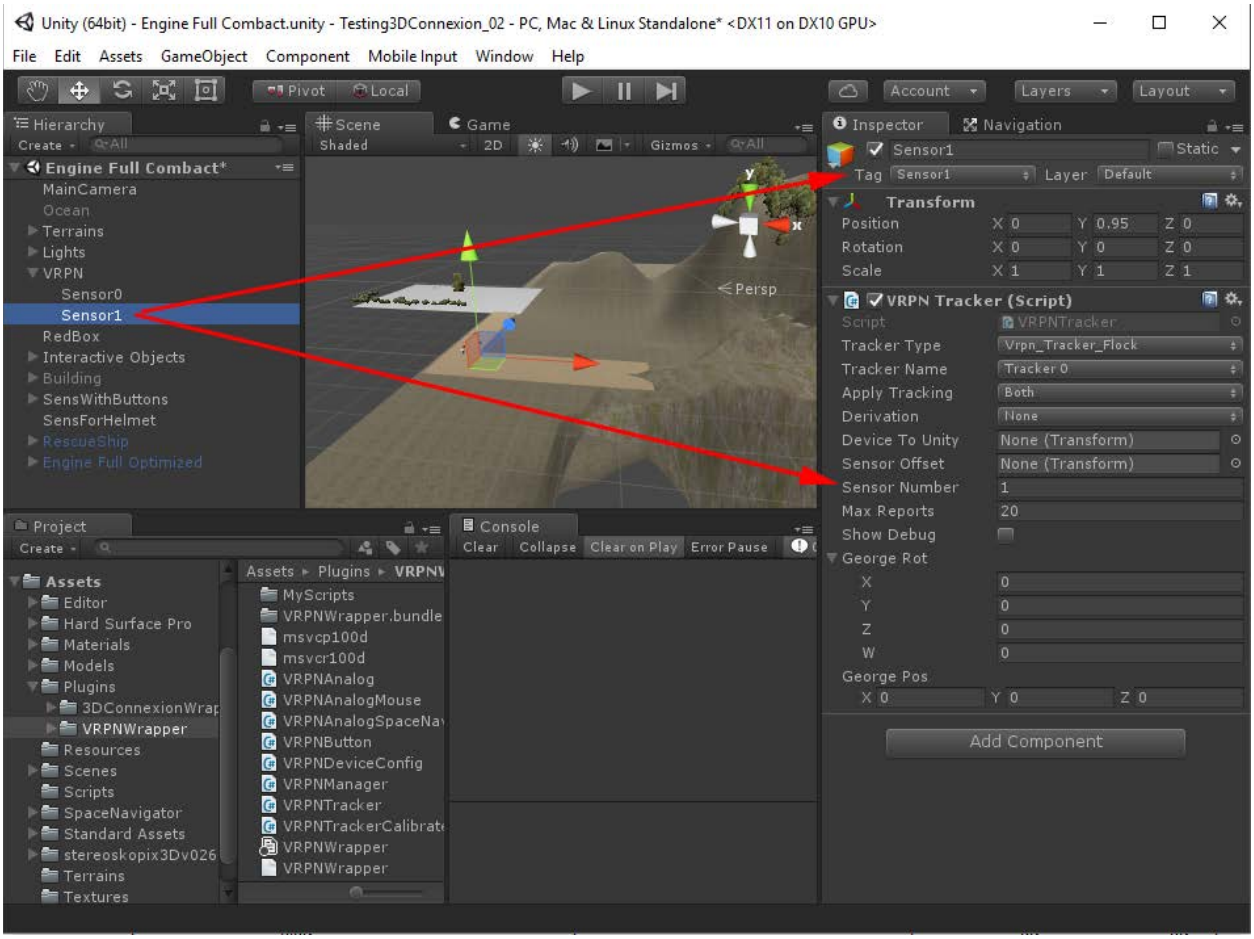
Στην συνέχεια σε κάθε ένα από τα αντικείμενα αυτά κάνουμε drag & drop το ίδιο script με την ονομασία **VRPNTracker.cs** το οποίο βρίσκεται στον φάκελο Assets\Plugins\VRPNWrapper του Unity3D. Επιλέγουμε το Sensor0, οπότε στο Inspector window εμφανίζεται το συστατικό VRPNTracker, και διαλέγουμε από το αναδυόμενο μενού να έχει τις παρακάτω παραμέτρους :

- Tracker Type : Vrpn_Tracker_Flock
- Tracker Name : Tracker0
- Apply Tracking : Both
- Derivation : None
- Device To Unity : None
- Sensor Offset : None
- Sensor Number : 0
- MaxmReports : 20



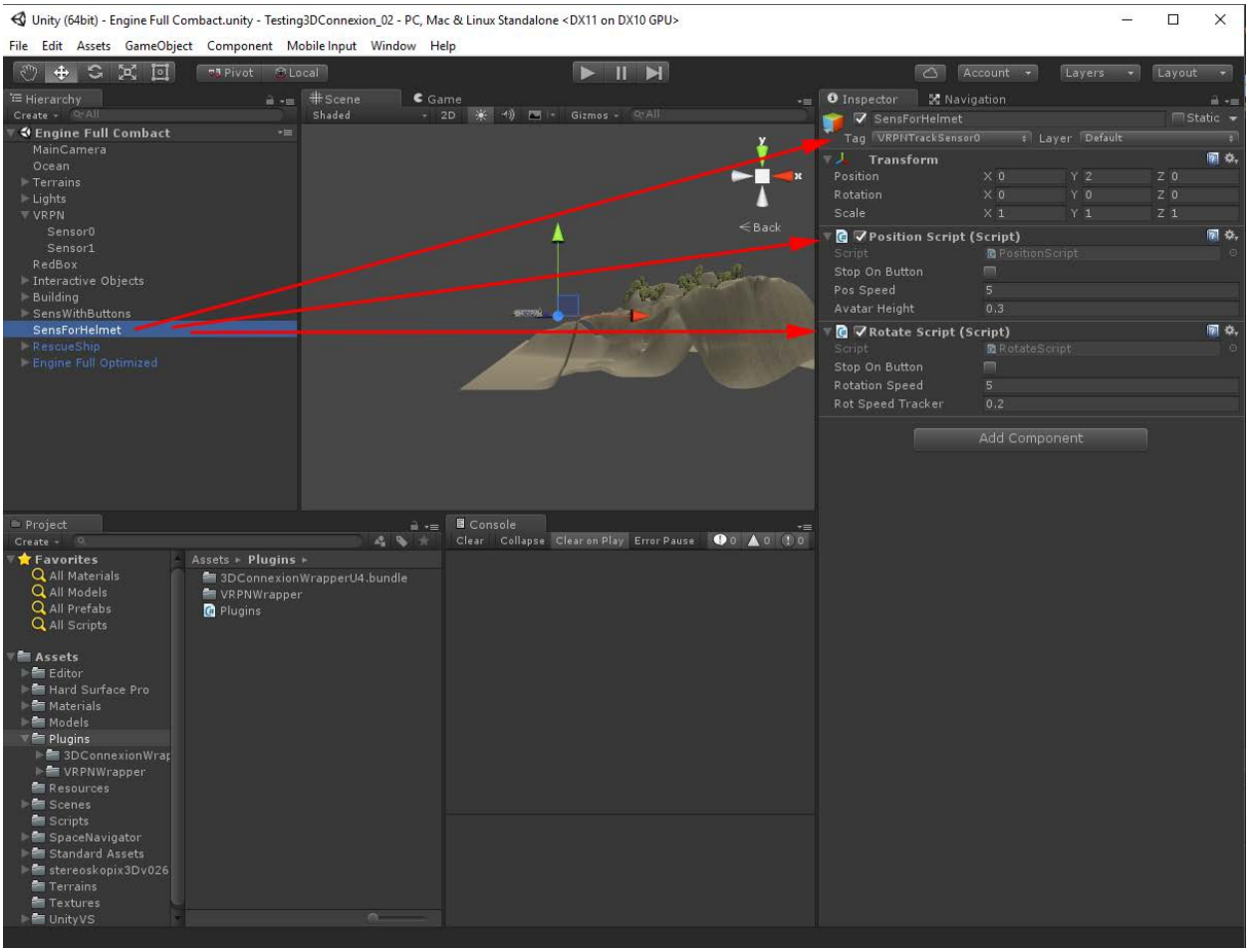
Για το αντικείμενο Sensor1 ισχύουν εξίσου οι παραπάνω επιλογές με την διαφορά ότι στο Sensor Number θα δώσουμε τιμή 1.

- Tracker Type : Vrpn_Tracker_Flock
- Tracker Name : Tracker0
- Apply Tracking : Both
- Derivation : None
- Device To Unity : None
- Sensor Offset : None
- Sensor Number : 1
- MaxmReports : 20



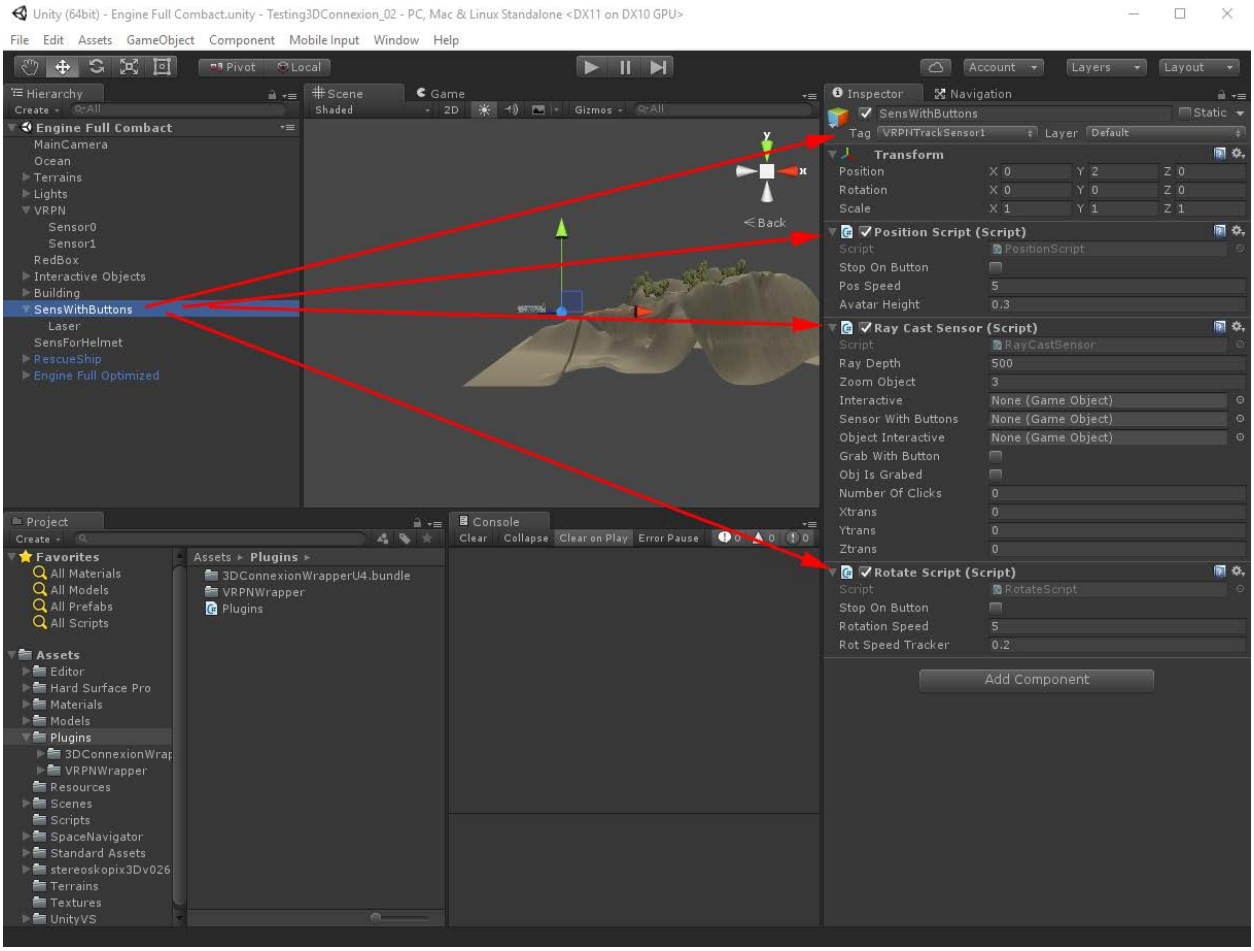
Επίσης κάτι τελευταίο αλλά σημαντικό από το Inspector window για κάθε ένα κενό αντικείμενο-αισθητήρα (Sensor0, Sensor1) δηλώνουμε και την ονομασία του από το αναδυόμενο μενού Tag με την βοήθεια του Add Tag. Έτσι για το αντικείμενο Sensor0 επιλέγουμε την ονομασία Sensor0 και αντίστοιχα για το αντικείμενο Sensor1 την ονομασία Sensor1. Οι ονομασίες αυτές είναι απαραίτητες διότι χρησιμεύουν στον διαχωρισμό των αισθητήρων όσον αφορά το προγραμματιστικό κομμάτι το οποίο θα δούμε αργότερα.

Στην συνέχεια δημιουργούμε άλλα δύο κενά αντικείμενα στο Unity3D με τις ονομασίες **SensWithButtons**, στο οποίο δηλώνουμε Tag -> VRPNTrackSensror0, και **SensForHelmet,** στο οποίο δηλώνουμε Tag -> VRPNTrackSensror1. Έπειτα κάνουμε drag & drop και στα δύο αντικείμενα τα δύο scripts που έχουμε φτιάξει και χρησιμοποιήσει για την μετακίνηση και περιστροφή αντικειμένων με την βοήθεια του 3D Connexion Space Navigator με ονομασίες **PosistionScript.cs** και **RotateScipt.cs** αντίστοιχα. Στο κενό αντικείμενο **SensForHelmet** αντιστοιχεί ο απλός αισθητήρας τον οποίο συνδέουμε στο κράνος εικονικής πραγματικότητας της εταιρίας **www.virtualresearch.com** με την ονομασία **VR1280**.



SensForHelmet

Στο κενό αντικείμενο **SensWithButtons** αντιστοιχεί ο αισθητήρας με το ενσωματωμένο ποντίκι το οποίο θα χρησιμοποιήσουμε για να μετακινούμαστε μέσα στο εικονικό περιβάλλον αλλά και για να αλληλοεπιδρούμε με διάφορα αντικείμενα. Έτσι εκτός των δύο προαναφερθέντων scripts **PosistionScript.cs** και **RotateScipt.cs** δημιουργούμε και ενσωματώνουμε άλλο ένα script με την ονομασία **RayCastSensor.cs** του οποίου ο κώδικας θα μελετηθεί στο κεφάλαιο 6.

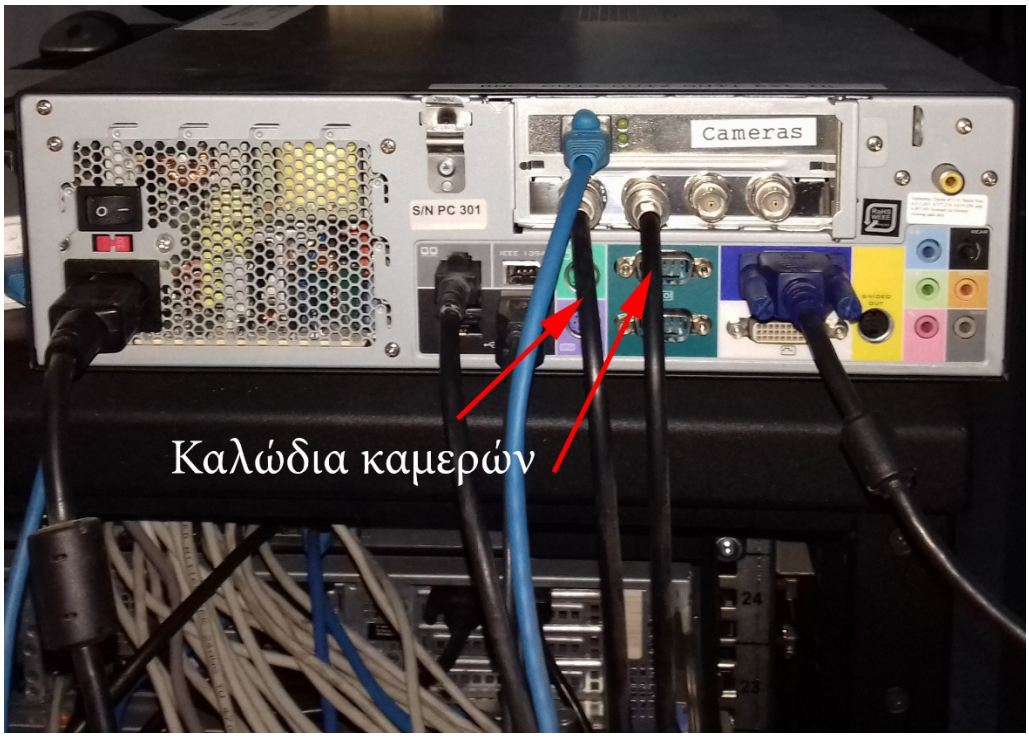


SensWithButtons

Τέλος, δημιουργούμε άλλο ένα κενό αντικείμενο ως " παιδί " (child) του **SensWithButtons** με την ονομασία Laser στο οποίο εισάγουμε δύο νέα Components, ένα νέο script με ονομασία **Laser.cs** και το **LineRenderer** πατώντας το κουμπί Add Component και μεταβαίνοντας στο μενού **Effects**.

