



Rapport de projet 2A

Tutoriel Projet

Interaction Hololens 2 avec un Bras Robot Simulé sous
ROS dans Unity3D

effectué par Jaheer Goulam

Enseignant encadrant :
Michel Chapron

TABLE DES MATIÈRES

| | |
|--|-----------|
| I. Introduction..... | 3 |
| - I.1 Aperçu du projet..... | 3 |
| - I.2 Objectifs et résultats attendus..... | 3 |
| II. Prérequis..... | 4 |
| - II.1 Connaissances nécessaires (Unity3D, ROS, Hololens 2)..... | 4 |
| - II.2 Matériel et logiciel requis..... | 4 |
| - II.3 Installation et configuration de l'environnement de développement..... | 5 |
| - II.3.1 Unity3D..... | 5 |
| - II.3.2 ROS dans une machine virtuelle..... | 6 |
| - II.3.3 Visual Studio 2019..... | 7 |
| - II.3.4 Hololens 2..... | 8 |
| III. Vue d'ensemble de l'Architecture du Projet..... | 9 |
| - III.1 Description de l'architecture ROS..... | 9 |
| - III.2 Intégration de Unity3D avec ROS..... | 9 |
| IV. Configuration de l'Environnement Unity3D..... | 10 |
| - IV.2 Partie Unity3D (environnement Hololens 2)..... | 10 |
| - IV.2.1 Mixed Reality Tool Kit (package)..... | 10 |
| - IV.1.3. Scène Unity..... | 10 |
| VI. Intégration avec ROS..... | 24 |
| - VI.1 Installation et Configuration de la communication entre Unity3D et ROS..... | 24 |
| - VI.4 Tests et validation de la simulation..... | 24 |
| VII. Déploiement sur Hololens 2..... | 25 |
| - VII.1 Préparation du projet Unity3D pour Hololens..... | 25 |
| Quand tous les paramètres seront corrects, appuyez sur build. Une fenêtre d'explorer s'ouvrira et vous devriez choisir un dossier dans lequel vous mettrez le build..... | 27 |
| - VII.2 Configuration de l'environnement de déploiement sur Visual Studio..... | 27 |
| - VII.3 Tests et dépannage sur Hololens 2..... | 28 |
| - IX.1 Problèmes courants et leurs solutions..... | 29 |
| X. Conclusion..... | 30 |
| - X.1 Récapitulatif des réalisations..... | 30 |
| - X.2 Limitations et perspectives d'amélioration..... | 30 |
| XI. Annexes..... | 30 |
| - XI.1 Références..... | 30 |

I. Introduction

- I.1 Aperçu du projet

Ce projet vise à fusionner les mondes de la simulation robotique et de la réalité augmentée, en utilisant Unity3D pour simuler un bras robotique effectuant des tâches de pick and place, ROS pour gérer la logique de contrôle et les communications du robot, et Hololens 2 pour projeter la simulation dans un espace physique. L'intégration de ces technologies avancées ouvre la voie à des applications innovantes dans l'enseignement, la recherche, et l'industrie, en offrant une plateforme immersive pour la conception et le test de systèmes robotiques.

- I.2 Objectifs et résultats attendus

Les objectifs de ce projet comprennent la démonstration de la capacité à simuler des interactions robotiques complexes dans un environnement virtuel, l'intégration fluide de cette simulation avec ROS pour un contrôle réaliste, et l'utilisation de Hololens 2 pour visualiser et interagir avec la simulation dans un espace physique. Les résultats attendus incluent la capacité à manipuler virtuellement des objets avec précision et à comprendre les applications potentielles de la réalité augmentée dans la robotique.

Un aspect crucial de ce projet a été l'adoption et l'adaptation d'un projet de base disponible sur GitHub, spécifiquement le Unity Robotics Hub [\[1\]](#) développé par Unity Technologies. Ce répertoire a servi de fondation solide pour notre travail, offrant des exemples de référence et des bibliothèques pour faciliter l'intégration entre Unity3D et ROS, ainsi qu'un point de départ pour la simulation robotique. Je referais donc le tutoriel qui a été proposé sur le GitHub mais comme n'importe quel tutoriel rien ne fonctionne au premier essai. Suite à l'adoption du projet initial disponible sur GitHub, notre objectif a évolué vers l'intégration de la simulation robotique dans un environnement immersif et interactif dédié au Hololens 2.

II. Prérequis

- II.1 Connaissances nécessaires (Unity3D, ROS, Hololens 2)

Les utilisateurs devraient avoir une compréhension de base de la programmation en C# pour Unity, une familiarité avec le système d'exploitation ROS, et une connaissance des principes de base de la réalité augmentée. Une expérience préalable avec les Hololens 2, bien que non essentielle, serait bénéfique.

Vous trouverez les sites officiels en annexes pour apprendre les bases Unity, l'environnement ROS et les fonctionnalités du Hololens 2.

- II.2 Matériel et logiciel requis

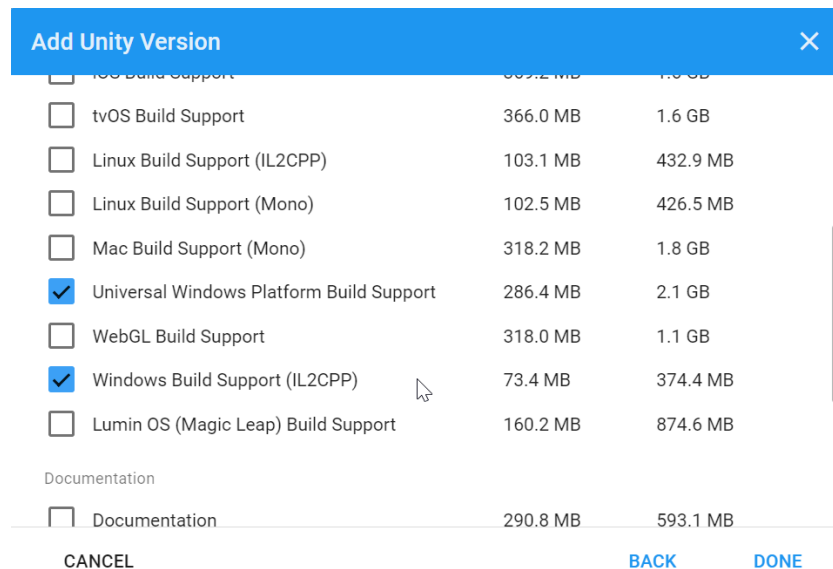
Pour ce projet vous auriez besoin de plusieurs logiciels avec les bonnes versions. J'ai testé plusieurs versions de chaque logiciel et je vais vous lister celles qui marchent pour notre projet. Vous aurez besoin :

- Unity3D 2020.3
- Visual Studio 2019
- Visual Studio code
- ROS melodic (dans une machine virtuelle sous Ubuntu 18.04)
- Hololens 2
- Windows 10 SDK (dernière version)
- Mixed Reality Toolkit (MRTK)
- PC Windows

- II.3 Installation et configuration de l'environnement de développement

- II.3.1 Unity3D

Il ne suffit pas seulement d'installer Unity3D 2020.3 avec la bonne version, il faut aussi ajouter le module Windows Universal Platform Build Support et le Windows Build Support



Le **Universal Windows Platform** (UWP) Build Support dans Unity est un module complémentaire qui permet aux développeurs de créer des applications et des jeux pour toutes les appareils fonctionnant sous Windows 10, y compris les PC, les tablettes, les smartphones, les Xbox, les HoloLens et plus encore, à partir d'un unique code source. UWP fait partie de l'écosystème Windows 10 et vise à aider les développeurs à créer des applications pouvant s'exécuter sur de multiples types d'appareils avec peu ou pas de modification nécessaire du code. Cela va donc servir à déployer notre application sur le Hololens 2. [\[2\]](#)

En parallèle de ceci, il faut aussi que vous installiez **Windows 10 SDK** (dernière version) [\[3\]](#) Le Windows Software Development Kit (SDK) est un ensemble d'outils de développement fournis par Microsoft qui permettent aux développeurs de créer des applications pour le système d'exploitation Windows. Il s'agit d'une suite essentielle pour le développement d'applications Windows, offrant des bibliothèques de programmation, des échantillons de code, des outils de compilation, et des interfaces de programmation d'applications (API) pour utiliser les fonctionnalités de Windows. Voici quelques utilisations clés du Windows SDK :

- Développement d'Applications Windows
- Accès aux API Windows
- Test et Débogage
- Développement pour la Réalité Mixte
- Intégration avec Visual Studio

Vous trouverez un lien pour débiter sur Unity3D, cela permettra de mieux comprendre la suite.

- II.3.2 ROS dans une machine virtuelle

Pour démarrer ce projet, il est nécessaire de mettre en place une machine virtuelle (VM). **Oracle VM VirtualBox** peut faire l'affaire. Avant de procéder à l'installation, assurez-vous que votre ordinateur est équipé d'une **carte Wi-Fi fonctionnelle**. Cette vérification est cruciale car, une fois le projet développé, le déploiement et la communication via Wi-Fi seront essentiels.



Nous opterons pour **Ubuntu 18.04** comme système d'exploitation au sein de la VM. Cette version est compatible avec ROS 1, qui est notre choix pour ce projet en raison de sa stabilité et de son large support communautaire. Bien que ROS 2 soit également une option viable, il est important de noter que la configuration différerait sensiblement.

Lors de la configuration de la VM, il est essentiel de l'établir avec une configuration de **réseau en mode pont** (Bridged Network). Cette configuration permet à la VM d'apparaître sur le réseau comme un dispositif indépendant, doté de sa propre adresse IP. Cela facilite les interactions réseau directes avec d'autres appareils sur le même réseau, ce qui est particulièrement important pour le déploiement et le fonctionnement de notre projet.