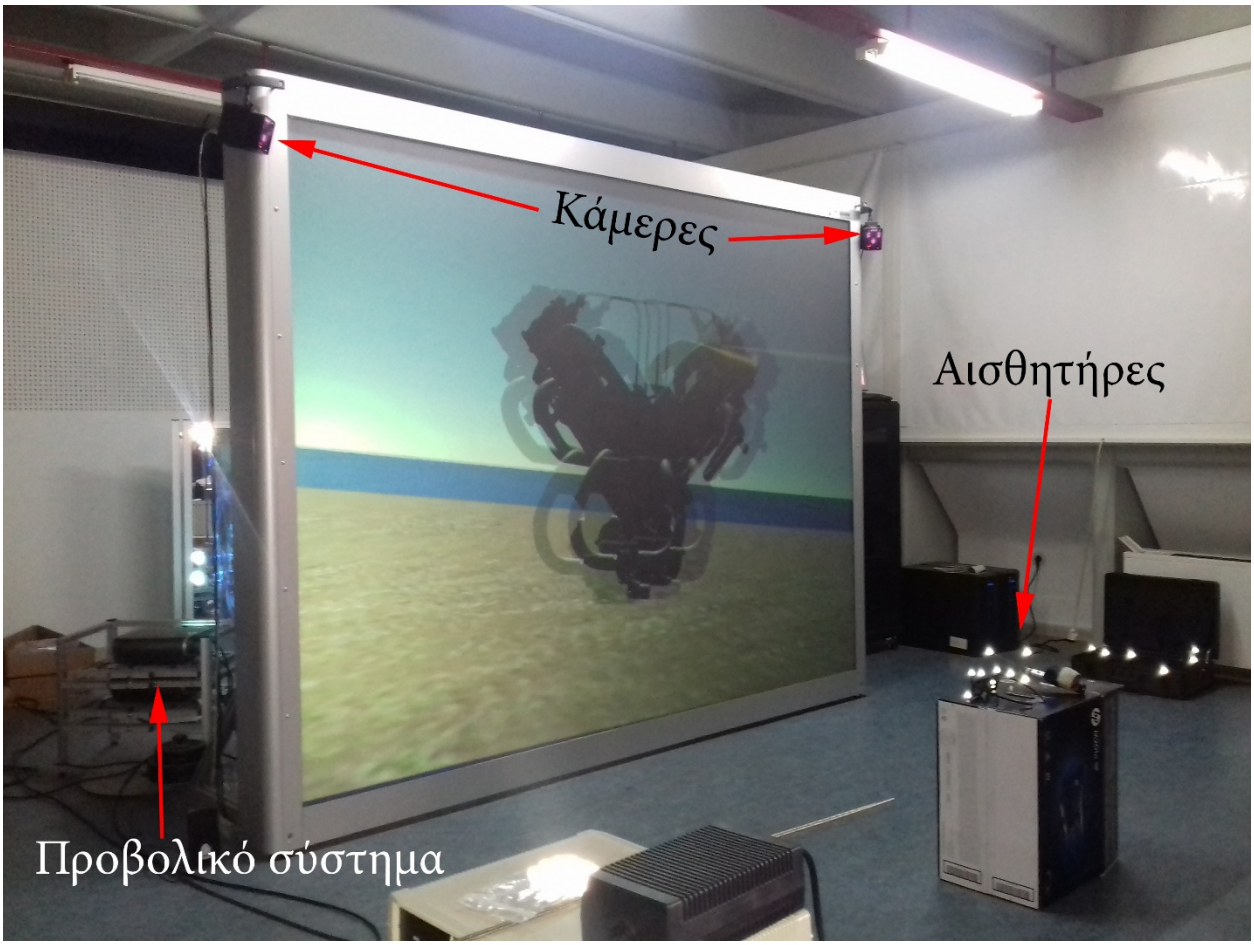
5.5.Διαχείριση του ARTTRACK System

Το εργαστήριο VR-LAB της σχολής Ναυπηγών Μηχανολόγων Μηχανικών είναι εξοπλισμένο και με ένα άλλο σύστημα έξι βαθμών ελευθερίας (6DOF) εντοπισμού της θέσης και προσανατολισμού αισθητήρων κατασκευασμένο από την εταιρία www.ar-tracking.com. Το συγκεκριμένο σύστημα αποτελείται από Ηλεκτρονικό Υπολογιστή εξοπλισμένο με PCI κάρτα με την βοήθεια της οποίας ελέγχουμε τις δύο τοποθετημένες σε πλαίσιο κάμερες. Οι κάμερες τροφοδοτούνται από ηλεκτρικό ρεύμα και είναι συνδεδεμένες με διακομιστή (router) μέσω καλωδίου ethernet ώστε να μπορούμε να λαμβάνουμε τα δεδομένα τα οποία στέλνουν μέσω των αισθητήρων από τον ηλεκτρονικό υπολογιστή όπως διακρίνεται και στις παρακάτω εικόνες.


Cameras
S/N PC 301
Καλώδια καμερών


Αριστερή κάμερα
Πάνελ
Γκρί καλώδιο ethernet
Μαύρο καλώδιο κάμερας
για την PCI κάρτα του PC

Το αλουμινένιο πλαίσιο στηρίζει ημιδιαφανές πάνελ όπου στο πίσω μέρος του βρίσκονται δύο προβολικά συστήματα αναπαραγωγής βίντεο τα οποία στέλνουν την εικόνα σε καθρέπτη και από τον καθρέπτη τελικά η εικόνα καταλήγει στο επίπεδο του πάνελ το οποίο βλέπει ο θεατής.



Τα δύο προβολικά συστήματα είναι συνδεδεμένα με τον κύριο υπολογιστή του εργαστηρίου μέσω σύνδεσης Display Port ενώ έχει χρησιμοποιηθεί η τεχνολογία NVIDIA Mosaic ώστε να αποτελούν ένα ενιαίο monitor κάτι το οποίο μας βοηθάει στην προβολή δύο ξεχωριστών εικόνων για την τελική απόδοση των τρισδιάστατων γραφικών.

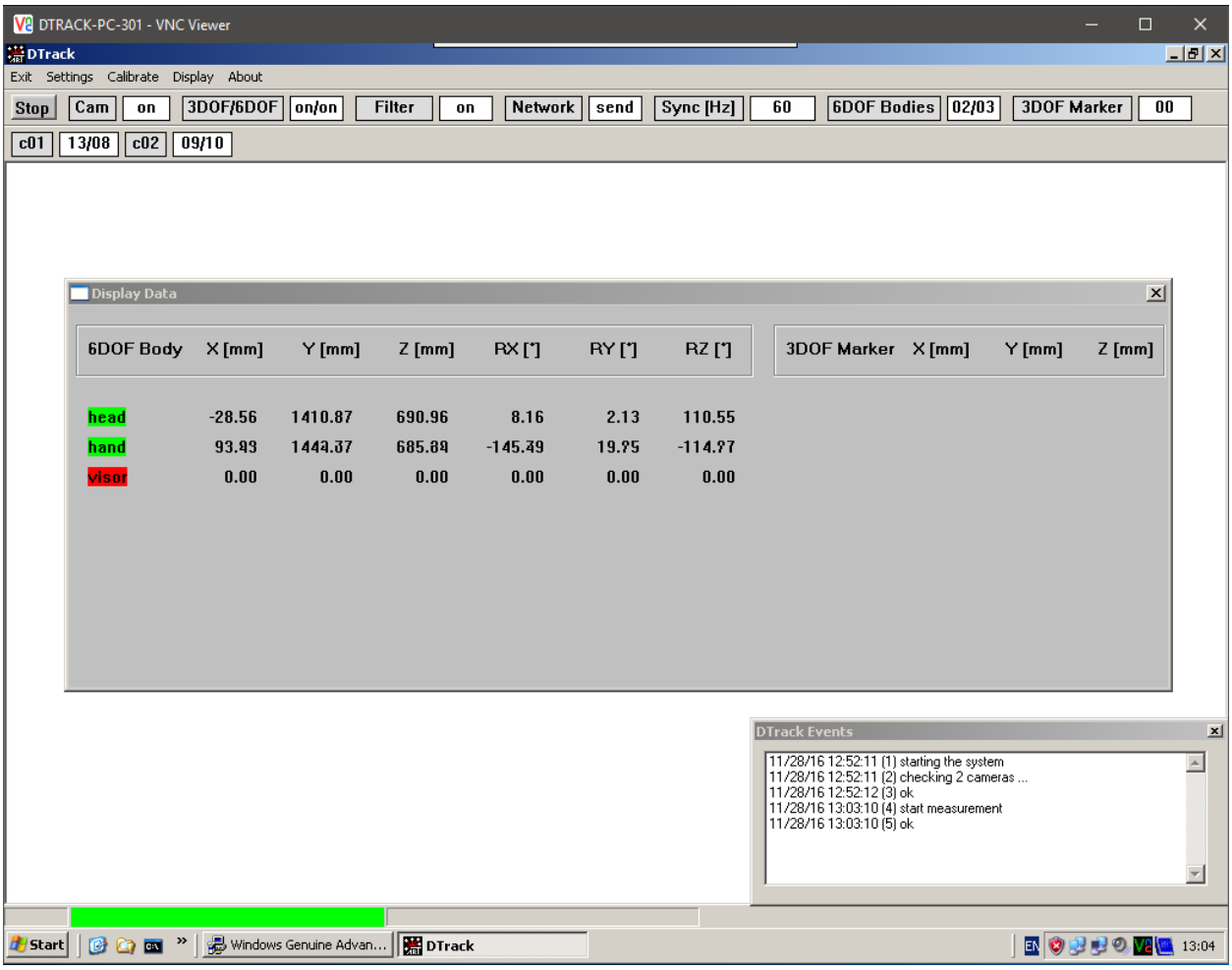
Οι κάμερες εντοπίζουν την θέση των αισθητήρων με υπέρυθρες ακτίνες ενώ οι αισθητήρες φέρουν σφαιρίδια τοποθετημένα σε διάφορες θέσεις και σκοπός τους είναι η αναγνώριση του τύπου, της θέσης και του προσανατολισμού τους από το πρόγραμμα DTrack το οποίο εκτελείται από τον υπολογιστή που είναι συνδεδεμένες. Το εργαστήριο μας διαθέτει γυαλιά-αισθητήρα τα οποία φοράει ο χρήστης καθώς και ποντίκι-αισθητήρα όπως διακρίνονται στις παρακάτω εικόνες.


Σφαιρίδια εντοπισμού αισθητήρα


Σφαιρίδια εντοπισμού αισθητήρα

Για την λειτουργία του εν λόγο συστήματος εκτελούμε το πρόγραμμα DTrack.exe στον υπολογιστή που είναι συνδεδεμένες οι κάμερες. Στην αριστερή πάνω γωνία του παραθύρου του προγράμματος εκτελούμε την εντολή Start και αμέσως λαμβάνουμε δεδομένα για την θέση και τον προσανατολισμό των συσκευών-αισθητήρων όπως διακρίνεται στην παρακάτω εικόνα.

- Head, αφορά τον αισθητήρα των γυαλιών
- Hand, αφορά τον αισθητήρα του ποντικιού

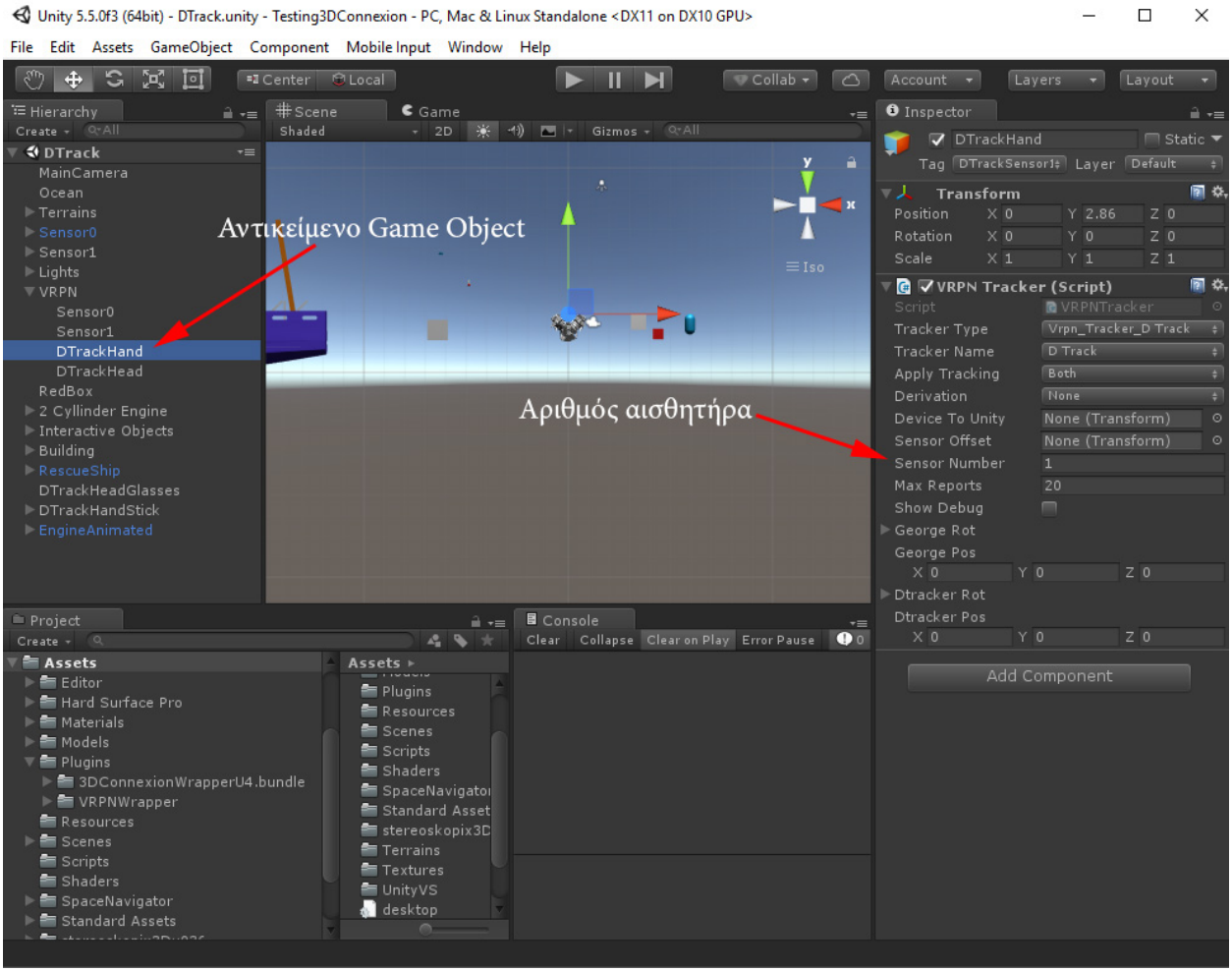


Για να λάβουμε τα δεδομένα μας στο Unity3D διαμέσου του VRPN Server ανατρέχουμε στο αρχείο vrpn.cfg και ξε-σχολιάζουμε την συσκευή με όνομα vrpn_Tracker_DTrack ώστε τελικά να έχουμε ενεργή την γραμμή με τα παρακάτω δεδομένα :

- vrpn_Tracker_DTrack DTrack 5000 0.5 2 2 2 1 0 3 3d -

5.5.1.Σύνδεση του ARTTRACK με το Unity3D

Το επόμενο βήμα είναι να εισάγουμε τέσσερα αντικείμενα (Game Objects) στο Project του Unity3D και να τους δώσουμε τα απαραίτητα συστατικά (Components) ώστε να γίνει ο κατάλληλος διαχωρισμός τους. Για τα αντικείμενα με την ονομασία **DTrackHand** και **DTrackHead** επιλέγουμε να τους δώσουμε τα Tags **DTrackSensor1** και **DTrackSensor2** και να εισάγουμε και στα δύο το συστατικό-script **VRPNTracker.cs**. Στο **DTrackSensor1** και συγκεκριμένα στο συστατικό **VRPNTracker** επιλέγουμε για Tracker Type τον Vrpn_Tracker_Dtrack, για TrackerName το DTrack ενώ στο πεδίο Sensor Number εισάγουμε τον αριθμό 1. Έτσι έχουμε δηλώσει στο Unity ότι αυτό το αντικείμενο να παίρνει δεδομένα από τον αντίστοιχο Tracker που δεν είναι άλλος από το DTrack με αριθμό αισθητήρα το 1. Αντίστοιχα ενεργούμε και για το άλλο αντικείμενο **DTrackSensor2** με την μόνη διαφορά ότι θα εισάγουμε τον αριθμό αισθητήρα 2 στο πεδίο Sensor Number.