Pour vérifier que votre machine virtuelle dispose d'une adresse IP distincte, saisissez la commande suivante :

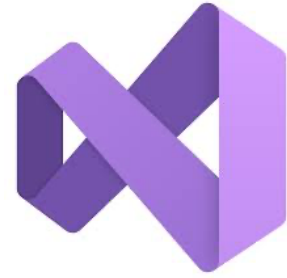
```
hostname -I
```

L'adresse IP affichée devrait commencer par 192.x.x.x, et non par 127.0.0.1, car cela indiquerait que vous utilisez une adresse locale.

Le document fourni en annexe détaille minutieusement les étapes requises pour l'installation de ROS ainsi que la configuration de l'environnement de travail (workspace) spécifique à notre projet. Il est fortement recommandé de consulter ce guide avant de débiter, afin de vous familiariser avec l'utilisation et les principes fondamentaux de ROS, comme exposé dans la section III.1. Cette préparation préalable est essentielle pour assurer une progression fluide et efficace dans le développement du projet.

- II.3.3 Visual Studio 2019

L'intégration de **Visual Studio 2019** constitue un pilier essentiel dans le développement de notre projet, offrant un environnement de développement intégré (EDI) riche en fonctionnalités pour la programmation, le débogage et la gestion de code. Visual Studio 2019 est choisi pour sa compatibilité étendue avec les langages de programmation C# et C++, essentiels pour le développement sous Unity3D et pour l'interaction avec ROS via des plugins ou des bibliothèques spécifiques.

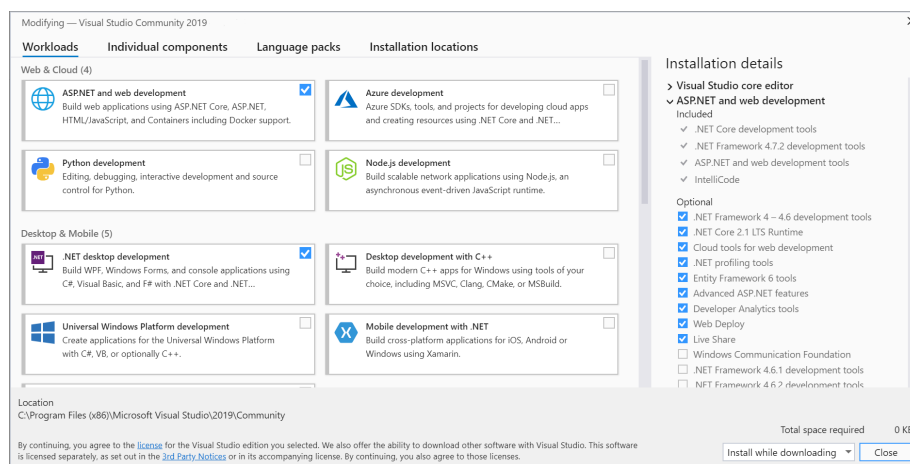


C'est à travers Visual Studio 2019 que nous procéderons au déploiement de notre application sur le casque Hololens 2. J'ai testé plusieurs versions de Visual Studio et celui là est le seul qui fonctionne avec notre projet sans erreur.

Installation de Visual Studio 2019 :

Vous auriez besoin d'installer les composants nécessaires pour notre projet, qui sont les suivants :

- Windows 10 SDK (la version que vous avez sur votre PC)
- USB Device Connectivity (facultatif, si vous ne souhaitez pas utiliser la Wi-Fi)
- C++ (v142) Universal Windows Platform tools (nécessaire avec Unity)



Une fois tous les composants sélectionnés, vous pouvez procéder à l'installation du logiciel.

- II.3.4 Hololens 2

Pour déployer notre application sur le casque Hololens 2 depuis un PC, il est nécessaire d'activer l'accès développeur sur le casque. Pour ce faire, naviguez dans les paramètres, sélectionnez "Mise à jour et sécurité", puis "Pour les développeurs". Activez ensuite le "Mode développeur".

Vous trouverez en annexe un lien pour bien débuter avec le Hololens 2, si cela est nécessaire [\[4\]](#)

III. Vue d'ensemble de l'Architecture du Projet

L'architecture de ce projet est conçue pour fusionner la simulation de robotique avancée avec les technologies de réalité augmentée, permettant ainsi une immersion et une interaction inédites dans le domaine de la simulation robotique. Au cœur de cette architecture se trouvent deux composants principaux : le système de contrôle robotique basé sur ROS (Robot Operating System) et l'environnement de développement Unity3D.

- III.1 Description de l'architecture ROS

ROS, ou Robot Operating System, sert de framework intermédiaire facilitant la communication et la gestion des processus entre les différents composants logiciels de notre robot simulé. L'architecture ROS est modulaire et distribuée, composée de nœuds qui communiquent entre eux via des topics, des services et des actions.



Dans notre projet, ROS est utilisé pour gérer les tâches de bas niveau, telles que le traitement des données sensorielles, la planification des mouvements et l'exécution des commandes de contrôle. Les nœuds ROS spécifiques sont développés pour chaque fonctionnalité requise, permettant une grande flexibilité et une modularité du système. Par exemple, un nœud peut être responsable de la simulation des capteurs du robot, tandis qu'un autre gère la logique de prise et de déplacement des objets.

Mon premier rapport explique en détail le fonctionnement de ROS dans notre projet. Je vous invite à le lire de même pour mieux comprendre la suite. Vous trouverez en annexe un lien pour en apprendre plus sur ROS [\[5\]](#)

- III.2 Intégration de Unity3D avec ROS

L'intégration de Unity3D avec ROS (Robot Operating System) constitue une étape cruciale de notre projet, fusionnant la simulation 3D immersive avec les capacités de contrôle robotique avancées. Cette synergie permet de créer des simulations interactives et réalistes de robots dans des environnements virtuels, tout en utilisant ROS pour gérer la logique de contrôle et la communication entre les composants du système.

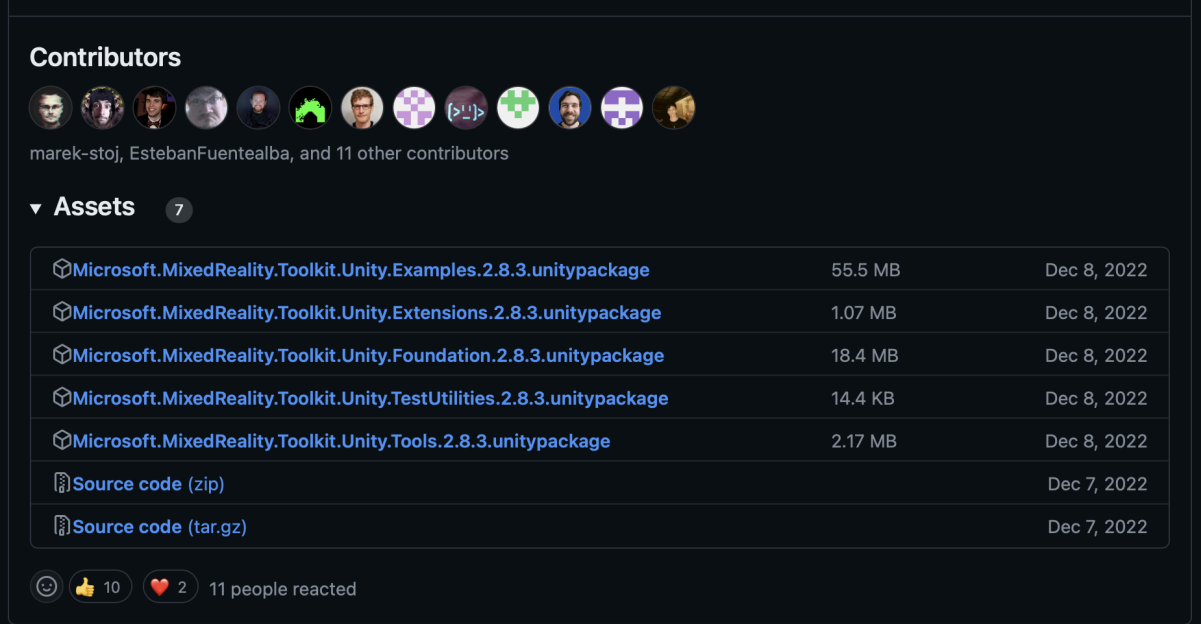
IV. Configuration de l'Environnement Unity3D

Cette section détaille la première partie du tutoriel Pick and Place [\[1\]](#) disponible sur GitHub, adaptée avec certaines modifications pour être compatible avec notre environnement Hololens 2 (vidéo tutoriel sur Youtube) [\[6\]](#)

- IV.2 Partie Unity3D (environnement Hololens 2)

- IV.2.1 Mixed Reality Tool Kit (package)

On aura besoin d'un package qui va nous permettre de créer une scène compatible avec notre environnement Hololens 2. Pour se faire, on va télécharger le Mixed Reality Tool Kit [\[7\]](#)



Contributors

marek-stoj, EstebanFuentealba, and 11 other contributors

▼ **Assets** 7

| | | |
|---|---------|-------------|
| Microsoft.MixedReality.Toolkit.Unity.Examples.2.8.3.unitypackage | 55.5 MB | Dec 8, 2022 |
| Microsoft.MixedReality.Toolkit.Unity.Extensions.2.8.3.unitypackage | 1.07 MB | Dec 8, 2022 |
| Microsoft.MixedReality.Toolkit.Unity.Foundation.2.8.3.unitypackage | 18.4 MB | Dec 8, 2022 |
| Microsoft.MixedReality.Toolkit.Unity.TestUtilities.2.8.3.unitypackage | 14.4 KB | Dec 8, 2022 |
| Microsoft.MixedReality.Toolkit.Unity.Tools.2.8.3.unitypackage | 2.17 MB | Dec 8, 2022 |
| Source code (zip) | | Dec 7, 2022 |
| Source code (tar.gz) | | Dec 7, 2022 |

10 2 11 people reacted

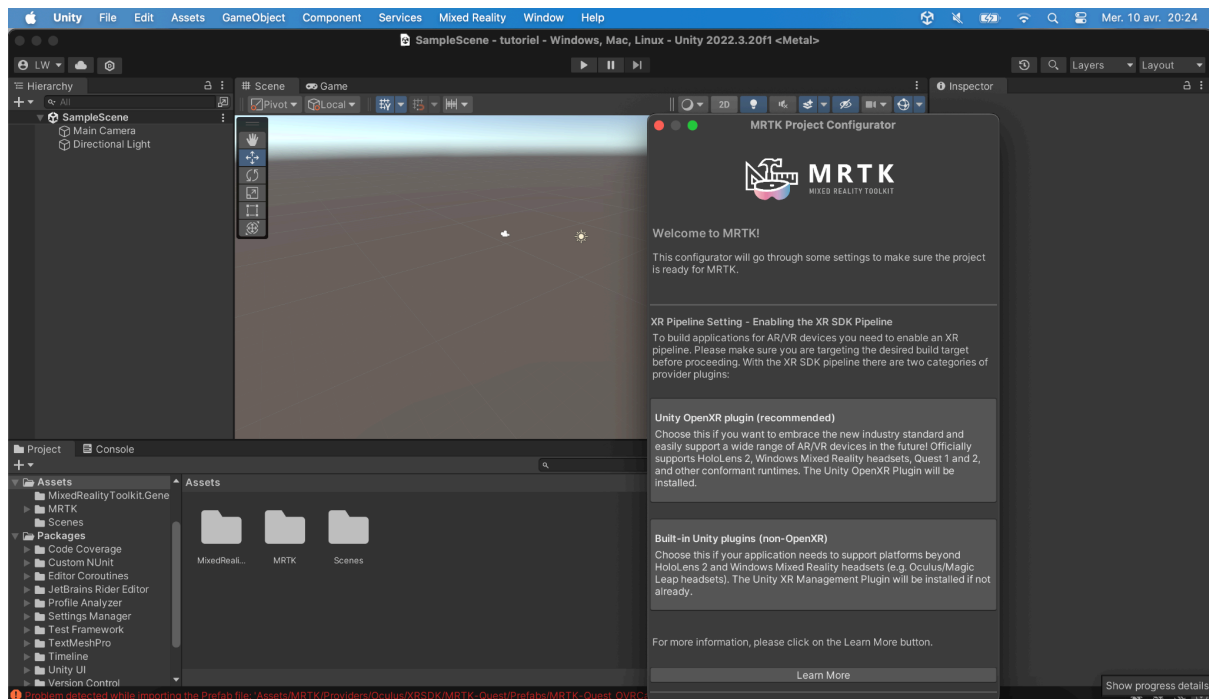
On choisira le Microsoft.MixedReality.Toolkit.Unity.Foundation.2.8.3.unitypackage

- IV.1.3. Scène Unity

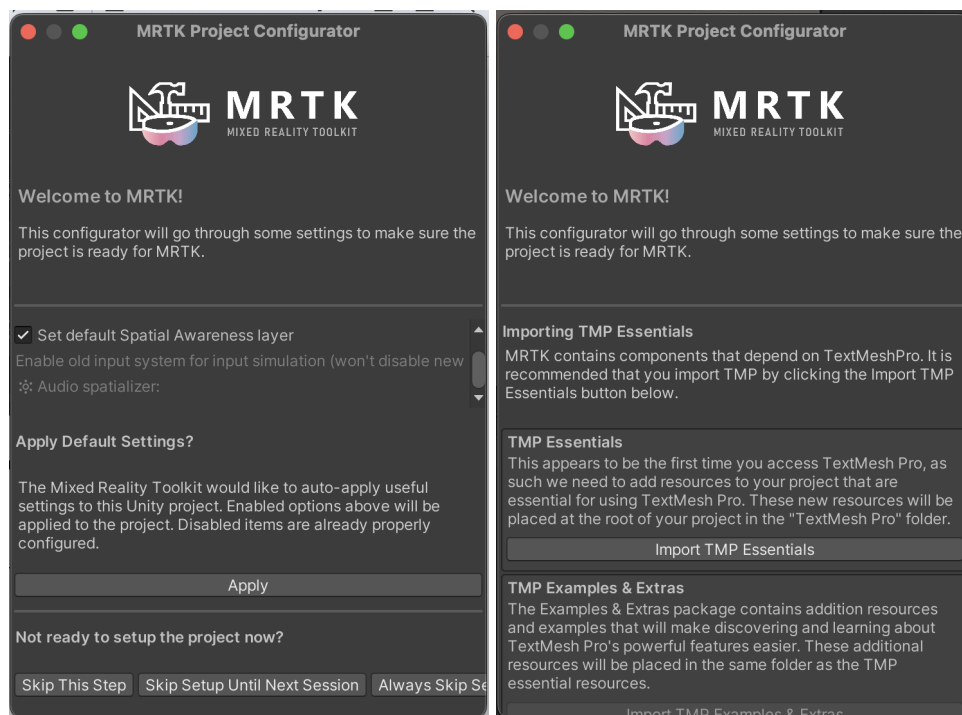
Cette partie sera essentiellement ce qui a été fait sur la vidéo Youtube [\[6\]](#) et le GitHub [\[1\]](#) en y apportant quelques modifications car j'ai rencontré des problèmes avec le tutoriel.

Après avoir créé un projet Unity3D vide, on commence par importer notre package Mixed reality Tool Kit installé précédemment.

Pour cela, vous pouvez tout simplement glisser votre package dans les ‘assets’ et cliquer sur ‘import’.

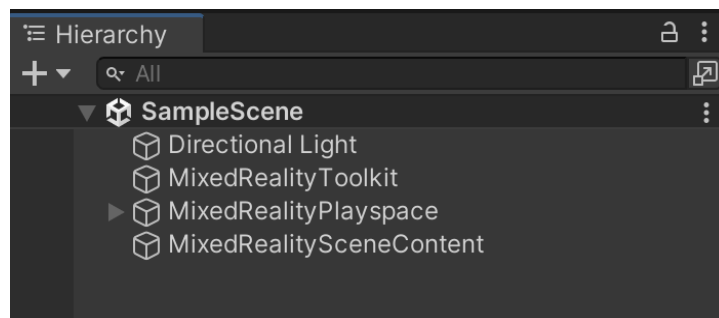


Après cela, une fenêtre va s’ouvrir et cliquer sur Unity OpenXR. Unity peut vous demander de rouvrir le projet, s’il ne le demande pas continuez la configuration et cliquez sur ‘Apply’ Puis importer le ‘TMP Essentials’.

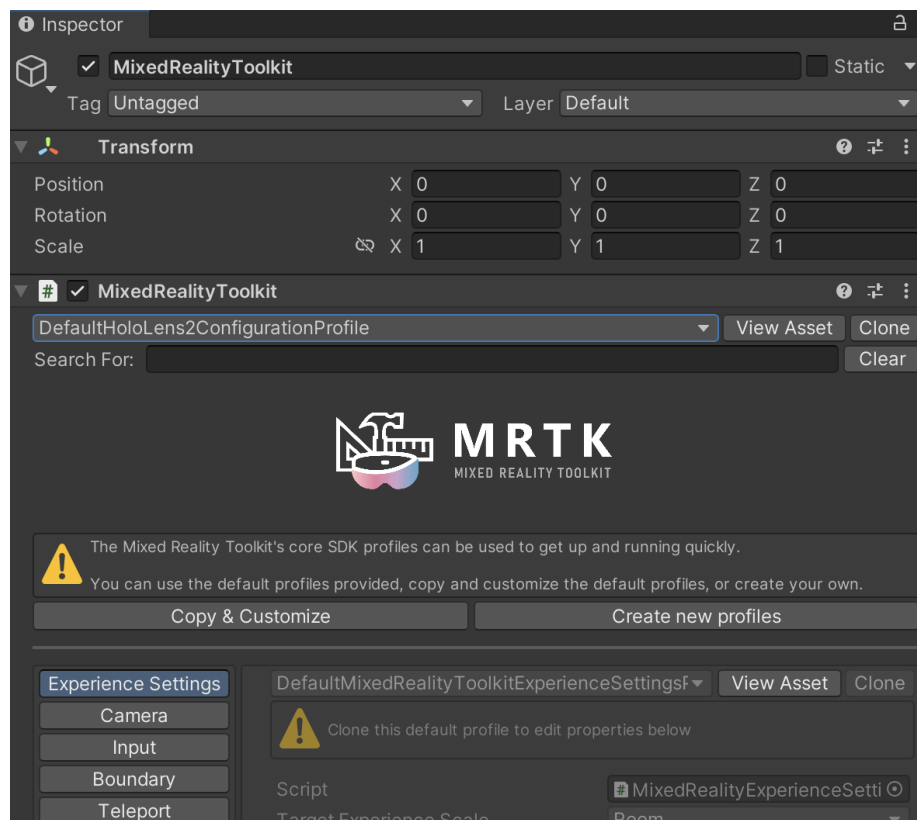


Quand vous aurez fini la configuration , vous devriez avoir ‘Mixed Reality’ sur la barre de menu.

Maintenant, nous allons configurer la scène pour que ça s’adapte à l’environnement Hololens 2. Pour ce faire, cliquez sur Mixed reality -> ToolKit -> Add to scene and configure.