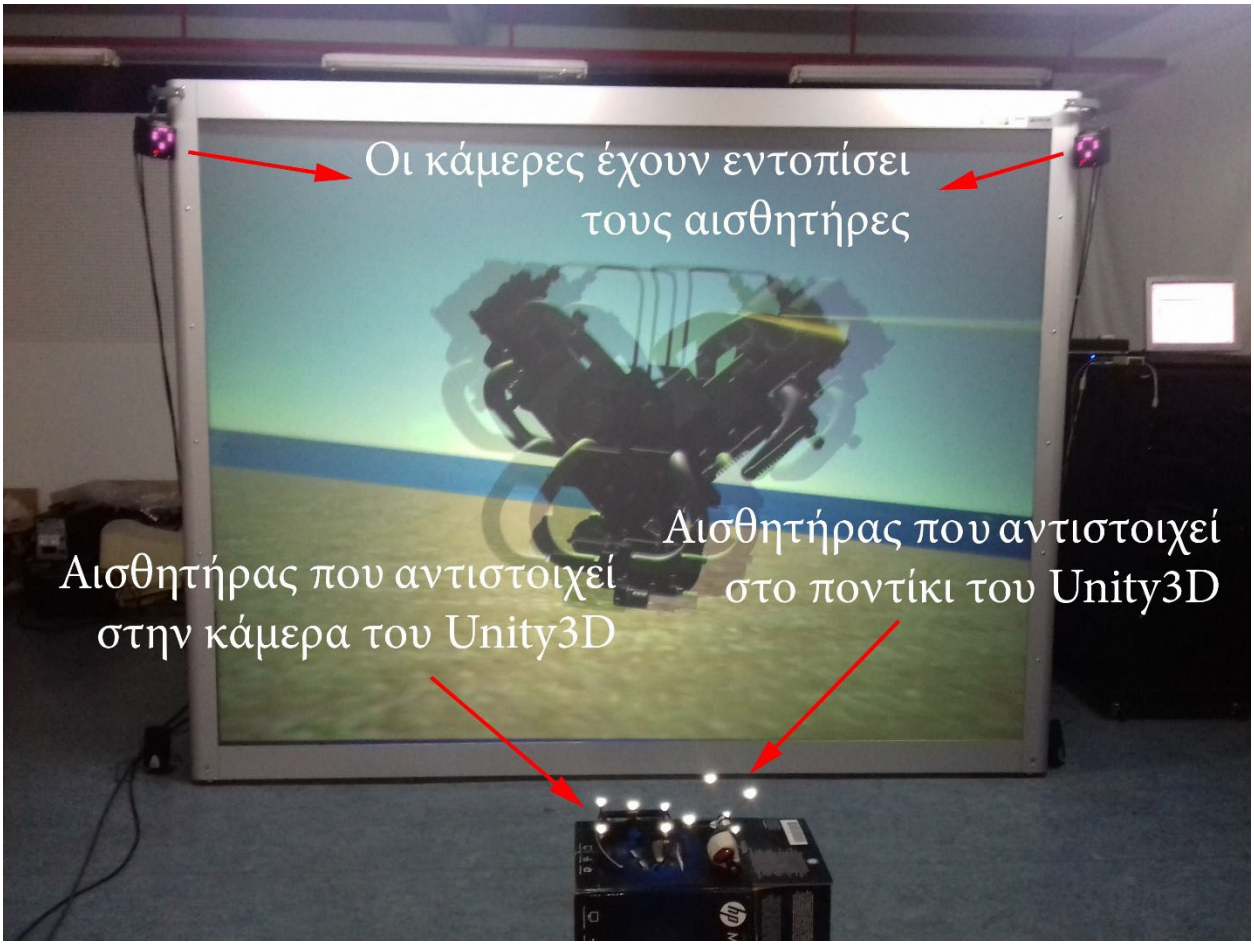


Το επόμενο αντικείμενο μας είναι η κυρία κάμερα **MainCamera** στην οποία αντιστοιχούμε το Tag **VRPNDTrackHead** και προσθέτουμε άλλα δύο συστατικά (components) τα :

- Movement Script, μας βοηθάει στην μετατόπιση και περιστροφή της κάμερας.
- NavMeshAgent, μας βοηθάει στην περιήγηση του χαρακτήρα μέσα στην σκηνή.

Τέλος το αντικείμενο στο οποίο θα αντιστοιχίσουμε τον ποντίκι μας είναι το **DTrackHandStick** στο οποίο δίνουμε την ονομασία Tag **VRPNDTrackHand** και προσθέτουμε τα παρακάτω συστατικά :

- Movement Script, μας βοηθάει στην μετατόπιση και περιστροφή του εικονικού ποντικιού.
- LineRenderer, μας βοηθάει στην απεικόνιση ακτίνας λέιζερ
- RaycastSensor, μας δίνει πληροφορίες για αντικείμενα που συγκρούονται με την ακτίνα λέιζερ και μας βοηθάει να εκτελούμε διάφορες ενέργειες μέσα στον εικονικό κόσμο.
- NavMeshAgent, μας βοηθάει στην περιήγηση του χαρακτήρα μέσα στην σκηνή.



6. Ανάλυση του Κώδικα

6.1. VRPNAnalogSpaceNavigator.cs για το 3D Connexion Space Navigator

Ανοίγουμε το αρχείο **VRPNAnalogSpaceNavigator.cs** με το Visual Studio και προσθέτουμε στην κατηγορία // Public Properties τις παρακάτω γραμμές :

```
public VRPNManager.Analog_Types AnalogType = VRPNManager.Analog_Types.vrpn_3DConnexion_Navigator;
public VRPNDeviceConfig.Device_Names AnalogName = VRPNDeviceConfig.Device_Names.device0;
```

Δηλώνοντας να είναι public το AnalogType και AnalogName έχουμε την δυνατότητα ελέγχου μέσα από το Unity3D ώστε αν θελήσουμε να πραγματοποιήσουμε κάποια μεταβολή αυτή να γίνει εύκολα και γρήγορα χωρίς να πρέπει να επέμβουμε στον κώδικα.

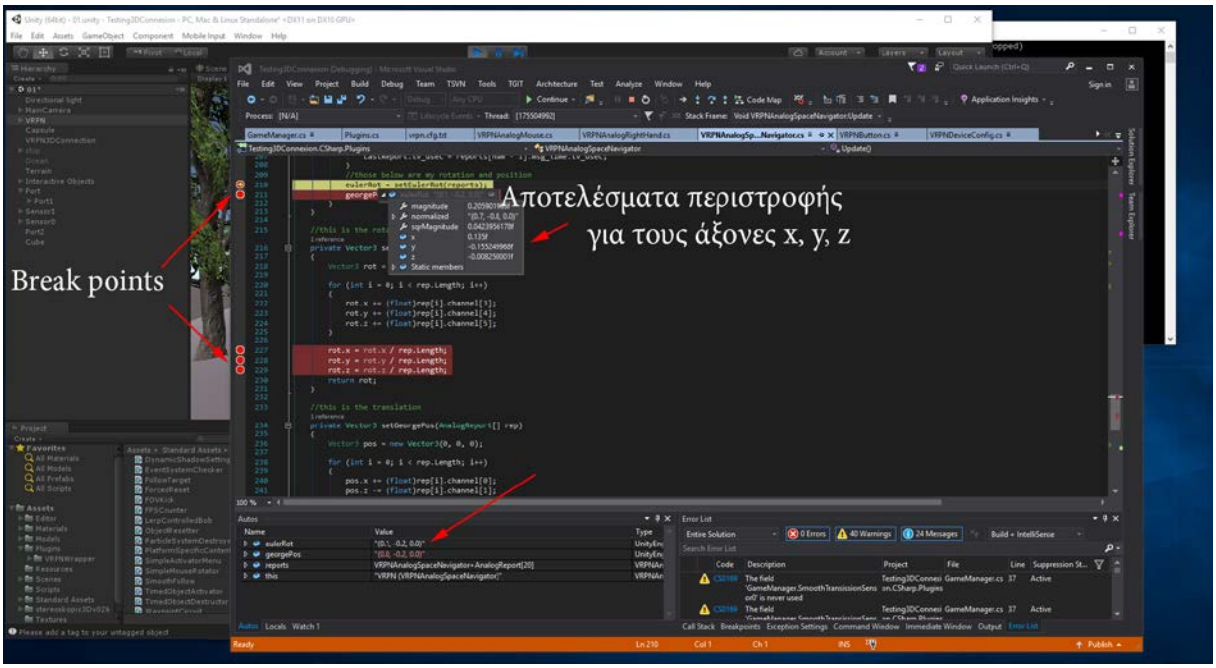
Στην συνέχεια δηλώνουμε άλλες δύο " δημόσιες " (public) μεταβλητές οι οποίες είναι τύπου Vector3 με ονομασίες georgePos (αφορά την θέση) και eulerRot (αφορά την περιστροφή) οι οποίες δέχονται τρία δεδομένα η κάθε μία και χρησιμοποιούνται για την μεταβολή της θέσης και του προσανατολισμού των αντικειμένων στον τρισδιάστατο χώρο.

```
//NTUA Position & Rotation on Public Properties
   public Vector3 eulerRot;
   public Vector3 georgePos;
```

Στην μέθοδο void Update() του κώδικα αποδίδονται τιμές στις δύο νέες παραπάνω μεταβλητές καλώντας τις αντίστοιχες μεθόδους setEulerRot και setGeorgePos με είσοδο τα δεδομένα από τα reports που παράγονται κατά την εκτέλεση (Play) της εφαρμογής.

```
//those below are my rotation and position
eulerRot = setEulerRot(reports);
georgePos = setGeorgePos(reports);
```

Τα reports στον κώδικα περιέχουν ουσιαστικά τις πληροφορίες που στέλνει η συσκευή μας στην σειριακή ή USB σύνδεσή της με τον υπολογιστή. Αυτές οι πληροφορίες δεν είναι τίποτα άλλο από τις τιμές των καναλιών (channels) με φάσμα -1, 0, 1. Για να δούμε κάποια από τα reports όταν κάνουμε debug στον κώδικά χρησιμοποιώντας το Visual Studio πρέπει να βάλουμε break point και με τις συντομεύσεις F5, F9 και F10 να ανατρέξουμε σε αυτά τα σημεία ώστε να μπορούμε να έχουμε γραφικά τα αποτελέσματα.



Γράφουμε τις μεθόδους αυτές ώστε να παίρνουν τα δεδομένα και να επιστρέφουν τον μέσο όρο ως αποτέλεσμα. Για την μέθοδο setEulerRot εργαζόμαστε ως εξής. Την δηλώνουμε ως private, οπότε δεν έχουμε πρόσβαση σε αυτήν μέσα από το Unity3D Editor παρά μόνο από το Visual Studio, είναι τύπου Vector3 (return type), με όνομα προφανώς το setEulerRot και σαν είσοδο (arguments) στην μέθοδο έχω το rep του οποίου ο τύπος (type of) είναι AnalogReport. Το rep λοιπόν περιέχει όλα τα δεδομένα που στέλνει το 3D Connexion Space Navigator και αποθηκεύονται στο AnalogReport οπότε εμείς θα διαχωρίσουμε τα κανάλια (channels) του rep και θα τα συνδέσουμε με την μεταβλητή rot η οποία είναι και αυτή τύπου Vector3. Με την εντολή " **for loop** " γεμίζουμε με τα δεδομένα των καναλιών channel[3], channel[4], channel[5] του rep τα rot.x, rot.y, rot.z ενώ στην συνέχεια τα διαιρούμε με το πλήθος των δεδομένων ενώ στο τέλος λέμε στην μέθοδο να επιστρέψει τα αποτελέσματα (return rot). Τα κανάλια λοιπόν channel[3], channel[4], channel[5] της συσκευής μας στέλνουν δεδομένα που αφορούν την περιστροφή των αξόνων οπότε τελικά ο κώδικας είναι ο παρακάτω :

```
//this is the rotation
   private Vector3 setEulerRot(AnalogReport[] rep)
   {
       Vector3 rot = new Vector3(0, 0, 0);

       for (int i = 0; i < rep.Length; i++)
       {
           rot.x += (float)rep[i].channel[3];
           rot.y += (float)rep[i].channel[4];
           rot.z += (float)rep[i].channel[5];
       }

       rot.x = rot.x / rep.Length;
       rot.y = rot.y / rep.Length;
       rot.z = rot.z / rep.Length;
       return rot;
   }
```

Για την μέθοδο setGeorgePos εργαζόμαστε με την ίδια λογική, η διαφορά είναι πως πλέον εκχωρούμε τις τιμές των καναλιών channel[0], channel[1], channel[2] του rep στις συνιστώσες pos.x, pos.y, pos.z, τα δεδομένα δηλαδή για την μετατόπιση των αντικειμένων (Game Objects), και φυσικά στο τέλος επιστρέφουμε την μεταβλητή pos η οποία είναι και αυτή τύπου Vector3 όπως φαίνεται και στον κώδικα των παρακάτω γραμμών.

```
//this is the translation
    private Vector3 setGeorgePos(AnalogReport[] rep)
    {
        Vector3 pos = new Vector3(0, 0, 0);

        for (int i = 0; i < rep.Length; i++)
        {
            pos.x += (float)rep[i].channel[0];
            pos.z -= (float)rep[i].channel[1];
            pos.y -= (float)rep[i].channel[2];
        }

        pos.x = pos.x / rep.Length;
        pos.y = pos.y / rep.Length;
        pos.z = pos.z / rep.Length;
        return pos;
    }
```

Ο τελικός κώδικας για το αρχείο **VRPNAnalogSpaceNavigator.cs** είναι ο παρακάτω :

```
using UnityEngine;
using System.Collections;
using System;
using System.Runtime.InteropServices;

public class VRPNAnalogSpaceNavigator : MonoBehaviour
{
    // TODO:
    // 1. Should VRPNAnalogStart use the "max" reports variable
    // 2. Create read-only list of analogs and their current values
    // 3. Create event/message driven reporting system

    // VRPN Button Report Structure
    const int MAX_ANALOG_CHANNELS = 128;

    [StructLayout(LayoutKind.Sequential)]
    public struct AnalogReport
    {
        public VRPNManager.TimeVal msg_time;
        public int num_channel;
        [MarshalAs(UnmanagedType.ByValArray, SizeConst = MAX_ANALOG_CHANNELS)]
        public double[] channel;
    }

    // Class Properties
    public static int num_analogs = 0;

    // Public Properties
    public VRPNManager.Analog_Types AnalogType = VRPNManager.Analog_Types.vrpn_3DConnexion_Navigator;
    public VRPNDeviceConfig.Device_Names AnalogName = VRPNDeviceConfig.Device_Names.device0;
    public int MaxReports = 20;
    public bool purgeReports = true;
    public bool useLastReportTime = false;
    public bool ShowDebug = false;
    //NTUA Position & Rotation on Public Properties
    public Vector3 eulerRot;
    public Vector3 georgePos;
```

```
    // Private Variables;
    private bool initialized = false;
    private VRPNManager.TimeVal LastReport;// = new VRPNManager.TimeVal();
    private IntPtr[] reportsPtr;
    private string debug_text = "";
    private int debug_xoffset;

    // DLL Function Imports
    [DllImport("VRPNWrapper/VRPNWrapper")]
    private static extern void VRPNAnalogStart(string name);

    [DllImport("VRPNWrapper/VRPNWrapper")]
    private static extern void VRPNAnalogReport(string name, [In, Out] IntPtr rep, [Out] IntPtr ts, int bu
tton);

    [DllImport("VRPNWrapper/VRPNWrapper")]
    private static extern int VRPNAnalogNumReports(string name);

    [DllImport("VRPNWrapper/VRPNWrapper")]
    private static extern void VRPNAnalogReports(string name, [In, Out] IntPtr[] reportsPtr, [In, Out] ref
 int cnt, [In, MarshalAs(UnmanagedType.LPStruct)] VRPNManager.TimeVal ts, bool clearReport);

    // Functions

    void Start()
    {
        //allocate unmanaged memory for analog reports
        reportsPtr = new IntPtr[MaxReports];
        AnalogReport report = new AnalogReport();
        report.num_channel = MAX_ANALOG_CHANNELS;
        report.channel = new double[MAX_ANALOG_CHANNELS];
        for (int i = 0; i < MaxReports; i++)
        {
            reportsPtr[i] = Marshal.AllocHGlobal(Marshal.SizeOf(typeof(AnalogReport)));
            Marshal.StructureToPtr(report, reportsPtr[i], true);
        }

        // Setup last report time memory
        LastReport = new VRPNManager.TimeVal();
    }

    private bool StartAnalog()
    {
        if (VRPNManager.initialized)
        {
            // Register Analog Device
            VRPNAnalogStart(AnalogName.ToString());
            num_analogs++;

            // Set up debug window
            if (ShowDebug)
            {
                debug_xoffset = num_analogs * 210;
                if (VRPNManager.debug_flag)
                    debug_xoffset += 405;
            }
            initialized = true;

            return true;
        }
        return false;
    }

    // Update is called once per frame
    void Update()
    {
        // Ensure device is ready
        if (!initialized && !StartAnalog()) return;
```

```
        // Check for new reports and process
        if (VRPNAnalogNumReports(AnalogName.ToString()) > 0)
        {
            // Get Reports
            int num = MaxReports;
            VRPNAnalogReports(AnalogName.ToString(), reportsPtr, ref num, LastReport, purgeReports);
            AnalogReport[] reports = new AnalogReport[num];

            // Process Reports
            int i;
            string reportString = AnalogName.ToString();
            string messageString;
            for (i = 0; i < num; i++)
            {
                reports[i] = (AnalogReport)Marshal.PtrToStructure(reportsPtr[i], typeof(AnalogReport));
                messageString = "<";
                for (int j = 0; j < reports[i].num_channel; j++)
                {
                    messageString += " " + reports[i].channel[j] + ",";
                }
                messageString += ">" + " @ " + reports[i].msg_time.tv_sec + "." + reports[i].msg_time.tv_u
sec;
                if (ShowDebug) reportString += "/n" + messageString;
            }

            if (ShowDebug) debug_text = reportString;

            // Only need time value of most recent report
            if (num > 0 && useLastReportTime)
            {
                LastReport.tv_sec = reports[num - 1].msg_time.tv_sec;
                LastReport.tv_usec = reports[num - 1].msg_time.tv_usec;
            }
            //those below are my rotation and position
            eulerRot = setEulerRot(reports);
            georgePos = setGeorgePos(reports);
        }
    }

    //this is the rotation
    private Vector3 setEulerRot(AnalogReport[] rep)
    {
        Vector3 rot = new Vector3(0, 0, 0);

        for (int i = 0; i < rep.Length; i++)
        {
            rot.x += (float)rep[i].channel[3];
            rot.y += (float)rep[i].channel[4];
            rot.z += (float)rep[i].channel[5];
        }

        rot.x = rot.x / rep.Length;
        rot.y = rot.y / rep.Length;
        rot.z = rot.z / rep.Length;
        return rot;
    }

    //this is the translation
    private Vector3 setGeorgePos(AnalogReport[] rep)
    {
        Vector3 pos = new Vector3(0, 0, 0);

        for (int i = 0; i < rep.Length; i++)
        {
            pos.x += (float)rep[i].channel[0];
            pos.z -= (float)rep[i].channel[1];
            pos.y -= (float)rep[i].channel[2];
        }
```

```
        pos.x = pos.x / rep.Length;
        pos.y = pos.y / rep.Length;
        pos.z = pos.z / rep.Length;
        return pos;
    }

    void OnGUI()
    {
        if (ShowDebug)
        {
            GUI.skin.box.alignment = TextAnchor.LowerLeft;
            GUI.Box(new Rect(debug_xoffset + 10, 20, 600, 45), debug_text);
        }
    }

}
```

Εφόσον έχουμε πλέον φτιάξει τα αρχεία και έχουμε επέμβει στον κώδικα του **VRPNAnalogSpaceNavigator.cs** ήρθε η στιγμή να χρησιμοποιήσουμε το 3D Connexion Space Navigator σε ένα αντικείμενο (Game Object) του Unity3D.