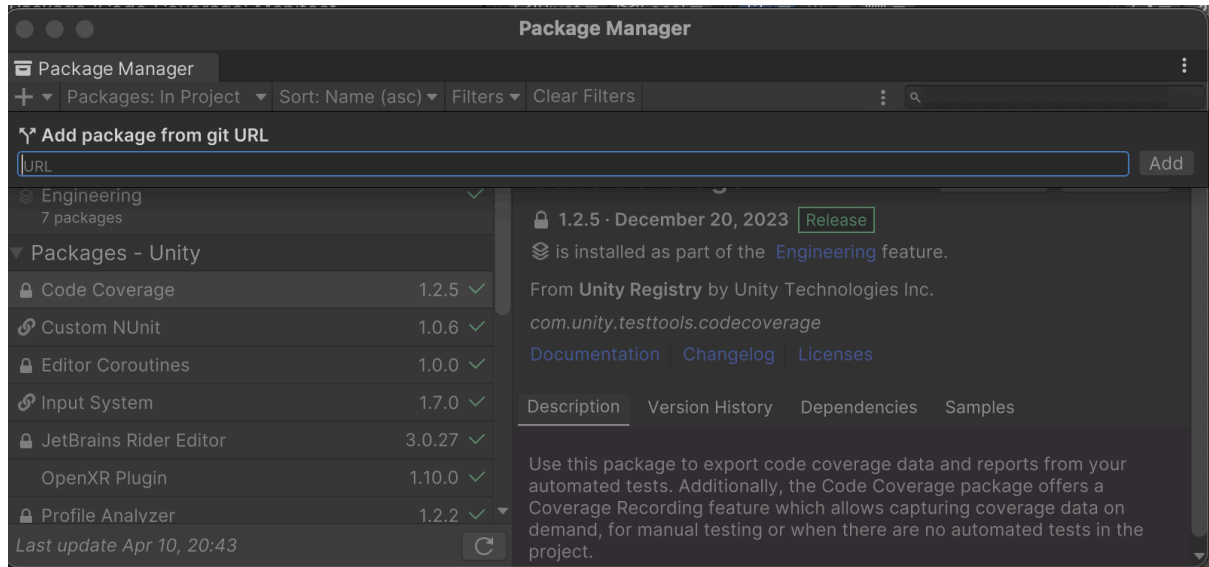


Vous verrez apparaître dans votre hiérarchie de nouveaux objets. Cliquez sur ‘MixedReality Toolkit’



Et changer la scène en ‘DefaultHololens2ConfigurationProfile’ puis appuyez sur ‘Clone’.

Maintenant on va importer le package ‘URDF Importer’ qu’on avait téléchargé précédemment. Pour cela, rendez-vous sur Windows -> Package Manager et cliquer sur ‘add package from git URL’



Rendez vous sur le GitHub, et cliquer sur ‘URDF Importer’

| Repo | Functionality |
|-----------------------------------|---|
| ROS TCP Endpoint | ROS node for sending/receiving messages from Unity |
| ROS TCP Connector | Unity package for sending, receiving, and visualizing messages from ROS |
| URDF Importer | Unity package for loading URDF files |

Vous trouverez normalement le lien pour le téléchargement du package :

<https://github.com/Unity-Technologies/URDF-Importer.git?path=/com.unity.robotics.urdf-importer#v0.5.2>

Copiez ce lien sur le package manager et importez- le. Vous verrez apparaître un nouveau dossier dans ‘Packages’ sur Unity.

On va faire de même avec un autre package, le ‘ROS TCP Connector’ sur le même GitHub.

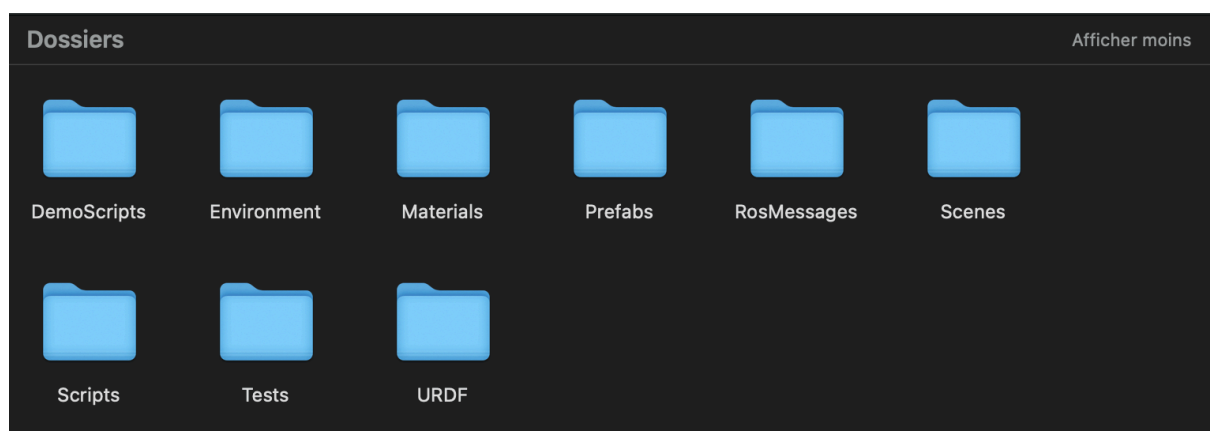
<https://github.com/Unity-Technologies/ROS-TCP-Connector.git?path=/com.unity.robotics.ros-tcp-connector>

Maintenant, on va importer les 'assets' du tutoriel sur Github. Pour commencer, on va cloner le projet sur votre pc avec la commande suivante :

```
git clone https://github.com/Unity-Technologies/Unity-Robotics-Hub.git
```

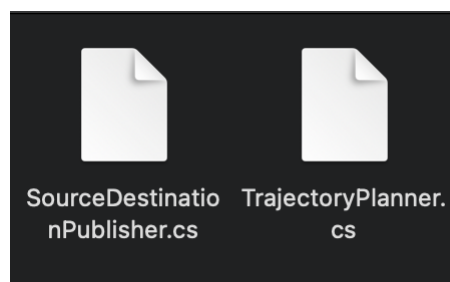
On va ensuite se rendre dans

Unity-Robotics-Hub/tutorials/pick_and_place/PickAndPlaceProject/Assets et glisser tous ces dossiers de ce répertoire dans votre 'Assets' dans Unity.

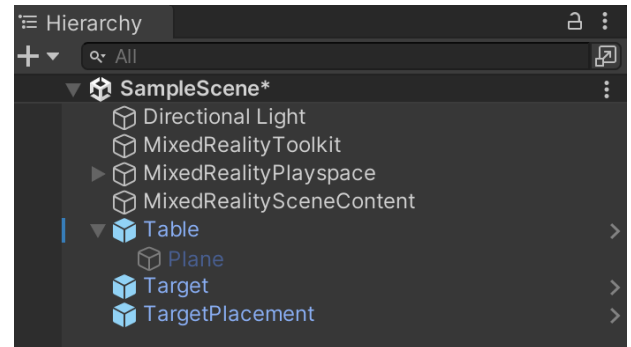


Maintenant rendez vous dans

/Unity-Robotics-Hub/tutorials/pick_and_place/Scripts et glisser les deux fichiers qu'il y a dedans dans Assets/DemoScripts



Maintenant, on va mettre tous nos objets sur la scène Unity, pour ce faire, rendez vous dans Assets/Prefabs et glisser les 3 éléments sur votre hiérarchie.



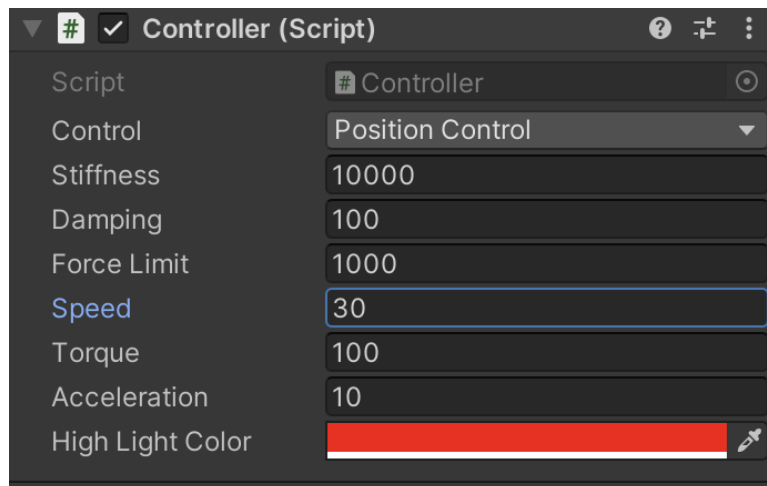
Pensez à désactiver le Plane

Maintenant, nous allons procéder à l'importation de notre robot. Précédemment, nous avons importé le package URDF Importer, qui va nous permettre de reconstituer le robot dans Unity à l'aide d'un fichier URDF.

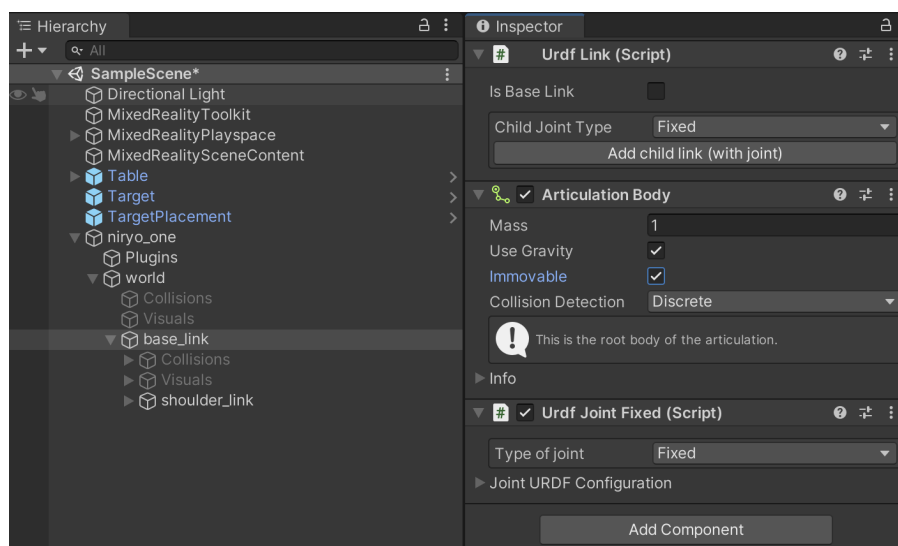
Dans vos "Assets", vous disposez déjà d'un dossier URDF contenant un fichier URDF du robot Niryo One. Si ce modèle ne répond pas à vos attentes, vous avez la possibilité de vous rendre sur le site officiel de Niryo pour télécharger le fichier adapté à vos besoins. Il est important de noter qu'un fichier URDF complet n'existe pas en tant que tel. Vous serez donc amené à le reconstruire vous-même à partir des informations fournies par le fabricant.