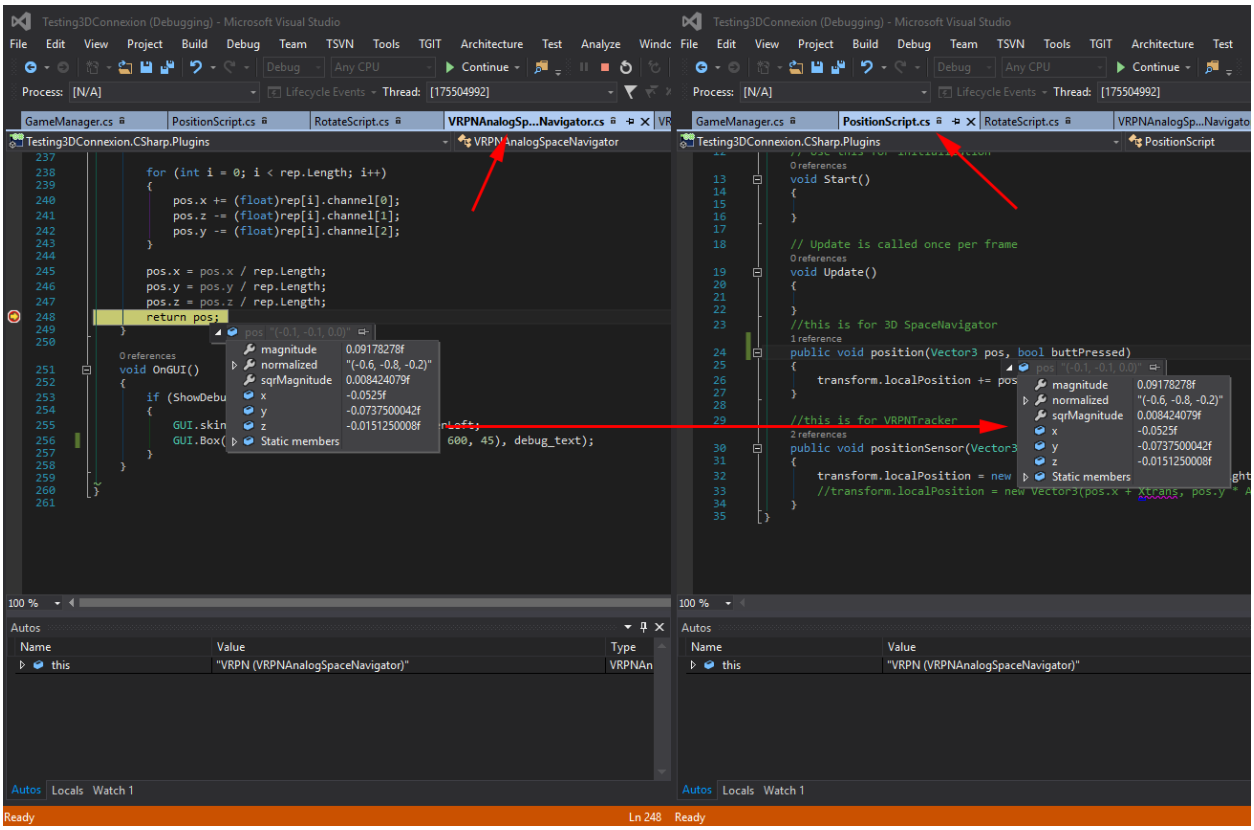
Κάνουμε λοιπόν δεξί κλικ στο Project window του Unity στην διαδρομή Assets\Plugins\VRPNWrapper και φτιάχνουμε έναν φάκελο με την ονομασία MyScripts ώστε να τον ξεχωρίζουμε από τους υπόλοιπους φακέλους. Με δεξί κλικ μέσα στον φάκελο αυτό δημιουργούμε δύο νέα C# script με τις ονομασίες **PositionScript.cs** και **RotateScript.cs** τα οποία θα ενσωματώσουμε στα αντικείμενα που θέλουμε να μετακινήσουμε.

Στο αρχείο PositionScript.cs εισάγουμε μια δημόσια (public) μεταβλητή με την ονομασία posSpeed ώστε να μπορούμε μεταβάλλοντας την τιμή της να ελέγχουμε την ταχύτητα της περιστροφής του αντικειμένου και δύο μεθόδους μέσα στην μέθοδο void Update(){} με την ονομασία position, η οποία αφορά μετακίνηση με την βοήθεια του 3D Connexion Space Navigator, και positionSensor, η οποία αφορά μετακίνηση με την βοήθεια του συστήματος Flock Of Birds για το οποίο θα μιλήσουμε στο σχετικό κεφάλαιο. Στο παρακάτω κείμενο διακρίνεται ο κώδικας ολοκληρωμένος.

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

public class PositionScript : MonoBehaviour
{
    public bool stopOnButton = false;
    public float posSpeed = 5f;
    //this is for the avatar height
    public float AvatarHeight = 5f;
    // Use this for initialization
    void Start()
    {

    }

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

    }
    //this is for 3D SpaceNavigator
    public void position(Vector3 pos, bool buttPressed)
    {
        transform.localPosition += pos * posSpeed * Time.deltaTime;
```

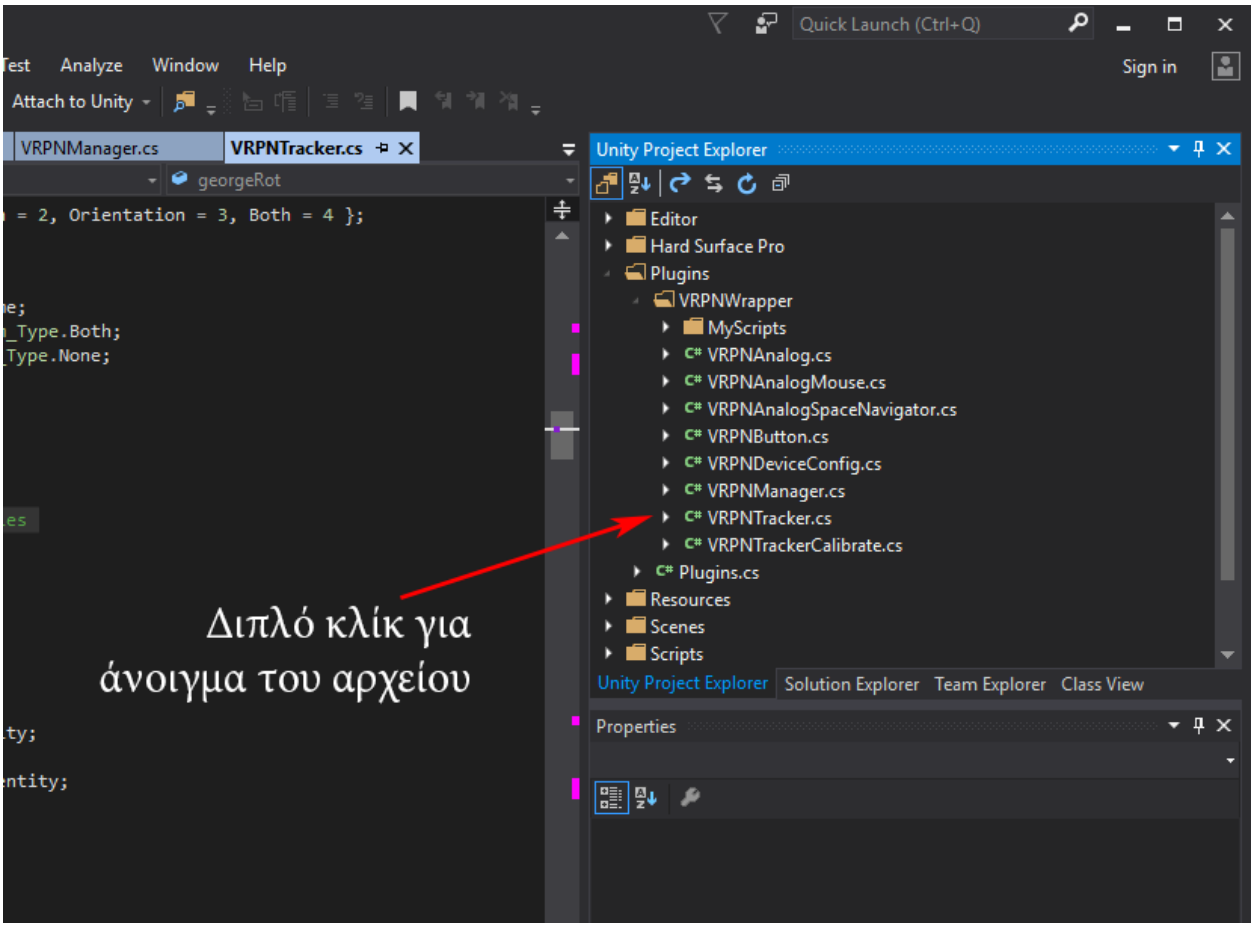
```
    }

    //this is for VRPNTracker
    public void positionSensor(Vector3 pos)
    {
        transform.localPosition = new Vector3(pos.x, pos.y + AvatarHeight, pos.z);
        //transform.localPosition = new Vector3(pos.x + Xtrans, pos.y * AvatarHeight, pos.z + Ztrans);
    }
}
```

Στην μέθοδο position έχουμε δύο δεδομένα εισόδου το Vector3 pos και το bool buttPressed. Το bool buttPressed αφορά το πάτημα ή όχι του κουμπιού που διαθέτει το 3D Connexion Space Navigator, αυτό όμως που μας ενδιαφέρει στην συγκεκριμένη φάση είναι το Vector3 pos το οποίο δεν είναι τίποτα άλλο από την έξοδο της μεθόδου **setGeorgePos** την οποία φτιάξαμε στο **VRPNAnalogSpaceNavigator.cs**. Η παρακάτω εικόνα μας δείχνει πως τα δεδομένα του pos μεταφέρονται στο **PositionScript.cs**.



6.2.VRPNTracker.cs του Flock Of Birds

Σε αυτό το κεφάλαιο θα ασχοληθούμε με την ανάλυση του κώδικα των αρχείων scripts που χρησιμοποιούμε στο σύστημα FOB . Η μεθοδολογία που ακολουθούμε είναι παρόμοια με αυτή του 3D Connexion Space Navigator. Κάνουμε λοιπόν διπλό κλικ μέσα από το Unity3D στο όνομα του script που θέλουμε να δουλέψουμε και το ανοίγουμε με το Visual Studio ή μπορούμε, εφόσον έχουμε ρυθμίσει το Visual Studio, από την καρτέλα Unity Project Explorer να μεταβούμε στο αρχείο που μας ενδιαφέρει και να το επεξεργαστούμε, όπως διακρίνεται και στην παρακάτω εικόνα.



Στο αρχείο **VRPNTracker.cs** εισάγουμε δύο νέες δημόσιες μεταβλητές ώστε να είναι ορατές στο Unity3D και να έχουμε οπτική επαφή με τις τιμές τους :

```
//NTUA Position and Rotation on Public Properties
   public Quaternion georgeRot;
   public Vector3 georgePos;
```

Έπειτα δηλώνουμε τι θα περιέχουν οι μεταβλητές αυτές. Από την μέθοδο **void Update() {}** εκμεταλλευόμαστε την **ReposRetriever(){}** για να πάρουμε τα τελευταία reports (**reps**) του κώδικα τα οποία περιέχουν όλες τις αριθμητικές πληροφορίες που στέλνουν οι συσκευές Bird στην σειριακή θύρα του υπολογιστή COM2. Διαχωρίζουμε με δύο νέες μεθόδους, οι οποίες έχουν ως είσοδο τη μεταβλητή reps, με ονομασίες georgePos και georgeRot**,** την θέση και τον προσανατολισμό αντίστοιχα.

```
//NTUA below is my configuration
    public void ReposRetriever()
    {
        TrackerReport[] reps = GetAveragePosQuat();

        georgeRot = setRot(reps);
        georgePos = setPos(reps);
    }
```

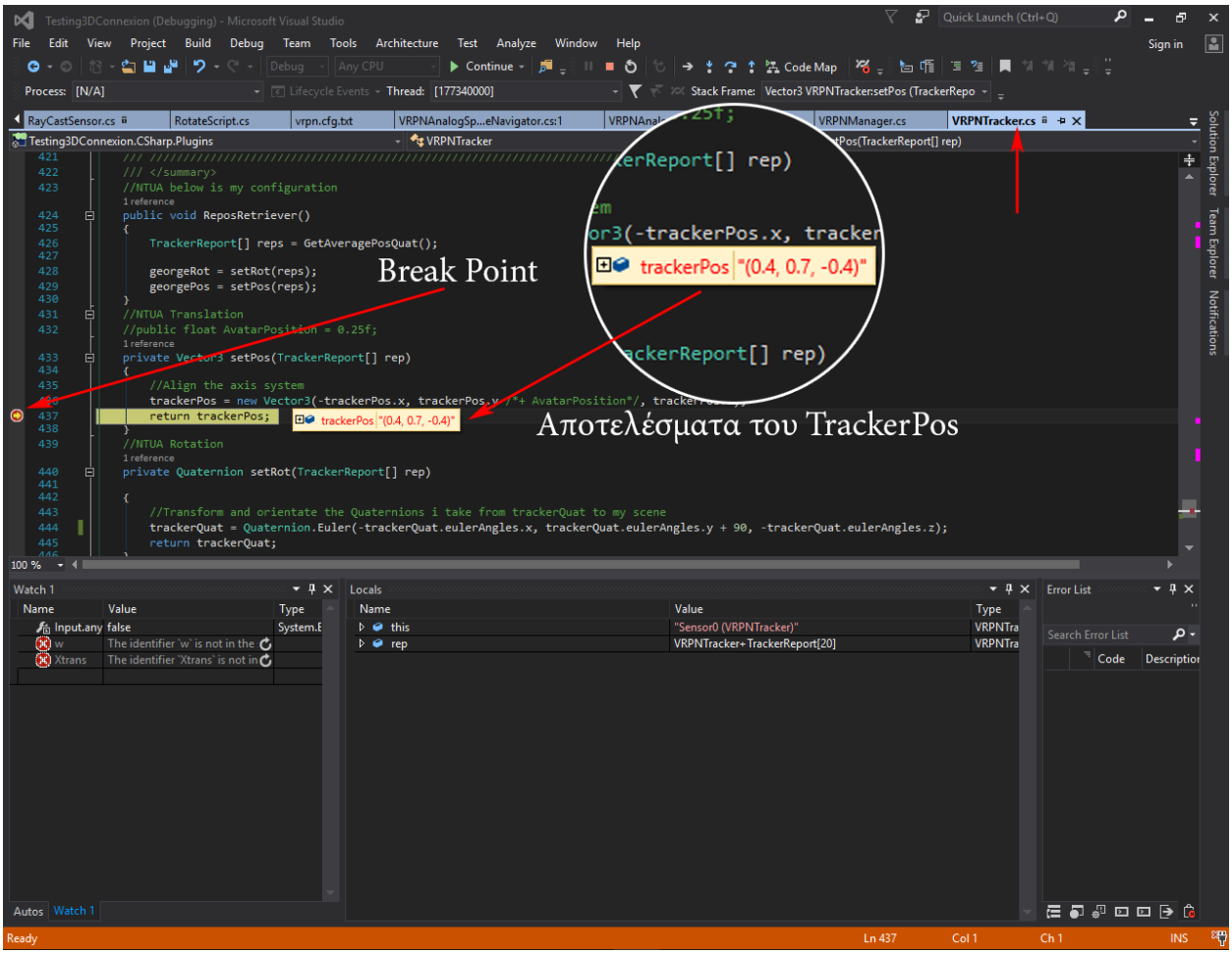
Το setPos είναι μια ιδιωτική μέθοδος τύπου Vector3 με είσοδο τη μεταβλητή rep η οποία είναι τύπου TrackerReport[]. Μέσα στην μέθοδο αυτή δηλώνουμε τη μεταβλητή trackerPos ως ένα νέο Vector3 με παραμέτρους

- trackerPos.x
- trackerPos.y
- trackerPos.z

και επιστρέφουμε τις τιμές που λαμβάνει με την εντολή return trackerPos. Σαν παρατήρηση βλέπουμε ότι στην παράμετρο trackerPos.x έχουμε προσθέσει το μείον (–) οπότε τελικά έχουμε -trackerPos.x. Με αυτό το τρόπο αλλάζουμε το πρόσημο των τιμών που λαμβάνουμε στον άξονα x. Έτσι δεν αλλάζουμε την φορά του ηλεκτρομαγνητικού πεδίου της συσκευής ERT, περιστρέφοντάς την με φυσικό τρόπο, αλλάζουμε την φορά που εμείς θέλουμε να βρίσκονται οι θετικές ή αρνητικές τιμές σε έναν άξονα.

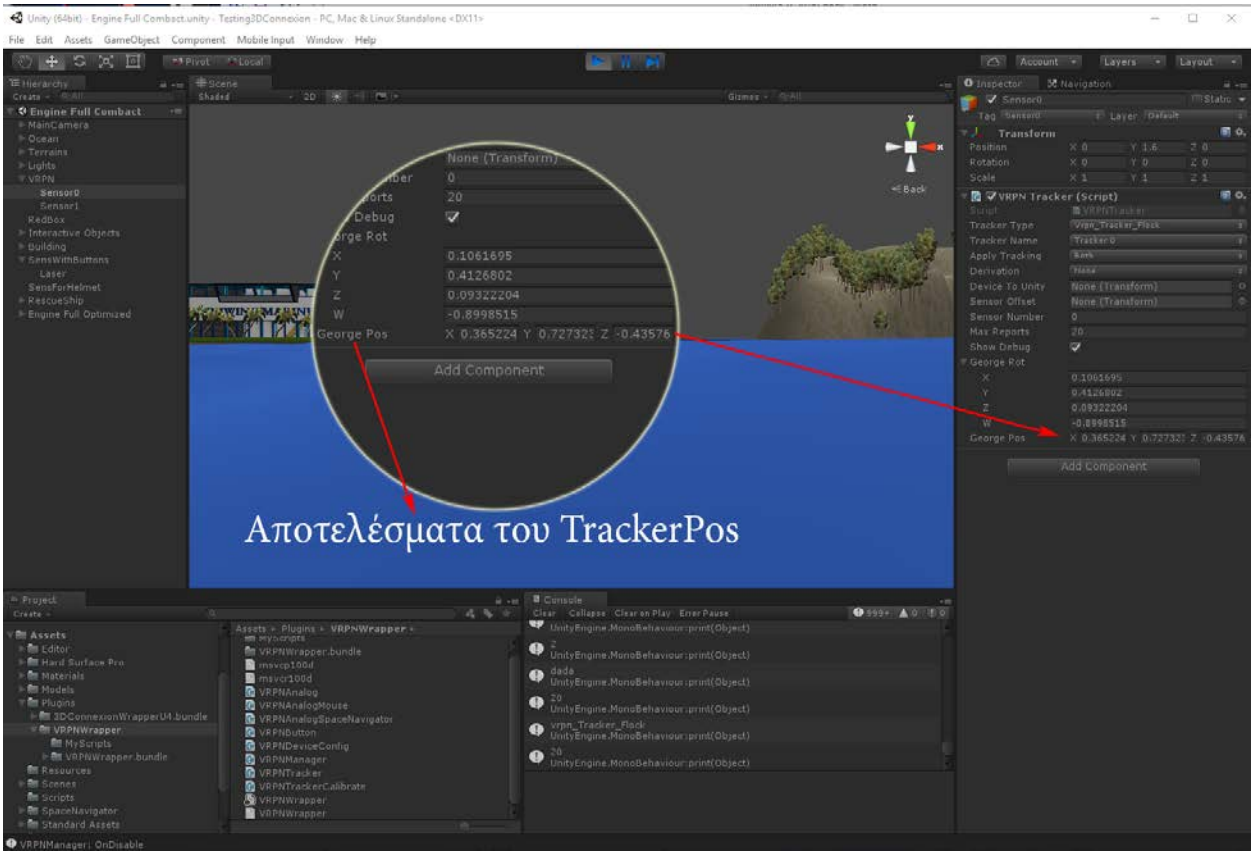
```
private Vector3 setPos(TrackerReport[] rep)
    {
        //Align the axis system
        trackerPos = new Vector3(-trackerPos.x, trackerPos.y /*+ AvatarPosition*/, trackerPos.z);
        return trackerPos;
    }
```

Για να δούμε τις τιμές αυτές στο Visual Studio κάνοντας Debug βάζουμε ένα Break Point στην γραμμή 437 return trackerPos οπότε λαμβάνουμε ως δεδομένα για τους άξονες $x = 0.4,\ y = 0.7,\ z = -0.4$. Τα δεδομένα αυτά αντιστοιχούν σε μέτρα (meter) και είναι η θέση που βρίσκεται ένας αισθητήρας σε σχέση με την συσκευή ERT δηλαδή σε σχέση με το τρισορθογώνιο σύστημα αναφοράς του ηλεκτρομαγνητικού πεδίου.

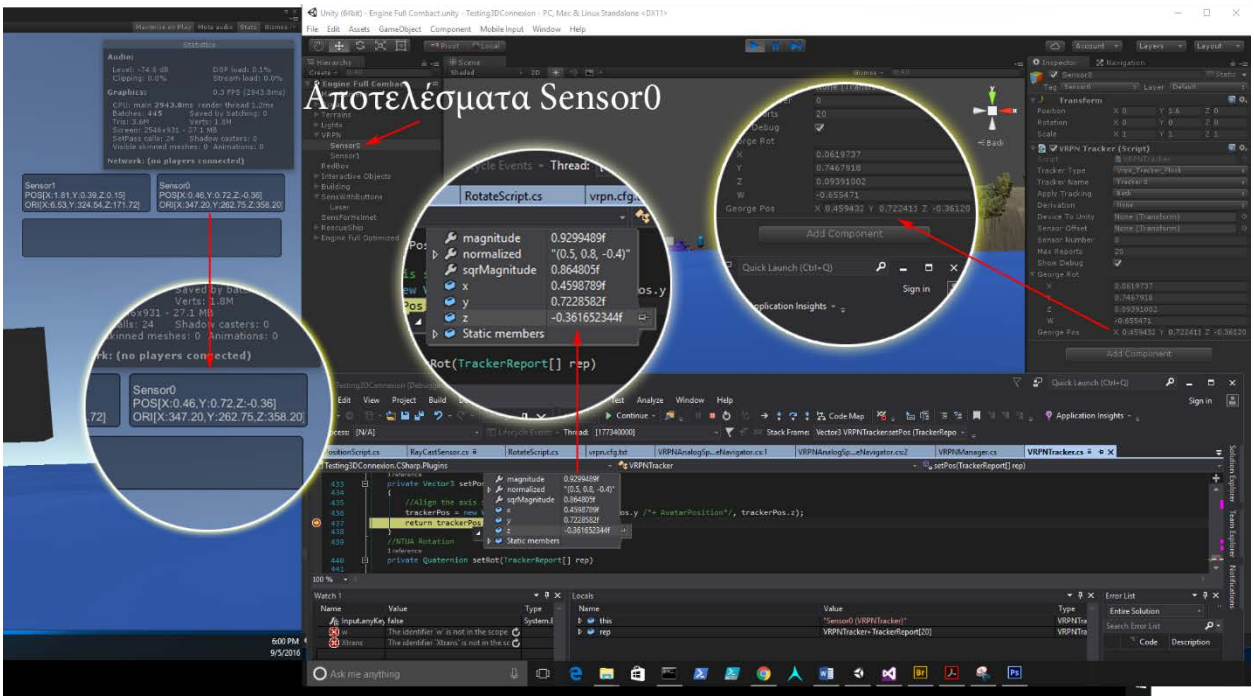


Αποτελέσματα trackerPos

Φυσικά τα αποτελέσματα που παίρνουμε στο Visual Studio αναμένουμε να είναι τα ίδια με αυτά στο Unity3D, κάτι το οποίο και συμβαίνει. Επιλέγουμε λοιπόν στο Hierarchy window του Unity3D το κενό αντικείμενο Sensor0 και στο Visual Studio επιλέγουμε Attach to Unity and Run και περιμένουμε να ξεκινήσει αυτόματα η εφαρμογή. Εφόσον συμβεί αυτό πατάμε την συντόμευση F5, με το Visual Studio πλέον ενεργό παράθυρο, και παίρνουμε τις ίδιες τιμές στην δημόσια μεταβλητή georgePos. Οι μικρές διαφορές στις τιμές οφείλονται στο γεγονός ότι το Visual Studio κάνει στρογγυλοποίηση των αριθμών. Πρέπει τέλος να σημειωθεί πως και σε αυτή την περίπτωση όπως και στο 3D Connexion Space Navigator το εύρος των τιμών είναι μεταξύ $-1, 0, 1$.



Στην παρακάτω εικόνα βλέπουμε τα ίδια αποτελέσματα για τον Sensor0 στο Visual Studio, Unity3D αλλά και στην έξοδο του Show Debug του κώδικα μέσω του Game window του Unity3D κάτι το οποίο δηλώνει ότι η συσκευή μας στέλνει σωστά τα αποτελέσματα μέσω του VRPN Server.



Το setRot είναι μια ιδιωτική μέθοδος τύπου Quaternion με είσοδο και πάλι τη μεταβλητή rep. Μέσα στην μέθοδο αυτή δηλώνουμε το trackerQuat ως μία νέα μεταβλητή τύπου Quaternion με παραμέτρους

- trackerQuat.eulerAngles.x
- trackerQuat.eulerAngles.y
- trackerQuat.eulerAngles.z

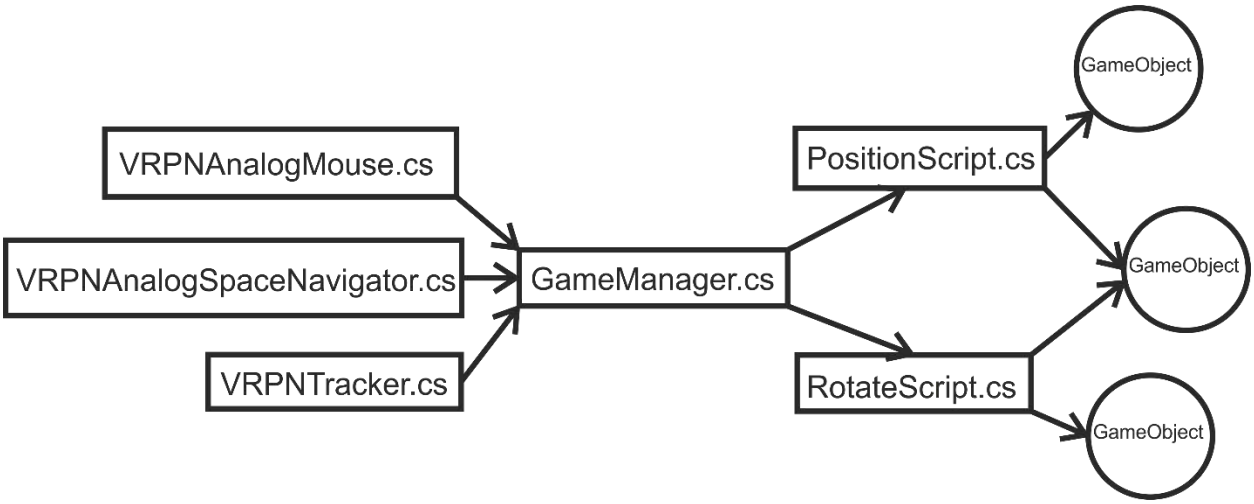
και πάλι επιστρέφουμε τις τιμές που λαμβάνει με την εντολή return trackerQuat.

Εδώ παρατηρούμε με τις παραμέτρους -trackerQuat.eulerAngles.x και -trackerQuat.eulerAngles.z ότι έχουμε αλλάξει το πρόσημο περιστροφής του άξονα x και z αντίστοιχα ενώ στην παράμετρο trackerQuat.eulerAngles.y προσθέτουμε σε κάθε αλλαγή της μεταβλητής +90 μοίρες. Οι τιμές και τα πρόσημα των παραμέτρων αλλάζουν αναλόγως την θέση και τον προσανατολισμό της μονάδας ERT και του ηλεκτρομαγνητικού ημισφαιρίου που αυτή δημιουργεί, οπότε ο χρήστης των συσκευών αυτών μπορεί να πειραματιστεί με διάφορες τιμές και πρόσημα ώστε να καταλήξει σε τιμές που εξυπηρετούν την εκάστοτε πειραματική διάταξη.

```
private Quaternion setRot(TrackerReport[] rep)

    {
        //Transform and orientate the Quaternions i take from trackerQuat to my scene
        trackerQuat = Quaternion.Euler(-trackerQuat.eulerAngles.x, trackerQuat.eulerAngles.y + 90, -
trackerQuat.eulerAngles.z);
        return trackerQuat;
     }
```

6.3.GameManager.cs

Ο τρόπος διαχείρισης των πληροφοριών που λαμβάνουμε από τις περιφερειακές συσκευές μας είναι το κλειδί για την σωστή εκμετάλλευσή τους. Αποφασίσαμε λοιπόν να δημιουργήσουμε ένα αρχείο script C# με την ονομασία **GameManager.cs** το οποίο θα είναι υπεύθυνο για την διοχέτευση των πληροφοριών αυτών στα αντικείμενα (Game Objects) του Unity3D. Κάνουμε λοιπόν δεξί κλικ στο Project windows και συγκεκριμένα στην διαδρομή Essets\Plugins\VRPNWrapper\Myscripts και επιλέγουμε Create -> C# Script. Στο άδειο αντικείμενο **VRPN** ενσωματώνουμε το **GameManager.cs** και το ανοίγουμε με το Visual Studio για επεξεργασία. Για να γίνει κατανοητή η εφαρμογή του **GameManager.cs** παραθέτουμε γραφικά τον τρόπο με τον οποίο επιθυμούμε να αλληλοεπιδράσει με τα άλλα scripts και Game Objects του Unity3D.



6.3.1. Delegates

Μια αρκετά σημαντική ιδιότητα της γλώσσας C# ονομάζεται delegate την οποία χρησιμοποιούμε ώστε να διαχειριζόμαστε τις μεθόδους (Functions) ως μεταβλητές (Variables). Στο Project που έχουμε φτιάξει με το Unity3D έχουμε αρκετά αντικείμενα (Game Objects) τα οποία θέλουμε να μετακινήσουμε και να περιστρέψουμε με την βοήθεια των συσκευών εικονικής πραγματικότητας που διαθέτει το εργαστήριο. Έτσι πρέπει να βρούμε ένα έξυπνο τρόπο ώστε να πληροφορήσουμε τα αντικείμενα αυτά για κάποια συμβάντα (events) τα οποία προκύπτουν κατά την εκτέλεση (Play) της εφαρμογής. Τα συμβάντα αυτά δεν είναι τίποτα άλλο από την μετακίνηση και περιστροφή των αξόνων του 3D Connexion Space Navigator αλλά και των αισθητήρων του συστήματος FOB.

6.3.1.1. Delegates για το 3D Connexion Space Navigator

Για να πάρουμε λοιπόν τα δεδομένα από το **VRPNSpaceNavigator.cs** εισάγουμε στο **GameManager.cs** τα παρακάτω delegate με δομή :

```
//3D Space Navigator
    public delegate void position(Vector3 pos, bool buttPressed);
    public position myPosition;

    public delegate void rotate(Vector3 rot, bool buttPressed);
    public rotate myRotate;

    private VRPNAnalogSpaceNavigator VRPNavigator;
    private VRPNButton VRPNBut;
```

Το πρώτο delegate το ονομάζουμε position**,** επιστρέφει κενό **(void)** και έχει είσοδο το pos το οποίο είναι Vector3 και το buttPressed το οποίο είναι λογικός τελεστής Boolean (true - false) όπου με την σειρά του δηλώνει αν πατήθηκε ή όχι το κουμπί του 3D Connexion Space Navigator. Ακριβώς από κάτω δηλώνουμε μία μεταβλητή τύπου delegate position και της δίνουμε το όνομα myPosition. Ακριβώς την ίδια τακτική και δομή ακολουθούμε για το επόμενο delegate που αφορά την περιστροφή με όνομα myRotate.

Πρέπει να αναφέρουμε ότι ο τύπος του delegate που φτιάξαμε πρέπει να είναι ίδιος με τον τύπο της μεθόδου που θέλουμε να καλέσουμε η οποία βρίσκεται σε ξεχωριστό script το **VRPNSpaceNavigator.cs** με όνομα **georgePos**, θέλουμε δηλαδή να επιστρέφει κενό (void) και να έχει είσοδο ένα Vector3 τα οποία και έχουμε δηλώσει.

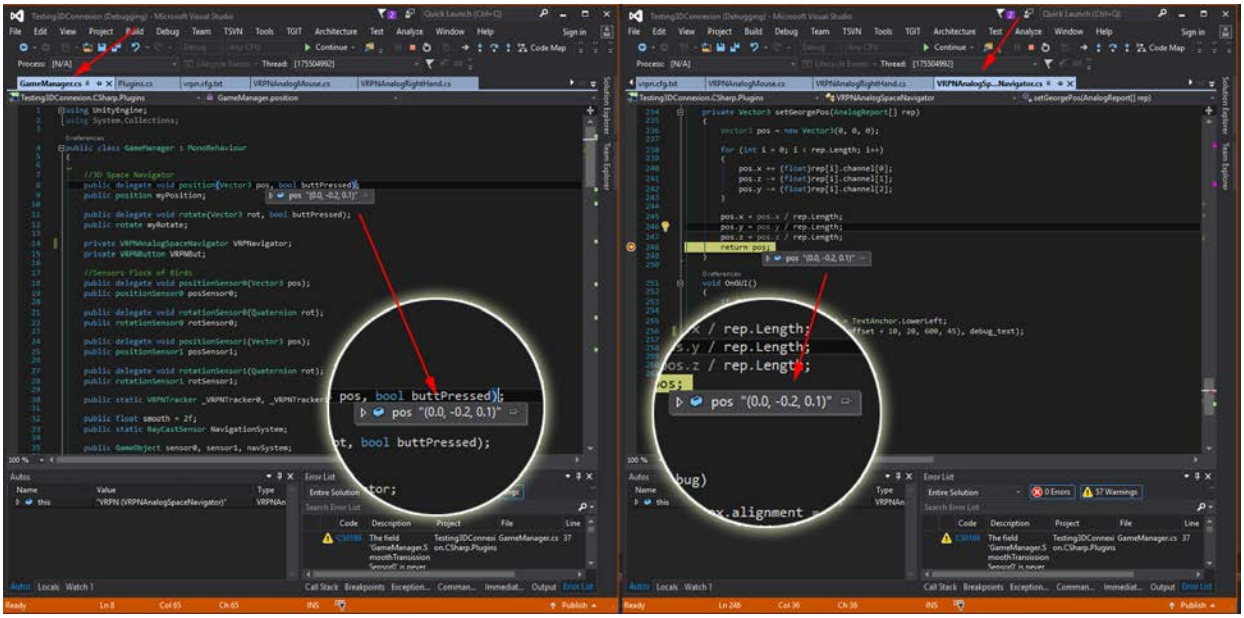
Στην συνέχεια δηλώνουμε δύο ιδιωτικές (private) μεταβλητές**,** την **VRPNavigator** η οποία είναι τύπου **VRPNSpaceNavigator,** ανήκει δηλαδή στην κλάση **VRPNSpaceNavigator.cs** και την **VRPNBut** τύπου **VRPNButton**, ανήκει δηλαδή στην κλάση **VRPNButton.cs**. Σε μια νέα μέθοδο με όνομα void Awake(){} παίρνουμε όλα τα συστατικά (Components) από τα scripts **VRPNSpaceNavigator.cs** και **VRPNButton.cs** ώστε να έχουμε πρόσβαση σε αυτά μέσω των μεταβλητών **VRPNavigator** και **VRPNBut** που δηλώσαμε πιο πάνω.