Sinon, rendez vous dans `Assets/URDF/Niryo_one` et faites clique droit sur `niryo_one` et sélectionnez sur 'Import robot from selected URDF file' puis sur 'Import URDF'

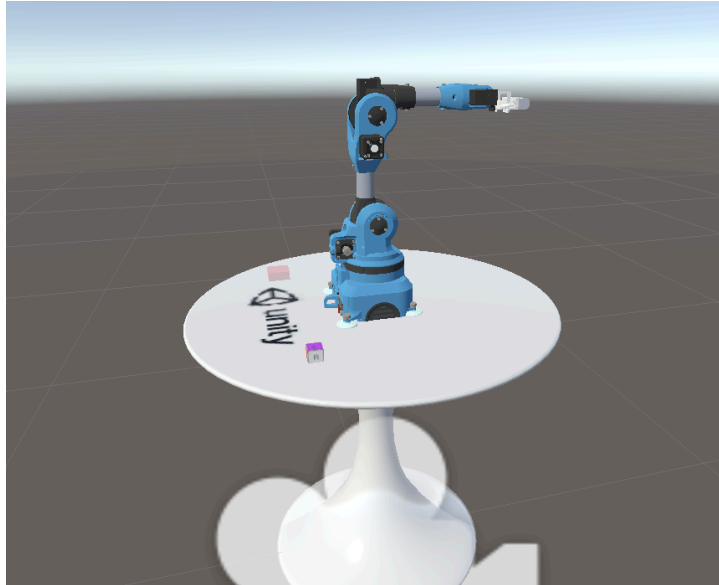
Une fois le robot généré, cliquez sur l'objet **niryo_one** et changez les paramètres du robot.



Ensuite, rendez-vous dans **base_link** et cochez la case '**immovable**'. Ainsi, votre robot sera accroché à la table et ne tombera pas.



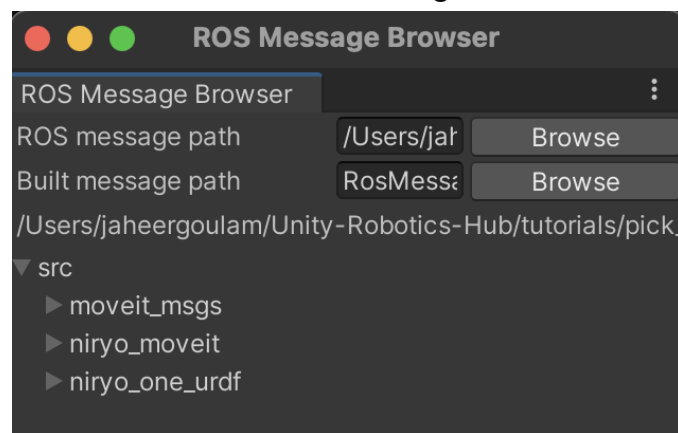
A ce stade, vous devriez avoir votre scène Unity comme ci-dessous.



Maintenant, rendez vous dans la barre de menu et cliquez sur Robotics -> ROS settings, cela va créer un dossier Ressources dans les Assets. Ensuite, glissez le **ROSConnectionPrefab** dans votre scène Unity.

Allez sur Robotics -> ROS messages et choisissez le bon répertoire pour ROS messages path, /Unity-Robotics-Hub/tutorials/pick_and_place/ROS/src

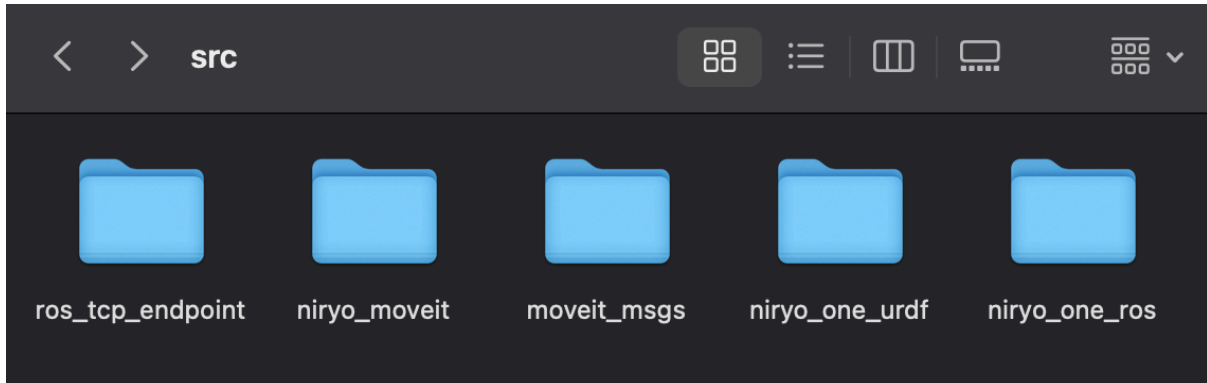
Vous devriez avoir tous ces dossiers dans ROS Messages.



Il est possible qu'il vous manque **moveit_msgs** car ce dernier est vide. Pour régler ce problème, lancez la commande :

```
git clone
https://github.com/ros-planning/moveit_msgs/tree/3175099a2b7fa75f74c36850a7
16b0fe603c8947
```

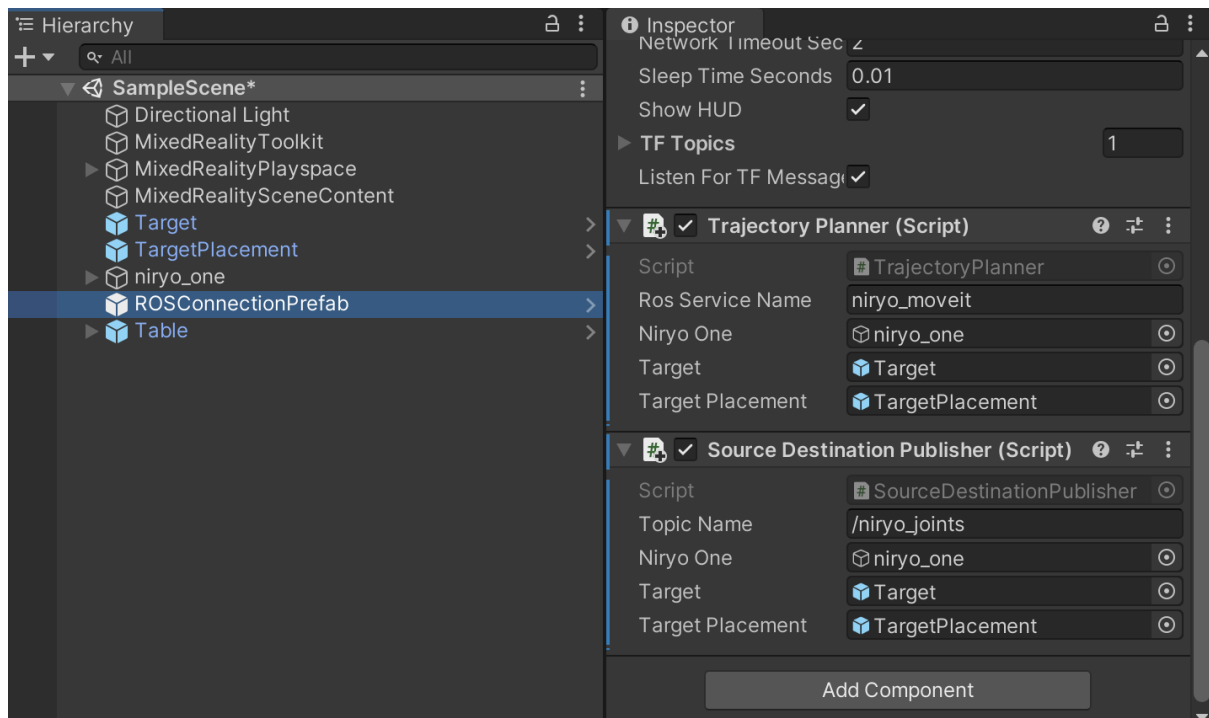
Vous aurez alors créé un nouveau dossier `moveit_msgs` sur votre pc. Rendez vous dans `/moveit_msgs/msg` et glissez tout ce qu'il y a dans votre répertoire source `src/moveit_msgs`



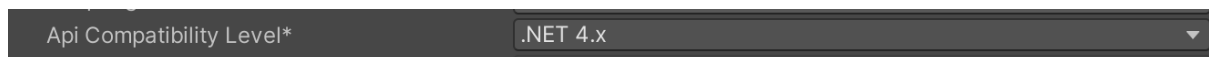
On retourne sur Unity et toujours dans la fenêtre ROS messages, on va build :

- RobotTrajectory.msg dans `moveit_msgs/msg`
- NiryoMoveitJoints.msg dans `niryo_moveit/msg`
- NiryoTrajectory.msg dans `niryo_moveit/msg`
- MoverService.msg dans `niryo_moveit/srv`

Ensuite, allez dans **ROSConnectionPrefab** et appuyez sur 'Add Component' pour ajouter les deux scripts **Trajectory Planner** et **Source Destination**. Puis, vous glissez les objets de notre scène dans les emplacements prévus, comme sur la capture ci-dessous.

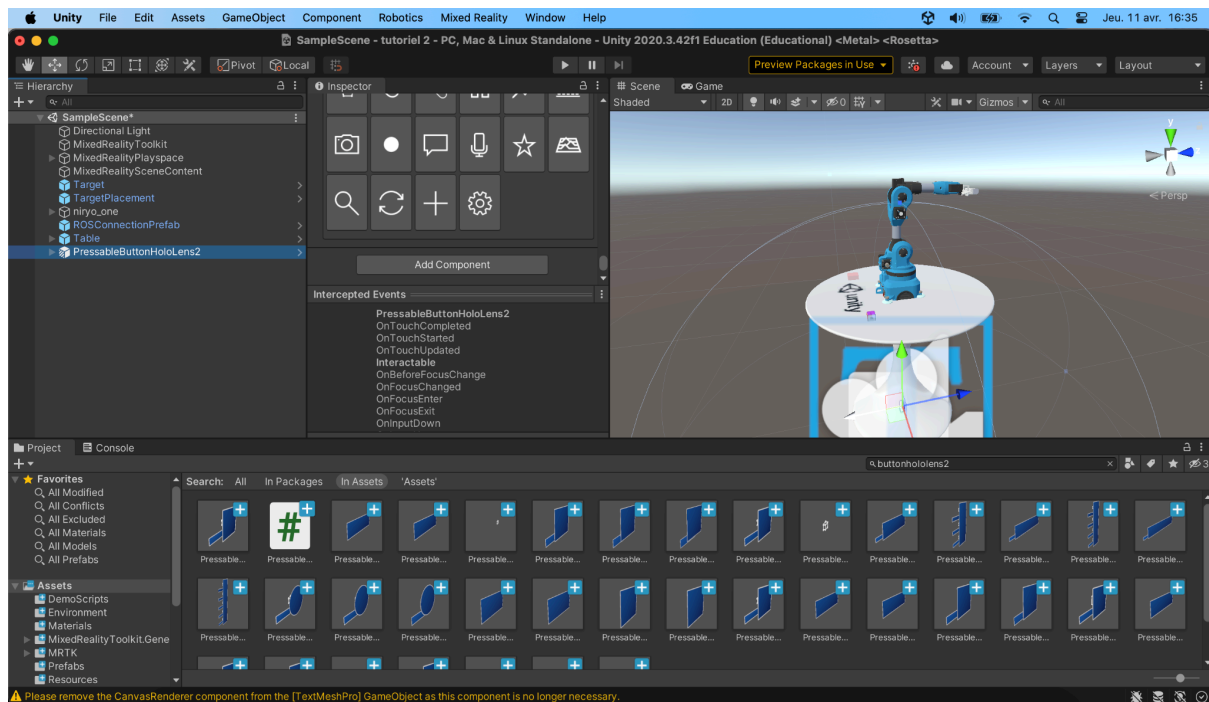


A ce stade, vous pourriez avoir une erreur sur la console, pour corriger ceci rendez vous dans la barre de menu Edit -> Project Settings -> Player -> Other Settings et choisissez **.Net 4.x**

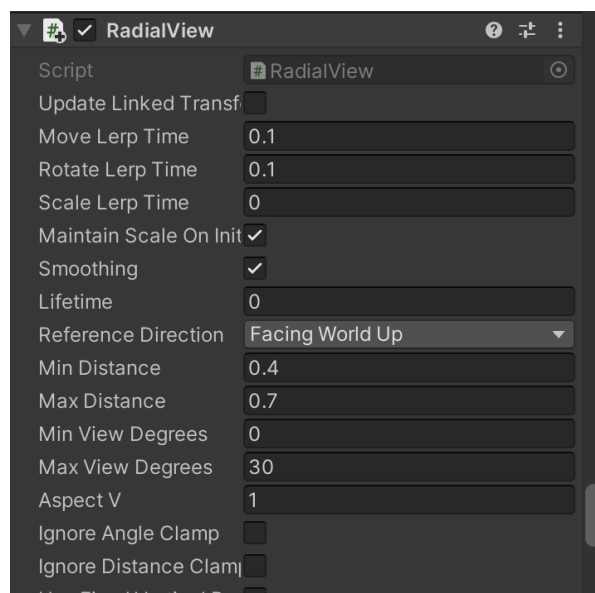


Maintenant, nous allons ajouter un bouton virtuel qui va lancer le script **TrajectoryPlanner**, c'est lui qui va actionner le robot pour qu'il déplace le robot.

Pour ce faire, rendez-vous sur la fenêtre 'Project' et cherchez **pressablebuttonhololens2**, et glissez cet objet dans votre scène Unity.

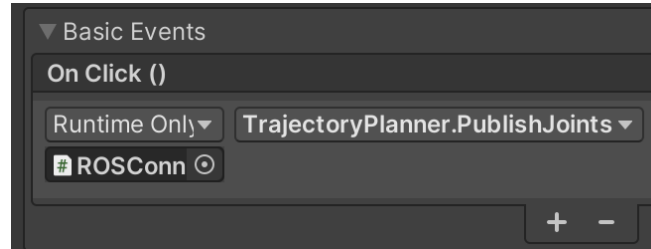


Maintenant, on veut que ce bouton nous suive avec notre tête. Pour cela on va ajouter un nouveau composant 'Add Component' et ajouter '**Radial View**'. On change les paramètres, **min distance** et **max distance**



Vous pouvez aussi changer la taille de ce bouton, en changeant le scale X et Y.

Maintenant, nous allons nous rendre dans Basic Events et glisser l'objet **ROSConnectionPrefab** depuis notre 'hiérarchie', puis cliquer sur `TrajectoryPlanner` -> `Publish.Joints`. Cela aura pour but d'activer le script au toucher de ce dernier.



À cette étape, il ne reste plus qu'à réaliser l'installation et la configuration de ROS pour assurer le bon fonctionnement de l'ensemble. Nous aborderons cette partie ultérieurement.

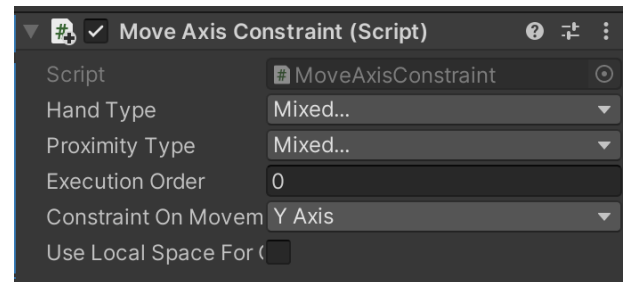
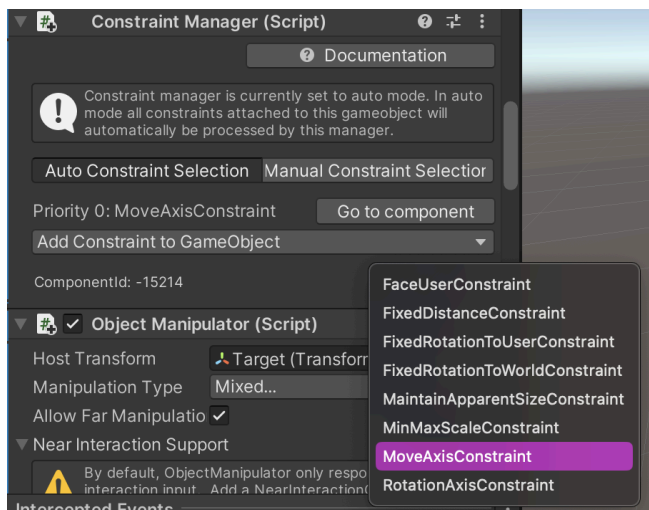
Remarque :

Nous allons maintenant effectuer quelques ajustements mineurs pour éviter tout désagrément lors du déploiement sur le Hololens 2.

Il est crucial de comprendre que la caméra virtuelle du MRTK se positionnera systématiquement à la coordonnée (0,0,0), conformément au fonctionnement du plugin. Cette configuration permettra à la caméra de s'aligner correctement avec la scène Unity. Il est également essentiel de s'assurer que la rotation de cette caméra soit réglée sur (0,0,0) pour prévenir toute inversion de votre scène dans le casque.

De ce fait, lors des tests de votre application, il est conseillé de placer votre casque sur le sol au moment de son démarrage afin de garantir que votre table soit correctement positionnée au niveau du sol, évitant ainsi qu'elle ne paraisse en élévation.