```
void Awake()
    {
        //Pick up all components from VRPNAnalogSpaceNavigator & VRPNButton (3D Space Navigator)
        VRPNavigator = GetComponent<VRPNAnalogSpaceNavigator>();
        VRPNBut = GetComponent<VRPNButton>();

    }
```

Εφόσον έχω πρόσβαση σε όλα τα συστατικά των **VRPNSpaceNavigator.cs** και **VRPNButton.cs** στην μέθοδο Update(){} δηλώνω τα delegate myPosition και myRotate να πάρουν από τα νέα αντικείμενα (Objects) **VRPNavigator** και **VRPNBut** τις μεθόδους georgePos(){}, eulerRot(){} και Button1Pressed(){}.

```
// Update is called once per frame
    void Update()
    {
        //3D Space Navigator update the incoming data form the device
        myRotate(VRPNavigator.eulerRot, VRPNBut.Button1Pressed);
        myPosition(VRPNavigator.georgePos, VRPNBut.Button1Pressed);

        //if objects are untagged print this message
        if (untaggedObjectExists)
        {
            Debug.Log("Please add a tag to your untagged object");
        }
```

Στην παρακάτω εικόνα βλέπουμε το **VRPNSpaceNavigator.cs** να στέλνει τα δεδομένα στο **GameManager.cs** ο οποίος με τη σειρά του τροφοδοτεί με τα δεδομένα αυτά τα script **PositionScript.cs** και **RotateScript.cs** τα οποία χρησιμοποιούμε για την μετακίνηση και περιστροφή των αντικειμένων στα οποία είναι ενσωματωμένα.



6.3.1.2. Delegates για το Flock Of Birds

Το σύστημα Flock of Birds είναι αρκετά πιο περίπλοκο γιατί περιλαμβάνει δύο αισθητήρες και πρέπει να γίνει διαχωρισμός τους. Για κάθε αισθητήρα λοιπόν θα δημιουργήσουμε δύο **delegates** ένα για την θέση και ένα για τον προσανατολισμό του. Έτσι λοιπόν στο script **GameManager.cs** εισάγουμε τέσσερα delegates με την δομή που φαίνεται παρακάτω :

```
//Sensors Flock of Birds
    public delegate void positionSensor0(Vector3 pos);
    public positionSensor0 posSensor0;

    public delegate void rotationSensor0(Quaternion rot);
    public rotationSensor0 rotSensor0;

    public delegate void positionSensor1(Vector3 pos);
    public positionSensor1 posSensor1;

    public delegate void rotationSensor1(Quaternion rot);
    public rotationSensor1 rotSensor1;

    public static VRPNTracker _VRPNTracker0, _VRPNTracker1;
    public GameObject sensor0, sensor1, navSystem;
```

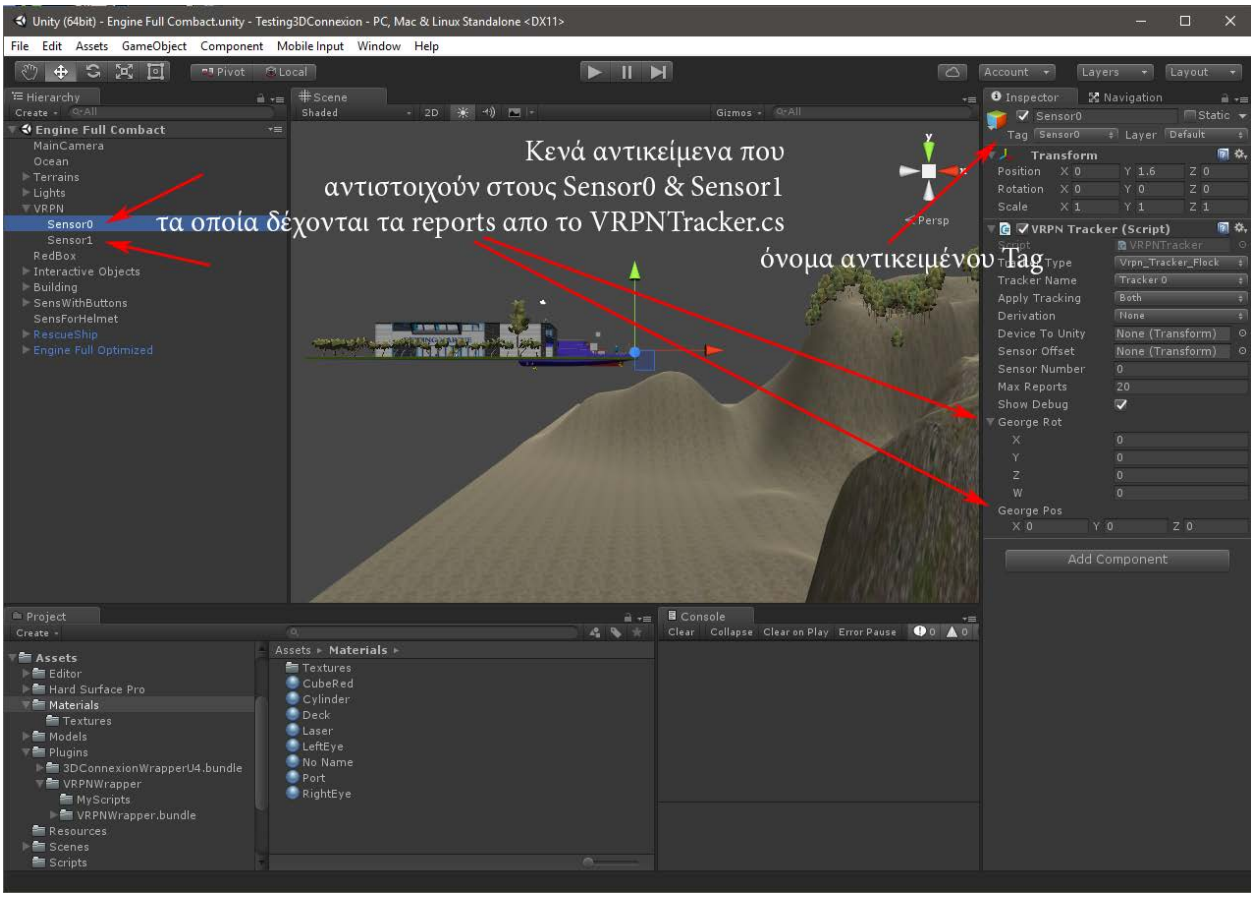
Το πρώτο delegate για την θέση αφορά τον απλό αισθητήρα Sensor0**,** έχει όνομα positionSensor0 και ως είσοδο έχει τη μεταβλητή pos του VRPNTracker η οποία είναι τύπου Vector3. Ακριβώς από κάτω δηλώνουμε (instantiate) μία μεταβλητή τύπου delegate positionSensor0 και της δίνουμε το όνομα posSensor0. Το δεύτερο delegate αφορά τον αισθητήρα με το ενσωματωμένο ποντίκι Sensor1, έχει όνομα positionSensor1 και ως είσοδο έχει τη μεταβλητή pos του VRPNTracker η οποία είναι και αυτή τύπου Vector3. Ακριβώς από κάτω δηλώνουμε (instantiate) μία μεταβλητή τύπου positionSensor1 και της δίνουμε το όνομα posSensor1.

Το πρώτο delegate για τον προσανατολισμό του απλού αισθητήρα Sensor0 έχει όνομα rotationSensor0 και σαν είσοδο έχει τη μεταβλητή rot η οποία είναι τύπου Quaternion. Ακριβώς από κάτω δηλώνουμε (instantiate)) μία μεταβλητή τύπου delegate rotationSensor0 και της δίνουμε το όνομα rotSensor0.

Για τον προσανατολισμό του αισθητήρα με το ενσωματωμένο ποντίκι πράττουμε ακριβώς τα ίδια με την διαφορά όπου 0 βάζουμε το 1.

6.3.2.Μεταβλητές συστήματος

Στην συνέχεια δηλώνουμε δύο δημόσιες μεταβλητές με ονόματα _VRPNTracker0 και _VRPNTracker1 τύπου VRPNTracker, μια μεταβλητή με όνομα NavigationSystem τύπου RayCastSensor και τρία αντικείμενα (GameObject) με ονόματα sensor0, sensor1 και navSystem. Στην μέθοδο void Awake(){} λέμε στο αντικείμενο sensor0 (GameObject) να βρει ένα αντικείμενο στο Unity3D με όνομα Tag "Sensor0" και αντίστοιχα για το sensor1 να βρει ένα αντικείμενο με όνομα Tag "Sensor1". Τα "Sensor0" και "Sensor1" είναι τα κενά αντικείμενα Sensor0 και Sensor1 που έχουμε ενσωματώσει και δηλώσει ως παιδιά (child) στο επίσης κενό αντικείμενο VRPN του Unity3D τα οποία διακρίνονται στο Hierarchy window και φαίνονται στην εικόνα που ακολουθεί.



Έτσι εκχωρούνται στις μεταβλητές _VRPNTracker0 και _VRPNTracker1 τα συστατικά (Components) των αντικειμένων Sensor0 και Sensor1. Το navSystem αντίστοιχα θα βρει το αντικείμενο με όνομα Tag "VRPNTrackSensor1", το οποίο είναι το SensWithButtons του Unity3D και αντιστοιχεί στον αισθητήρα με το ενσωματωμένο ποντίκι, οπότε στο NavigationSystem εκχωρούνται τα συστατικά (Components) του **RayCastSensor.cs**. Το **RayCastSensor.cs** είναι ένα C# script το οποίο ενσωματώνουμε στο κενό αντικείμενο SensWithButtons και μας βοηθάει στην μετακίνηση του εικονικού εαυτού μας μέσα στην σκηνή (Scene) της εφαρμογής – παιχνιδιού το οποίο θα αναλύσουμε αργότερα. Η τελική μορφή του τμήματος του κώδικα που αφορά την μέθοδο void Awake(){} διακρίνεται παρακάτω.

```
void Awake()
    {
        //Pick up all components from VRPNAnalogSpaceNavigator & VRPNButton (3D Space Navigator)
        VRPNavigator = GetComponent<VRPNAnalogSpaceNavigator>();
        VRPNBut = GetComponent<VRPNButton>();

        //Pick up all components from VRPNTracker (Flock Of Birds)
        sensor0 = GameObject.FindGameObjectWithTag("Sensor0");
        sensor1 = GameObject.FindGameObjectWithTag("Sensor1");
        _VRPNTracker0 = sensor0.GetComponent<VRPNTracker>();
        _VRPNTracker1 = sensor1.GetComponent<VRPNTracker>();

        //Set sensors position by clicking in the navigation mesh
        navSystem = GameObject.FindGameObjectWithTag("VRPNTrackSensor1");
        NavigationSystem = navSystem.GetComponent<RayCastSensor>();
    }
```

6.3.3.Πίνακες (arrays)

Για να στείλουμε τα δεδομένα που λαμβάνουμε από τα script **VRPNAnalogSpaceNavigator.cs, VRPNTracker.cs** και **VRPNButton.cs** στα script **PositionScript.cs** και **RotateScript.cs** μέσω του **GameManager.cs**, με την βοήθεια της μεθόδου delegate, πρέπει στην μέθοδο void Start(){} να δημιουργήσουμε πίνακες (arrays -> []).

Όσον αφορά τα δεδομένα που λαμβάνουμε για τον προσανατολισμό των αισθητήρων του συστήματος FOB αλλά και του 3D Connexion Space Navigator δημιουργούμε έναν πίνακα [] τύπου RotateScript τον οποίο ονομάζουμε objectsRotateScript. Με την εντολή Object.FindObjectsOfType εντοπίζουμε τα αντικείμενα αυτού του τύπου και τα αποθηκεύουμε στον πίνακα. Με τον επαναληπτικό βρόχο foreach διατρέχουμε τα αποθηκευμένα αντικείμενα και για κάθε ένα από αυτά, αναλόγως τι όνομα Tag έχουν, εκτελούμε τον αντίστοιχο κώδικα για την περιστροφή τους. Οι περιπτώσεις των ονομάτων Tags διαχωρίζονται σε "Interactive" τα οποία είναι τα διαδραστικά αντικείμενα, σε "VRPNTrackSensor0" που είναι το αντικείμενο που έχουμε ενσωματώσει στην κάμερα και σε "VRPNTrackSensor1", δηλαδή το αντικείμενο του αισθητήρα με το ποντίκι. Τα δεδομένα που λαμβάνουμε για την θέση των αισθητήρων του συστήματος FOB και του 3D Connexion Space Navigator τα διαχειριζόμαστε με παρόμοιο τρόπο όπως διακρίνεται στις γραμμές του παρακάτω κώδικα δημιουργώντας πίνακα [] τύπου PositionScript.

```
// Use this for initialization
    void Start()
    {
         //this is for rotation
        RotateScript[] objectsRotateScript = Object.FindObjectsOfType(typeof(RotateScript)) as RotateScrip
t[];
        foreach (RotateScript obj in objectsRotateScript)
        {
            switch (obj.tag)
            {
                case "Interactive":
                    myRotate += obj.rotate;
                    break;
                case "VRPNTrackSensor0":
                    rotSensor0 += obj.rotationSensor;
                    break;
                case "VRPNTrackSensor1":
                    rotSensor1 += obj.rotationSensor;
                    break;
                default:
                    Debug.Log("Please add a tag to your object");
                    untaggedObjectExists = true;
                    break;
             }
```

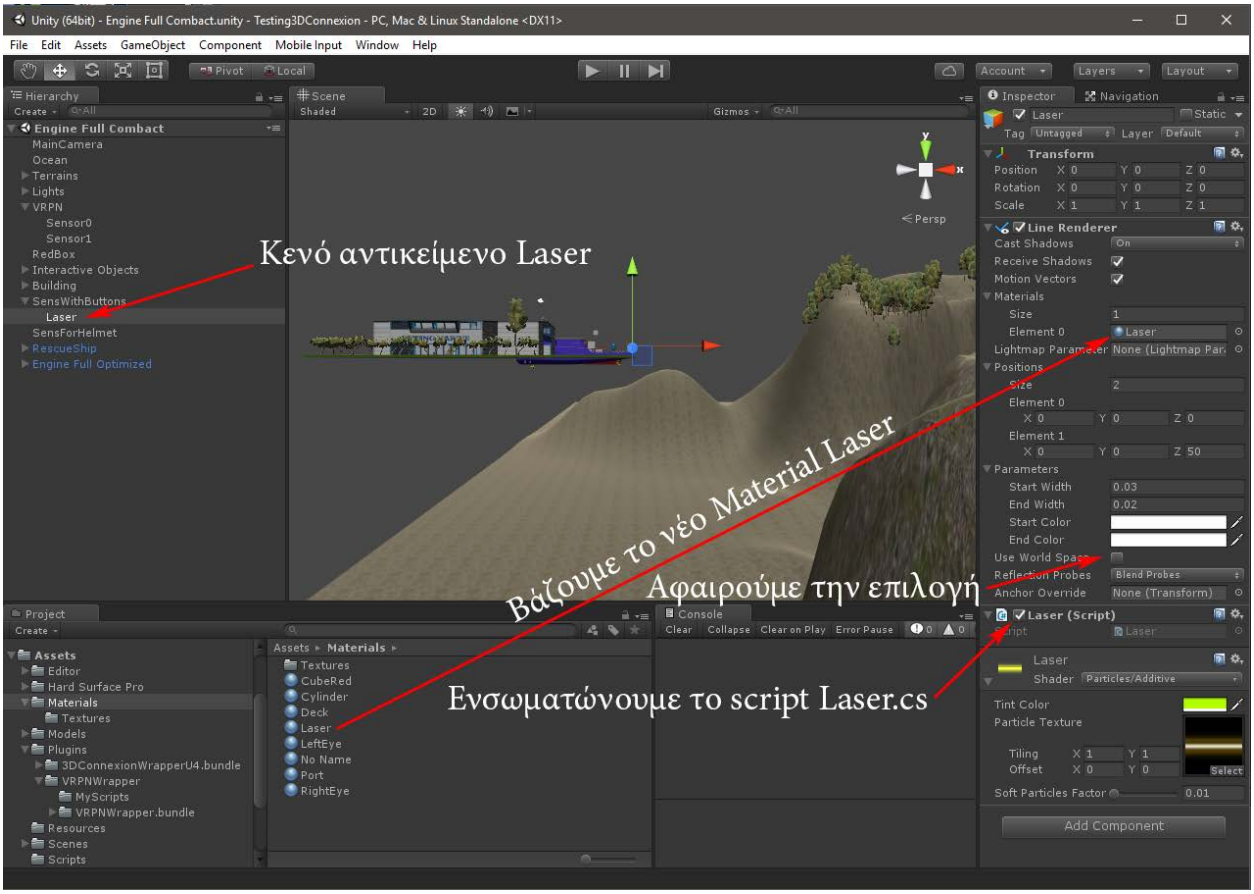
```
        }