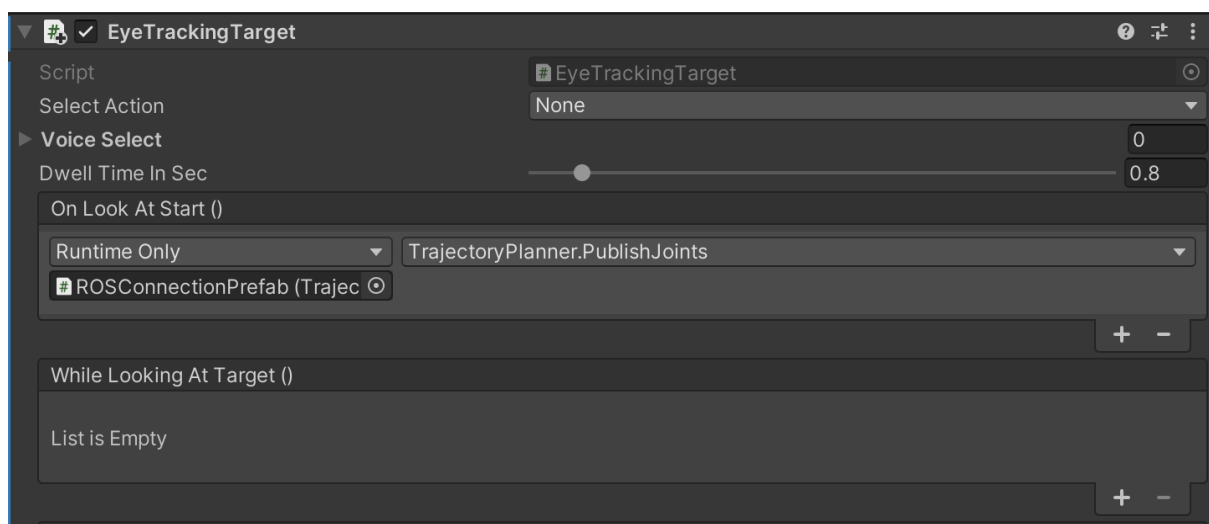
On veut maintenant pouvoir bouger le cube. Pour ce faire, cliquez sur l'objet Target et ajoutez le composant 'Object Manipulator'. Puis on va ajouter une contrainte pour pas que l'on bouge selon l'axe Y.



Quand on sera en mode '**Game**', on pourra simuler la **manipulation Hololens** en appuyant sur **espace**.

On va maintenant améliorer le projet en commençant à utiliser le **Eye Tracking** du casque pour l'activation du bouton. [8]. Suivez le tuto pour l'activation du Eye Gaze.

Maintenant, nous allons créer un deuxième bouton virtuel, nommé "**pressablebuttonhololens2**". Ce bouton sera positionné juste en face du robot, sans nécessité de suivre notre tête. Nous lui ajouterons un composant "Eye Tracking Target" via l'option "Add Component". Ensuite, nous indiquerons au bouton de déclencher le script lorsqu'il est regardé. Nous le configurons pour cela dans la fonction "On Look At Start()".



Ajoutons une dernière étape. Dans le dossier "Assets" -> "DemoScripts", ouvrez le script "**TrajectoryPlanner.cs**". Nous allons le modifier afin de réinitialiser la position du cube à sa position initiale après un délai de 5 secondes. Ainsi, nous pourrons relancer le script en appuyant sur le bouton.

Ajoute un champ pour stocker la position initiale de `m_target` au début de ton script

```
//position initial  
private Vector3 initialTargetPosition;
```

Ensuite, ajoutez ce code pour Initialiser ce champ dans la méthode `Start()` pour capturer et stocker la position initiale de `m_target` :

```
void Start()  
{  
    //récupérer la position initiale  
    initialTargetPosition = m_Target.transform.position;  
}
```

Ajoute une méthode à ton script pour gérer la réinitialisation de `m_target` après un délai spécifié. Utilise une coroutine pour introduire le délai :

```
Frequently called 1 usage  
IEnumerator ResetTargetPositionAfterDelay(float delay)  
{  
    // Attend le délai spécifié  
    yield return new WaitForSeconds(delay);  
  
    // Réinitialise la position de m_target à sa position initiale  
    m_Target.transform.position = initialTargetPosition;  
}
```

On va démarrer la coroutine après que la pince ai déposé le cube à l'emplacement voulu.

```
// All trajectories have been executed, open the gripper to place the target cube  
OpenGripper();  
  
// À la fin de la méthode ExecuteTrajectories  
StartCoroutine(routine: ResetTargetPositionAfterDelay(5));
```

VI. Intégration avec ROS

- VI.1 Installation et Configuration de la communication entre Unity3D et ROS

Les instructions détaillées pour la configuration requise de ce projet sont fournies dans **l'annexe**. Avant de commencer, assurez-vous d'avoir correctement installé une machine virtuelle et téléchargé la version appropriée d'Ubuntu, en respectant les spécifications recommandées. En suivant attentivement les étapes décrites dans l'annexe, vous devriez être en mesure de configurer votre environnement de développement sans difficulté.

- VI.4 Tests et validation de la simulation

De retour sur Unity3D il faut préciser l'adresse IP de la machine virtuelle, pour cela comme indiqué dans la II.3.2, il faut lancer la commande sur le terminal :

```
hostname -I
```

Copiez l'adresse IP obtenue et, dans Unity3D, naviguez jusqu'à `Robotics -> ROS Settings`. Dans le champ `ROS IP Address`, collez l'adresse IP que vous avez précédemment copiée.

Vous pouvez se mettre en 'Game mode' et l'icone ci-dessous devrait changer de couleur.



Sur la machine virtuelle, vous verrez normalement voir sur votre terminal :

```
[INFO] [1712855055.853254]: Connection from 192.168.1.100
[INFO] [1712855055.920068]: RegisterSubscriber(/tf, <class 'tf2_msgs.msg._TFMessage.TFMessage'>) OK
[INFO] [1712855055.942004]: RegisterPublisher(/niryo_joints, <class 'niryo_moveit_msgs._NiryoMoveitJoints.NiryoMoveitJoints'>) OK
[INFO] [1712855055.963235]: RegisterRosService(niryo_moveit, <class 'niryo_moveit_srv._MoverService.MoverService'>) OK
```

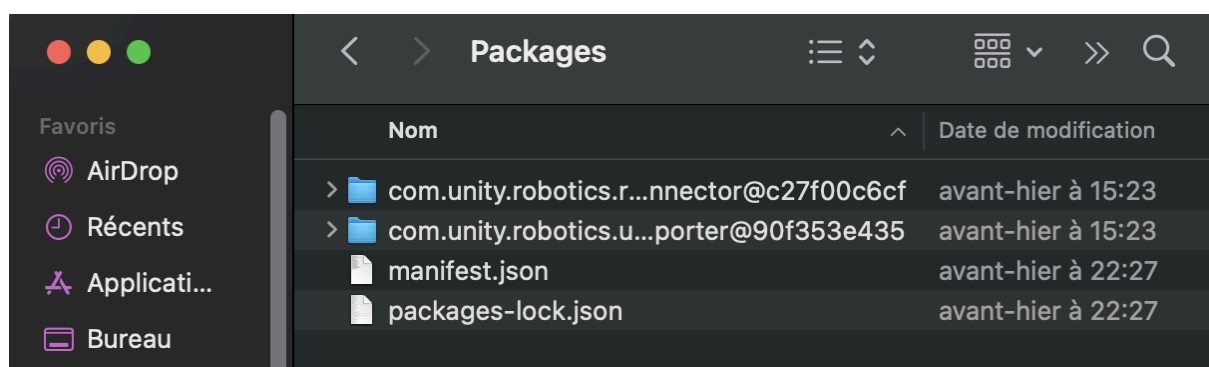

Ce qui indique qu'il est prêt à recevoir des messages, et donc vous pouvez appuyez sur le bouton pour lancer le script.

VII. Déploiement sur Hololens 2

Maintenant que notre projet est pleinement fonctionnel au sein d'Unity, l'étape suivante consiste à le déployer sur le Hololens 2. Avant de procéder, quelques ajustements supplémentaires dans Unity sont nécessaires pour assurer une transition fluide. Une fois ces configurations terminées, nous passerons à Visual Studio pour finaliser le processus de déploiement.

- VII.1 Préparation du projet Unity3D pour Hololens

- **Ouvrez deux fenêtres de l'Explorateur** de fichiers sur votre ordinateur.
- **Dans la première fenêtre**, naviguez jusqu'au répertoire de votre projet Unity, puis accédez au dossier Packages.
- **Dans la deuxième fenêtre**, dirigez-vous vers le dossier de votre projet, puis naviguez vers Library -> PackageCache.
- **Localisez les dossiers** nommés **ROS TCP Connector** et **URDF Importer** dans le répertoire Packages de la première fenêtre.
- **Glissez et déposez** ces deux dossiers depuis la deuxième fenêtre vers le répertoire **Packages** dans la première fenêtre.



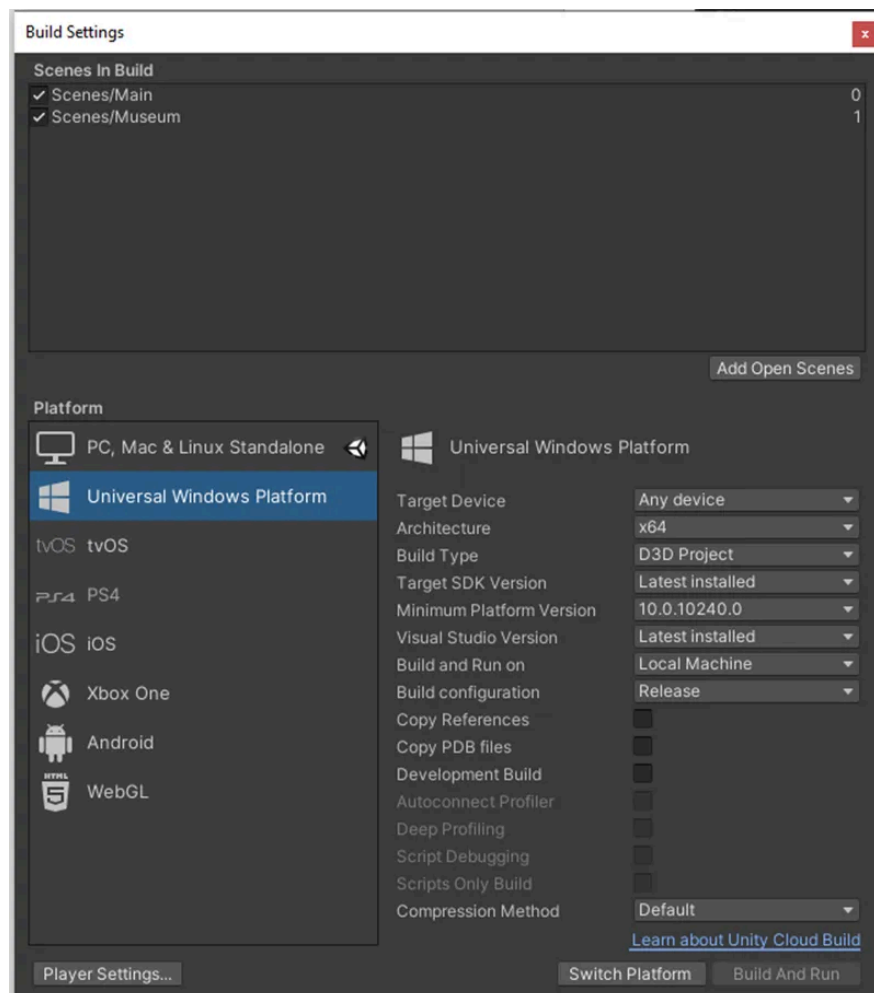
Dans Unity, nous entamons d'abord en naviguant vers les 'Project Settings', puis vers 'XR Plugin Management' où nous cochons la case '**Windows Mixed Reality**'.