        //this is for position
        PositionScript[] objectsPositionScript = Object.FindObjectsOfType(typeof(PositionScript)) as Posit
ionScript[];
        foreach (PositionScript obj in objectsPositionScript)
        {
            switch (obj.tag)
            {
                case "Interactive":
                    myPosition += obj.position;
                    break;
                case "VRPNTrackSensor0":
                    posSensor0 += obj.positionSensor;
                    break;
                case "VRPNTrackSensor1":
                    posSensor1 += obj.positionSensor;
                    break;
                default:
                    Debug.Log("Please add a tag to your object");
                    untaggedObjectExists = true;
                    break;
            }
        }
    }
```

6.4.Laser.cs

Θέλουμε να κατασκευάσουμε μια ακτίνα, η οποία πρέπει να είναι ορατή κατά την εκκίνηση της εφαρμογή μας, με την βοήθεια της οποίας θα έχουμε την δυνατότητα να αλληλοεπιδρούμε με διάφορα αντικείμενα (GameObjects) και να κάνουμε κάτι με αυτά, όπως να αλλάξουμε την θέση και τον προσανατολισμό τους. Η ακτίνα αυτή θέλουμε να προεκτείνεται από το αντικείμενο SensWithButtons προς τον εικονικό μας κόσμο σε μία εύλογη απόσταση οπότε για την δημιουργία της θα ακολουθήσουμε τα παρακάτω βήματα.

Κάνουμε δεξί κλικ στο Hierarchy window επιλέγουμε Create Empty και δίνουμε το όνομα Laser. Με drag and drop τοποθετούμε το κενό αυτό αντικείμενο Laser στο αντικείμενο SensWithButtons ώστε το πρώτο να είναι παιδί (child) του δεύτερου. Προσοχή πρέπει να δοθεί στα βήματα και το τρόπο που επιλέγονται τα αντικείμενα και πώς αυτά δημιουργούνται, στην συγκεκριμένη περίπτωση έχω επιλέξει ένα κενό αντικείμενο το οποίο θα χρησιμοποιήσω για το λέιζερ διότι δεν πρέπει να υπάρχουν στο αντικείμενο αυτό κανενός είδους Component όπως είναι το Box Collider ή Mesh Collider.

Στην συνέχεια φτιάχνουμε ένα C# script με την ονομασία **Laser.cs** και το ενσωματώνουμε στο άδειο αντικείμενο Laser, στο οποίο έχουμε βάλει και το συστατικό (Component) Line Renderer πατώντας το κουμπί Add Component -> Effects. Το επόμενο βήμα είναι να φτιάξουμε ένα νέο Material, έτσι κάνουμε δεξί κλικ στο Project windows και επιλέγουμε Create -> Material προσέχοντας να του δηλώσουμε τον τύπο Particle/Additive. Εφόσον έχουμε δημιουργήσει το νέο υλικό (Material) επιλέγουμε ένα χρώμα της αρεσκείας μας και εισάγουμε μια οποιαδήποτε εικόνα που φαντάζει ότι μπορεί να έχει μια ακτίνα λέιζερ και να είναι ορατή. Το νέο αυτό υλικό (Material) το τοποθετούμε στο Line Renderer -> Materials -> Element0 κάνοντας το drag and drop. Προσοχή στο Line Renderer Component αν θέλουμε να αλλάζει θέση και προσανατολισμό σύμφωνα με την κίνηση του αισθητήρα του συστήματος FOB θα πρέπει να αφαιρέσουμε την επιλογή Use World Space.

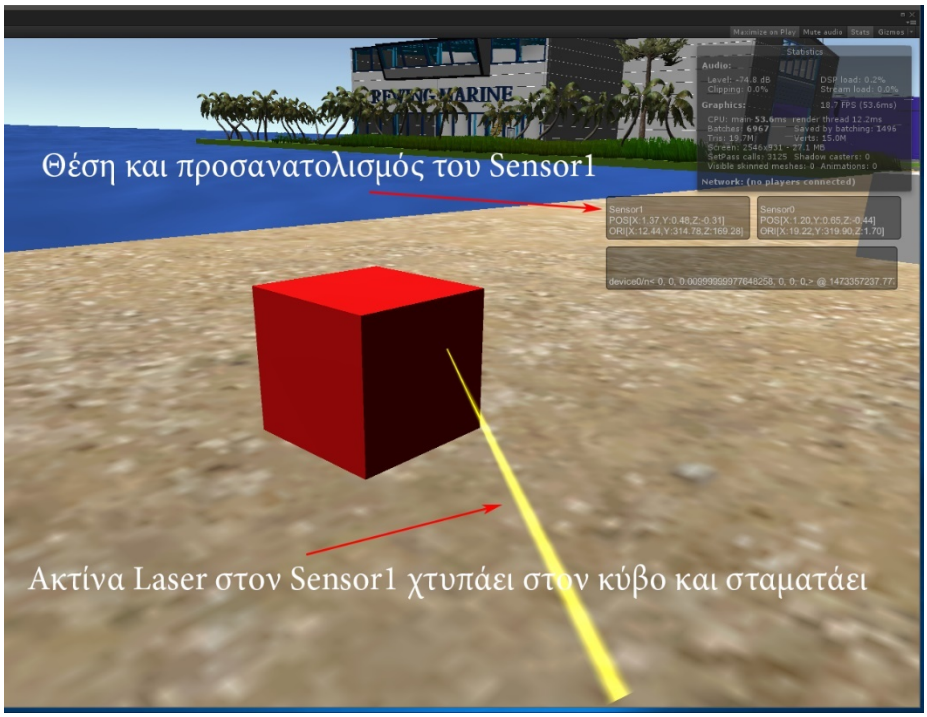


Στην συνέχεια ανοίγουμε το C# script με την ονομασία **Laser.cs** με το Visual Studio και γράφουμε τον κώδικα. Στην αρχή δηλώνουμε μια ιδιωτική μεταβλητή με όνομα line τύπου LineRenderer ενώ στην μέθοδο void Start(){} εκχωρούμε όλα τα συστατικά του LineRenderer σε αυτή την μεταβλητή line = GetComponent<LineRenderer>();. Στην μέθοδο void FixedUpdate(){} δηλώνουμε άλλη μια μεταβλητή με όνομα hit η οποία είναι τύπου RaycastHit και στη συνθήκη if καλούμε μια μέθοδο του Unity3D με ονομασία Physics.Raycast. Σαν είσοδο αυτή η μέθοδος έχει τα, transform.position - είναι η θέση που ξεκινάει το λέιζερ, transform.forward - είναι η διεύθυνση που θα ταξιδέψει και out hit - περιέχει την πληροφορία σχετικά με τον συγκρουστή (Collider), συγκρουστές ορίζουμε τα αντικείμενα που μπορούν να αλληλοεπιδράσουν σαν φυσικές οντότητες και να συγκρουστούν μεταξύ τους.

Με την συνθήκη if ελέγχουμε αν έχουμε χτυπήσει κάτι το οποίο έχει την ιδιότητα του συγκρουστή (Collider):

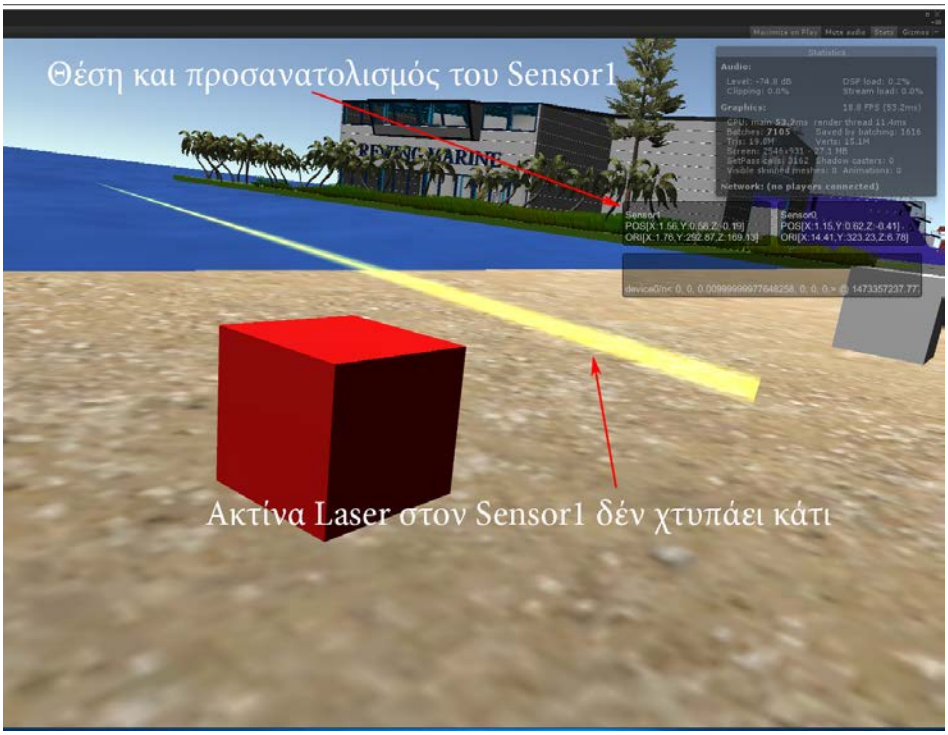
- Αν ναι τότε η ακτίνα λέιζερ σταματάει πάνω στο αντικείμενο με την βοήθεια του hit.distance.

```
if (hit.collider)
            {
                line.SetPosition(1, new Vector3(0, 0, hit.distance));
            }
```



- Αν όχι θα ταξιδεύσει μέχρι τις 200 μονάδες μέτρησης του Unity3D.

```
else
            {
                line.SetPosition(1, new Vector3(0, 0, 200));
            }
```



Η τελική γραφή του κώδικα είναι η παρακάτω.

```
using UnityEngine;
using System.Collections;

[RequireComponent(typeof(LineRenderer))]
public class Laser : MonoBehaviour
{
    private LineRenderer line;
    // Use this for initialization
    void Start()
    {
        line = GetComponent<LineRenderer>();
    }

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

    }
    void FixedUpdate()
    {

        RaycastHit hit;
        if (Physics.Raycast(transform.position, transform.forward, out hit))
        {
            if (hit.collider)
            {
                line.SetPosition(1, new Vector3(0, 0, hit.distance));
            }
            else
            {
                line.SetPosition(1, new Vector3(0, 0, 200));
            }
        }
    }
}
```

6.5.RayCastSensor.cs

Έχουμε φτιάξει την ακτίνα λέιζερ και έχουμε ρυθμίσει να μας δίνει οπτικά μια εικόνα με την οποία ελέγχουμε την κίνησή μας. Σε αυτό το σημείο θα δούμε πώς μπορούμε, με την βοήθεια μιας άλλης ακτίνας Ray η οποία έχει την ίδια διεύθυνση και φορά με την προηγούμενη ακτίνα, να δίνουμε κίνηση σε αντικείμενα τα οποία έχουμε δηλώσει να έχουν όνομα Tag “Interactive” και να περιέχουν συστατικά (Components) με την ονομασία BoxCollider και RigidBody.

Στην διαδρομή Assets\Plugins\VRPNWrapper\MyScripts μέσα από το Project window του Unity3D κάνουμε δεξί κλικ και δημιουργούμε άλλο ένα C# script με την ονομασία **RayCastSensor.cs** και το ενσωματώνουμε στο άδειο αντικείμενο SensWithButtons κάνοντάς το drag & drop. Χρησιμοποιούμε το αντικείμενο SensWithButtons γιατί σε αυτό έχουμε αντιστοιχίσει τον αισθητήρα του συστήματος FOB με το ενσωματωμένο ποντίκι με τα κουμπιά. Ανοίγουμε αυτό το νέο αρχείο με το Visual Studio και προσθέτουμε τις απαραίτητες γραμμές κώδικα.

Στην αρχή εισάγουμε δημόσιες μεταβλητές οι οποίες είναι οι :

- RayDepth, ρυθμίζει το βάθος της ακτίνας μέσα στον εικονικό κόσμο.
- ZoomObject, βήμα για να μεταβάλλουμε την απόσταση, από τον εαυτό μας, των αντικειμένων που έχουμε πιάσει με τον αισθητήρα.
- interactive, sensorWithButtons, objectInteractive, δηλώνουμε τα ενεργά αντικείμενα (GameObject).
- GrabWithButton, objIsGrabed, είναι μεταβλητές Boolean (true - false) χρήσιμες για να πιάνουμε τα αντικείμενα κάνοντας κλικ με το ποντίκι μας.
- numberOfClicks, είναι ο αριθμός των κλικ που κάνουμε με το μεσαίο πλήκτρο, καθώς κρατάμε στο δεξί μας χέρι τον αισθητήρα με το ενσωματωμένο ποντίκι, διαχωρίζοντας έτσι την επιλογή για το αν θέλουμε να πιάσουμε ή να αφήσουμε ένα αντικείμενο (GameObject).
- Μεταβλητές τις οποίες χρησιμοποιούμε στην μέθοδο public void OnClick(){} με την βοήθεια της οποίας κάνοντας κλικ με το δεξιό πλήκτρο μία ή δύο φορές απομακρύνουμε ή φέρνουμε κοντά μας αντικείμενα που έχουμε πιάσει.
    - mouseClicksStarted = false;
    - mouseClicks = 0;
    - mouseTimerLimit = .25f;
- Xtrans, Ytrans, Ztrans, είναι μεταβλητές που χρησιμοποιούμε για το σύστημα μετακίνησης (NavigationSystem) του εικονικού μας εαυτού μέσα στον εικονικό κόσμο (Scene) της εφαρμογής – παιχνιδιού.

```
//Change the depth of the ray cast
    public float RayDepth = 500f;
    //Double click & zoom in-out grabbed objects
    public float ZoomObject = 3f;
    public bool mouseClicksStarted = false;
    public int mouseClicks = 0;
    public float mouseTimerLimit = .25f;
    //Insert some important game objects
    public GameObject interactive, sensorWithButtons, objectInteractive;
    public bool GrabWithButton, objIsGrabed = false;
    public int numberOfClicks = 0;
    public int numberOfClicksButton0 = -1;
    //Public variables for navigation system
    public float Xtrans = 0f;
    public float Ytrans = 0f;
    public float Ztrans = 0f;
```

Στην μέθοδο void Start(){} δηλώνουμε με την βοήθεια του GameObject.FindWithTag ότι το αντικείμενο sensorWithButtons είναι αυτό με την ονομασία (Tag) "VRPNTrackSensor1" δηλαδή ο αισθητήρας με το ενσωματωμένο ποντίκι. Στην μέθοδο void Update(){} με διάφορες συνθήκες if statements εκτελούμε ξεχωριστές ενέργειες όπου για κάθε ενέργεια έχουμε δημιουργήσει ξεχωριστές μεθόδους τις οποίες καλούμε όταν χρησιμοποιηθεί κάποιο από τα τρία πλήκτρα του ποντικιού.

Με την πρώτη συνθήκη πατώντας το δεξί πλήκτρο (GetMouseButtonDown(0)) του ποντικού καλούμε την μέθοδο OnClick(); η οποία μας δίνει την δυνατότητα με ένα κλικ να απομακρύνουμε αντικείμενα που έχουμε πιάσει ενώ με δύο κλικ να τα φέρουμε κοντά μας.

```
if (Input.GetMouseButtonDown(0))
        {
            OnClick();
        }
```

Στην μέθοδο OnClick(); λοιπόν αποφασίζουμε αν έχουμε ένα ή δυο κλικ από το ίδιο πλήκτρο μέσω της χρονικής υστέρησης μεταξύ τους. Δηλαδή, αν ο χρόνος μεταξύ των κλικ είναι μικρότερος της τιμής mouseTimerLimit = .25f που έχουμε δηλώσει σαν μεταβλητή έχουμε δύο κλικ και τρέχουμε την εντολή ZoomObject = ZoomObject + 0.1f; ώστε να φέρουμε κοντά μας το αντικείμενο. Σε κάθε άλλη περίπτωση (else) έχουμε ένα κλικ οπότε εκτελούμε την εντολή ZoomObject = ZoomObject - 0.1f; απομακρύνοντας έτσι το αντικείμενο.

```
public void OnClick()
    {
        mouseClicks++;
        if (mouseClicksStarted)
        {
            return;
        }
        mouseClicksStarted = true;
        Invoke("checkMouseDoubleClick", mouseTimerLimit);
    }

    private void checkMouseDoubleClick()
    {
        if (mouseClicks > 1)
        {
            ZoomObject = ZoomObject - 0.1f;
            Debug.Log("Double Click");
        }
        else
        {
            ZoomObject = ZoomObject + 0.1f;
            Debug.Log("Single Click");
        }
        mouseClicksStarted = false;
        mouseClicks = 0;
    }
```

Με την δεύτερη συνθήκη if πιάνουμε και αφήνουμε αντικείμενα με το πάτημα του μεσαίου κουμπιού GetMouseButtonDown(2)). Αν ο αριθμός των κλικ είναι ίσος με ένα (numberOfClicks == 1) εκχωρούμε στη μεταβλητή GrabWithButton την τιμή αληθής (true), ενώ στη συνέχεια γίνεται έλεγχος της τιμής της μεταβλητής μέσω της συνθήκης if (GrabWithButton) και καλείται η μέθοδος void Carry(){} όταν η συνθήκη είναι αληθής. Σε άλλη περίπτωση (else) καλούμε την μέθοδο void Drop(){} ώστε να αφήσουμε το αντικείμενο.

```
if (Input.GetMouseButtonDown(2))
        {
            numberOfClicks++;
            if (numberOfClicks == 1) GrabWithButton = true;
            else
            {
                if (objectInteractive != null) Drop(objectInteractive);
                numberOfClicks = 0;
                GrabWithButton = false;
            }
        }
```

Έχουμε δηλώσει μια τοπική μεταβλητή με όνομα hitInfo τύπου RaycastHit και με την βοήθεια της συνθήκης if εντοπίζουμε όλα τα αντικείμενα (GameObjects) που βρίσκονται στην σκηνή με όνομα Tag "Interactive" και καλούμε για αυτά την μέθοδο void Carry(){}.

```
if (!objIsGrabed) objectInteractive = hitInfo.collider.gameObject;
            if (GrabWithButton)
            {
                if (objectInteractive.tag == "Interactive")
                {
                    Carry(objectInteractive);
                }
            }
```

Η μέθοδος void Carry(){} σαν είσοδο (input) έχει την μεταβλητή objectInteractive η οποία είναι τύπου GameObject. Αν τα στοιχεία της μεταβλητής αυτής περιέχουν (=true) την ιδιότητα (Component) isKinematic και έχουν ονομασία Tag = " Interactive " τότε προσδίδουμε σε αυτά την θέση (sensorWithButtons.transform.position + sensorWithButtons.transform.forward;) και τον προσανατολισμό (sensorWithButtons.transform.rotation;) του αισθητήρα με το ενσωματωμένο ποντίκι, έτσι το αντικείμενο ~~θα~~ ακολουθεί την κίνηση του αισθητήρα δίνοντάς μας την ψευδαίσθηση ότι το έχουμε πιάσει.

```
void Carry(GameObject objectInteractive)
    {
        objIsGrabed = true;
        objectInteractive.GetComponent<Rigidbody>().isKinematic = true;
        //position the objects i have graped to the sensors position
        objectInteractive.transform.position = sensorWithButtons.transform.position + sensorWithButtons.tr
ansform.forward;
        objectInteractive.transform.rotation = sensorWithButtons.transform.rotation;
        //print this to console
        Debug.Log("You picked up an object");
     }
```

Με την τελευταία εντολή if, η οποία σαν συνθήκη έχει αν πατήθηκε το αριστερό κουμπί (Input.GetMouseButtonDown(1)), προσθέτουμε τις συντεταγμένες της θέσης που χτυπάμε με την ακτίνα λέιζερ στις συντεταγμένες που έχει κάθε φορά ο εικονικός μας εαυτός (Avatar) μέσα στην σκηνή. Με αυτό τον τρόπο μπορούμε να μετακινούμαστε στον εικονικό κόσμο χωρίς να είναι απαραίτητη η φυσική μετακίνηση του σώματός μας στον πραγματικό κόσμο.

Συγκεκριμένα, από το hitInfo παίρνουμε της συντεταγμένες X,Y,Z του απομακρυσμένου άκρου της ακτίνας:

- hitInfo.point.x
- hitInfo.point.y
- hitInfo.point.z

και τις προσθέτουμε στις συντεταγμένες των αισθητήρων οι οποίοι καθορίζουν την θέση του εικονικού μας εαυτού.

```
//store temporary informations about the object we hit && Casting the Ray from my direction(transform.posi
tion) into the scene forward(transform.forward)
        RaycastHit hitInfo;
        Debug.DrawRay(transform.position, transform.forward * RayDepth, Color.black);
        if (Physics.Raycast(transform.position, transform.forward, out hitInfo))
        {
            //store the object temporary to avoid grabbing other objects intersecting the Ray
            if (!objIsGrabed) objectInteractive = hitInfo.collider.gameObject;
            if (GrabWithButton)
            {
                if (objectInteractive.tag == "Interactive")
                {
                    Carry(objectInteractive);
                }
            }
            Debug.Log("you are hitting something");

            //Navigation System
            if (Input.GetMouseButtonDown(1))
            {
                Xtrans = hitInfo.point.x;
                Ytrans = hitInfo.point.y;
                Ztrans = hitInfo.point.z;
                Debug.Log(hitInfo.point);
            }
        }
```

6.6.PositionScript.cs

Το PositionScript.cs είναι άλλο ένα script σε γλώσσα C# στο οποίο καθορίζουμε την θέση που θα έχουν τα αντικείμενα (GameObjects) όταν αυτά δέχονται δεδομένα από το 3D Connexion SpaceNavigator και το σύστημα αισθητήρων FOB.

Όσον αφορά τα αντικείμενα που δέχονται δεδομένα από το ποντίκι 3D Connexion SpaceNavigator, δημιουργούμε την μέθοδο internal void position(Vector3 pos){} η οποία όπως βλέπουμε έχει σαν είσοδο τη μεταβλητή pos τύπου Vector3, η οποία λαμβάνει δεδομένα από το αντίστοιχο delegate που βρίσκεται στο **GameManager.cs** όπου με την σειρά του δέχεται τις πληροφορίες από το script **VRPNAnalogSpaceNavigator.cs**. Τα δεδομένα της pos προστίθενται στην ήδη υπάρχουσα θέση των αντικειμένων transform.localPosition += pos * posSpeed; πολλαπλασιαζόμενα με την μεταβλητή public float posSpeed = _f; ώστε να έχουμε τον έλεγχο της δραστικότητας στην κίνησή τους.

Αντίστοιχα για το σύστημα FOB δημιουργούμε την μέθοδο public void positionSensor(Vector3 pos) η οποία και αυτή έχει είσοδο άλλη μια μεταβλητή ονόματος pos τύπου Vector3 η οποία ομοίως προέρχεται από το delegate του **GameManager.cs** και λαμβάνει πληροφορίες αυτή τη φορά μέσω του script **VRPNTracker.cs**. Στην ήδη υπάρχουσα θέση των αντικειμένων transform.localPosition προσθέτουμε τα νέα δεδομένα εφόσον τα έχουμε διαχωρίσει σύμφωνα με τους άξονες X,Y,Z = new Vector3(pos.x, pos.y + AvatarHeight, pos.z); ενώ η μεταβλητή AvatarHeight είναι ένας προσθετικός παράγοντας που καθορίζει το ύψος του Avatar, δηλαδή του εικονικού μας εαυτού.

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

public class PositionScript : MonoBehaviour
{
    public bool stopOnButton = false;
    public float posSpeed = 0.02f;
    //this is for the avatar height
    public float AvatarHeight = 0.3f;
    // Use this for initialization
    void Start()
    {

    }

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

    }
    //this is for 3D SpaceNavigator
    internal void position(Vector3 pos)
    {
        throw new NotImplementedException();
    }
    public void position(Vector3 pos, bool buttPressed)
    {
        transform.localPosition += pos * posSpeed;
    }

    //this is for VRPNTracker
    public void positionSensor(Vector3 pos)
    {
        transform.localPosition = new Vector3(pos.x, pos.y + AvatarHeight, pos.z);
        //transform.localPosition = new Vector3(pos.x + Xtrans, pos.y * AvatarHeight, pos.z + Ztrans);
    }
}
```

6.7.RotateScript.cs

Όσον αφορά το script RotateScript.cs διαχωρίζουμε και εδώ τα δεδομένα που αφορούν τις περιστροφές περί των συστημάτων αναφοράς που στέλνουν τα 3D Connexion SpaceNavigator και το σύστημα αισθητήρων FOB. Για το πρώτο χρησιμοποιούμε την μέθοδο public void rotate(Vector3 rot, bool buttPressed) η οποία έχει είσοδο τη μεταβλητή rot τύπου Vector3 την οποία δημιουργούμε στο αντίστοιχο delegate του **GameManager.cs** και δέχεται πληροφορίες από το script **VRPNAnalogSpaceNavigator.cs**. Για το δεύτερο, δηλαδή το σύστημα αισθητήρων FOB, έχουμε την μέθοδο public void rotationSensor(Quaternion rot) η οποία σαν είσοδο έχει τη μεταβλητή rot η οποία όμως είναι τύπου Quaternion όπου με την σειρά της η rot προέρχεται από το delegate του script **GameManager.cs** και λαμβάνει δεδομένα από το script **VRPNTracker.cs**.

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

public class RotateScript : MonoBehaviour
{
    public bool stopOnButton = false;
    public float rotationSpeed = 5f;
    public float rotSpeedTracker = 0.2f;
    // Use this for initialization
    void Start()
    {

    }

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

    }
    //this is for 3D SpaceNavigator
    internal void rotate(Vector3 rot)
    {
        throw new NotImplementedException();
    }

    public void rotate(Vector3 rot, bool buttPressed)
    {
        if (!stopOnButton)
            transform.localRotation *= Quaternion.Euler(rot * rotationSpeed);
        else if (stopOnButton && !buttPressed)
            transform.localRotation *= Quaternion.Euler(rot * rotationSpeed);
    }
    //this is for VRPNTracker
    public void rotationSensor(Quaternion rot)
    {
        transform.localRotation = rot;
    }
}
```

7. Παρατηρήσεις

Ένα σύστημα το οποίο αποτελείται από ηλεκτρονικούς υπολογιστές και περιφερειακές συσκευές διατάσσοντας ένα ενιαίο περιβάλλον εικονικής πραγματικότητας μπορεί να είναι απλό όμως τις περισσότερες φορές, λόγω του αριθμού των συσκευών και της χρήσης τους, μπορεί να αποτελέσει μια ιδιαίτερα περίπλοκη πρόκληση. Το εργαστήριο της σχολής Ναυπηγών Μηχανολόγων Μηχανικών VR-LAB διαθέτει σύγχρονα υπολογιστικά συστήματα καθώς και συσκευές εικονικής πραγματικότητας με την βοήθεια των οποίων είμαστε σε θέση να δημιουργήσουμε ένα ρεαλιστικό εικονικό περιβάλλον χωρίς να υστερεί ιδιαίτερα από πιο σύγχρονες τεχνολογικές διατάξεις του εμπορίου.

Το τρισδιάστατο ποντίκι της εταιρίας www.3dconnexion.eu με ονομασία 3D Connexion Space Navigator είναι μια συσκευή η οποία εξυπηρετεί σχεδιαστές τρισδιάστατων γραφικών, συνεργάζεται με προγράμματα όπως τα Autodesk Inventor, Solidworks, Autodesk Maya και 3D StudioMax, και δεν προοριζόταν κατασκευαστικά και λειτουργικά ως συσκευή εικονικής πραγματικότητας. Εμείς καταφέραμε, εκμεταλλευόμενοι τις δυνατότητες που μας δίνονται από το σύστημα VRPN, να συνδέσουμε την συσκευή αυτή με ένα περιβάλλον τρισδιάστατων γραφικών μέσω του Unity3D και να το χρησιμοποιήσουμε ως σύστημα αναφοράς για την μετακίνηση και περιστροφή αντικειμένων. Αυτό που έχει ιδιαίτερο ενδιαφέρον είναι ο τρόπος διασύνδεσης του 3D Connexion Space Navigator ο οποίος έγινε μέσω προγραμματισμού γράφοντας δηλαδή κώδικα (scripts) σε γλώσσα C# χωρίς να είναι αναγκαία η χρησιμοποίηση του προγράμματος οδηγού (drivers) που παρέχεται από την ίδια την εταιρία.

Από την άλλη μεριά το σύστημα της εταιρίας www.ascension-tech.com με την ονομασία Flock of Birds είναι εξ αρχής σχεδιασμένο και κατασκευασμένο να εξυπηρετεί διατάξεις εικονικής πραγματικότητας. Πρωτοεμφανιζόμενο το έτος 1992 μπορεί κάποιος να υποθέσει ότι αποτελεί πλέον μια ξεπερασμένη τεχνολογία και δεν θα έπρεπε να έχει θέση σε ένα σύγχρονο σύστημα Virtual Reality, όμως αυτό δεν είναι αληθές. Μειονεκτήματα που μπορούμε να υποθέσουμε ότι έχει το σύστημα FOB, σε σχέση με πιο σύγχρονες διατάξεις ανίχνευσης αισθητήρων, είναι ο όγκος που καταλαμβάνει στον χώρο, η κατανάλωση ρεύματος, οι περίπλοκες ρυθμίσεις και η συνδεσμολογία του και τέλος το γεγονός ότι κάθε αισθητήρας πρέπει να συνοδεύεται από έναν μικρό σε ισχύ υπολογιστή**.** Όμως η ακρίβεια των δεδομένων που λαμβάνουμε καθώς και το γεγονός ότι η ανίχνευση των αισθητήρων δεν επηρεάζεται από φυσικά εμπόδια καθιστά το μαγνητικό σύστημα ανίχνευσης FOB ιδιαίτερα χρήσιμο και αξιόπιστο.

Η καρδιά της πειραματικής διάταξης εικονικής πραγματικότητας που στήθηκε για την παρούσα διπλωματική εργασία είναι αναμφισβήτητα η συλλογή προγραμμάτων που αποτελούν το **VRPN**. Το αρκτικόλεξο **VRPN** αναλύεται σε **Virtual Reality Peripheral Network** ενώ σε ελληνική απόδοση **Περιφερειακό Δίκτυο Εικονικής Πραγματικότητας**. Είναι μια προσπάθεια ανοιχτού κώδικα του οποίου η βάση σχεδιάστηκε και υλοποιήθηκε από τον **Russell M. Taylor II** καθηγητή του πανεπιστημίου της Βόρειας Καρολίνα στο τμήμα Επιστήμης των Υπολογιστών (Computer Science) και ακολουθεί το πρωτόκολλο αδειών **Boost Software License 1.0 (BSL1.0)**. Σκοπός του είναι η ομαδοποίηση περιφερειακών συσκευών σε ένα περιβάλλον εικονικής πραγματικότητας, όσον αφορά τον τρόπο χρήσης και διαχείρισης, υπό τον έλεγχο ενός κεντρικού διαχειριστή (Server)**,** ενώ πρέπει να αναφέρουμε πως την περαιτέρω ανάπτυξη και συντήρηση του **VRPN** έχει αναλάβει η εταιρία http://sensics.com/.

Η επιτυχία του ανωτέρου συστήματος έγκειται στο γεγονός ότι μπορούμε να αντιμετωπίσουμε τις περιφερειακές συσκευές με παρόμοιο τρόπο είτε πρόκειται για ανιχνευτές, συσκευές με κουμπιά ή απλά συσκευές αναλογικού σήματος γράφοντας και παραμετροποιώντας κώδικα σε γλώσσα C#. Αυτή την στιγμή υποστηρίζονται εκατοντάδες συσκευές και πλέον αποτελεί τον βασικό μηχανισμό διαχείρισής τους σε εφαρμογές εικονικής πραγματικότητας βιομηχανικού επιπέδου.

Λογιζόμενοι το ενδιαφέρον που δείχνουν εταιρίες όπως η Microsoft HoloLens, Unity Technologies, Google, Facebook/Oculus VR, Samsung Gear VR, Nvidia, AMD και Valve το μέλλον της εικονικής πραγματικότητας φαντάζει ενθουσιώδες. Το 2015 κατασκευάστηκαν και διατέθηκαν στην ευρεία αγορά προϊόντα μαζικής κατανάλωσης για εφαρμογές ηλεκτρονικών παιχνιδιών, ενώ μεγάλο ενδιαφέρον έχει το γεγονός ότι ακόμα και τα κινητά τηλέφωνα σύγχρονης τεχνολογίας μπορούν να χρησιμοποιηθούν στην εικονική πραγματικότητα. Όσον αφορά την επιστήμη της ναυπηγικής και γενικότερα της μηχανολογίας ευχόμαστε στο άμεσο μέλλον να γίνει ενσωμάτωση της τεχνολογίας αυτής σε προγράμματα CAD/CAM ώστε ο εκάστοτε μηχανικός να έχει την δυνατότητα σχεδίασης νέων μηχανολογικών συστημάτων μέσα από ένα διαδραστικό τρισδιάστατο περιβάλλον.

8. Χρήσιμες ιστοσελίδες

8.1. https://github.com/vrpn/
8.2. https://sourceforge.net/p/vrpnwrapper/discussion/1087054/thread/020a5275/?limit=25#dfb5/aa75
8.3. https://grabcad.com/
8.4. https://www.3dcontentcentral.com/
8.5. https://forum.unity3d.com/threads/finally-an-interactive-tutorial-series-that-will-teach-you-c-for-unity3d.175410/
8.6. http://www.cs.uoi.gr/~loukas/courses/Data_Structures/index.files/Page355.htm
8.7. https://channel9.msdn.com/Series/C-Sharp-Fundamentals-Development-for-Absolute-Beginners
8.8. https://research.cc.gatech.edu/uart/content/vrpnwrapper
8.9. https://inmagicwetrust.wordpress.com/

9. Βιβλιογραφία

9.1. Unity 4.x Game AI Programming - Authors : Aung Sithu Kyaw, Clifford Peters
9.2. Unity AI Programming Essentials - Authors : Curtis Bennett, Dan Violet Sagmiller
9.3. Virtual Reality Technology - Authors : Grigore C. Burdea, Philippe Coiffet
9.4. Unity in Action – Author : Joe Hocking
9.5. Unity 4.x Game Development by Example Beginner's Guide – Authors : Ryan Henson Creighton
9.6. A Programmer's Guide to C# 5.0, 4th Edition – Authors : Gunnerson Eric, Wienholt Nick
9.7. C# Game Programming Cookbook for Unity 3D - Author : Jeff W. Murray
9.8. Learning C# by Developing Games with Unity 3D Beginner's Guide - Author : Terry Norton
9.9. Unity3D Documentation - Author : https://unity3d.com/