Ensuite, nous nous dirigeons vers 'Quality'. Si nécessaire, nous ajustons la qualité graphique du projet, ce qui nous fera gagner du temps lors du déploiement.

Pour garantir que notre application fonctionne avec ROS, il est essentiel de lui accorder l'accès à Internet, car nous allons nous connecter au même réseau wifi à la fois sur votre machine virtuelle et sur le casque. Pour ce faire, rendez-vous dans 'Player' -> 'Publishing Settings' et cochez les cases : '**InternetClientServer**' et '**PrivateInternetClientServer**'.

Toujours dans l'onglet 'Player', vous avez la possibilité de modifier le nom de votre application dans la section '**Product Name**'.

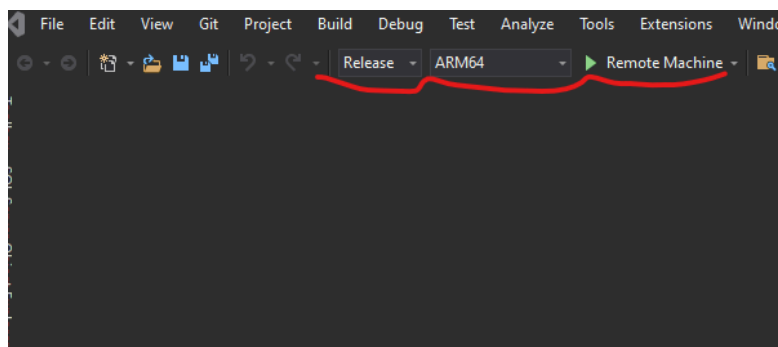
Maintenant, passons à l'étape du build. Vous devez sélectionner les mêmes paramètres que ceux illustrés dans la capture ci-dessous. Vous pouvez modifier le périphérique cible par Hololens et la version de Visual Studio par celle de 2019. Assurez-vous également d'ajouter votre scène dans '**Add Open Scene**'. Pour l'architecture, vous pouvez le laisser comme ceci ou essayer ARM64 si vous rencontrez des problèmes dans la partie suivante.



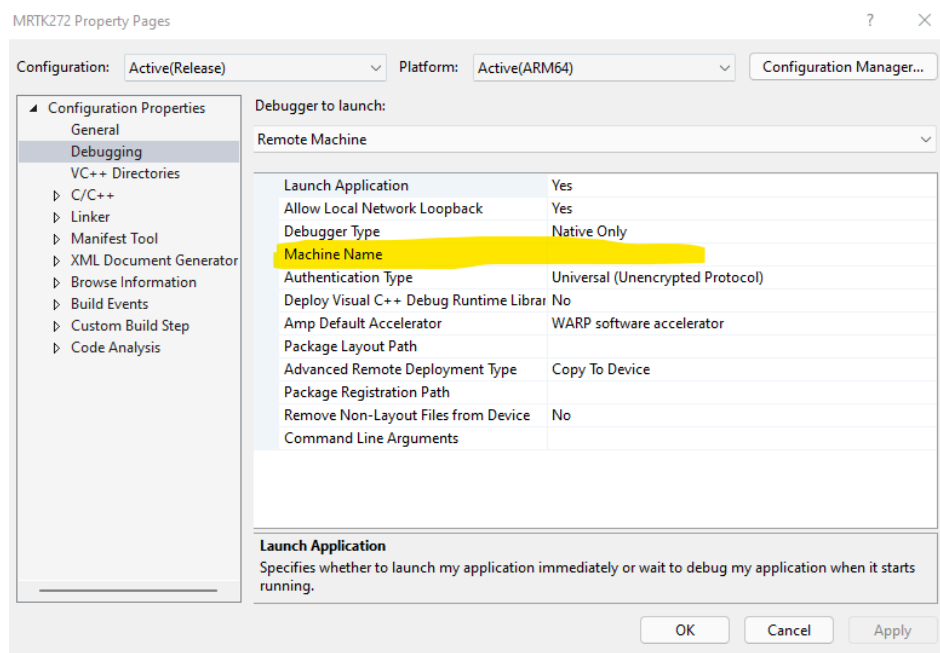
Quand tous les paramètres seront corrects, appuyez sur build. Une fenêtre d'explorer s'ouvrira et vous devriez choisir un dossier dans lequel vous mettrez le build.

- VII.2 Configuration de l'environnement de déploiement sur Visual Studio

Rendez vous dans le dossier où vous aviez placé votre build précédemment compilé. Ouvrez le fichier .sln sur Visual Studio 2019. Choisissez les bons paramètres, Release, ARM64 et remote machine.



Après cela, accédez à Project > Properties > Configuration Properties > Debugging. Saisissez l'adresse IP de votre casque Hololens. Assurez-vous de vous connecter au même réseau wifi que celui de votre machine virtuelle.



Appuyez sur Apply puis appuyez sur Remote machine pour commencer le débogage.

Cela prendra un certain temps, mais à la fin vous verrez apparaître une fenêtre disant que la compilation a réussi. Vérifiez que vous n'avez aucune erreur.

Si ce n'est pas le cas, vous verrez apparaître une fenêtre de débogage sans arrêt. Vous pouvez tout simplement arrêter le débogage. A présent tout devrait fonctionner dans votre casque.

- VII.3 Tests et dépannage sur Hololens 2

Vous pouvez tout simplement faire la même manipulation sur votre workspace ROS et en appuyant (ou en regardant) le bouton, cela devrait lancer le script.

IX. Résolution de Problèmes et Dépannage

- IX.1 Problèmes courants et leurs solutions

Un problème récurrent est que le robot ne reçoit pas bien les messages ROS ou qu'il bug et n'arrive pas à bouger correctement.

Solutions et raisons de ce problème :

- ➔ Que vous travailliez avec Unity ou sur le Hololens, il est conseillé de relancer l'application et de recommencer le processus pour assurer un fonctionnement optimal. Assurez-vous que votre espace de travail ROS est actif. Généralement, cette configuration fonctionne bien.
- ➔ Il est crucial de ne pas modifier la position de votre scène Unity ; maintenez tout à la position (0, 0, 0). Si ces paramètres sont altérés, le script pourrait ne pas fonctionner correctement et le calcul de la trajectoire pourrait être compromis.
- ➔ Évitez d'activer l'objet 'manipulator' pour votre robot. Le robot a été assemblé à partir du modèle URDF manipulator et, de ce fait, il ne se comporte pas comme un objet 3D ordinaire. Activer ce composant pourrait endommager le robot.
- ➔ Considérez également la possibilité de déplacer votre cube si celui-ci semble être hors de portée du robot. Un ajustement de sa position pourrait résoudre le problème d'accessibilité.

X. Conclusion

- X.1 Récapitulatif des réalisations

Jusqu'à présent, votre projet a été configuré pour fonctionner à la fois sur Unity3D et sur HoloLens 2. Vous devriez être en mesure d'interagir avec le bouton virtuel, soit en le touchant, soit en utilisant le regard (Gaze), pour activer le script qui mettra en marche le robot.

- X.2 Limitations et perspectives d'amélioration

Je vous ai présenté un échantillon de code qui pourrait s'avérer bénéfique pour le développement de votre projet. Il vous appartient maintenant d'explorer les modifications possibles à apporter au code pour optimiser le projet.

Étant donné que la simulation virtuelle fonctionne correctement, nous pourrions envisager de la tester sur un robot réel dans notre laboratoire. Il serait judicieux de placer le robot physique sur une table et de superposer le robot virtuel directement sur ce dernier, en procédant de même pour le cube. Ainsi, les coordonnées du cube resteraient identiques, ce qui vous permettrait d'envoyer des messages ROS simultanément à l'application et au robot réel.

XI. Annexes

- XI.1 Références

- <https://github.com/Unity-Technologies/Unity-Robotics-Hub> [1]
- <https://learn.microsoft.com/fr-fr/windows/uwp/get-started/universal-application-platform-guide> [2]
- <https://developer.microsoft.com/en-us/windows/downloads/windows-sdk/> [3]
- <https://learn.microsoft.com/fr-fr/training/paths/beginner-hololens-2-tutorials/> [4]
- <http://wiki.ros.org/melodic/Installation/Ubuntu> [5]
- <http://wiki.ros.org/>
- <https://www.youtube.com/watch?v=HV1v8mXNmLA> [6]
- <https://github.com/microsoft/MixedRealityToolkit-Unity/releases/> [7]
- <https://learn.microsoft.com/en-us/windows/mixed-reality/mrtk-unity/mrtk2/features/input/eye-tracking/eye-tracking-basic-setup?view=mrtkunity-2022-05> [8]