Augmented Reality Applications for Human-Robot Interaction – EGH400 Capstone Project

Control of Mobile Robots Using the Microsoft HoloLens 2 – Documentation

Student Author: Torsten Sprey

Student Id: n10772057

Supervisor: Prof. Niko Suenderhauf

Version: 1.0

Last Update: 25/10/2023

Table of Contents

About 2
Getting the Project 3
Importing from GitHub Repository 3
Setting up Unity for HoloLens 2 Development 4
Adding MRTK to Unity 4
NuGet Package Manager 6
Using Unity for HoloLens 2 Development 6
Unity API for HoloLens 6
Useful Assets From the Unity Asset Store 6
Optimisation 7
Building and Deploying to the HoloLens 8
Localisation of Markers 9
Microsoft's QR Code Tracking Sample 9
Connecting Unity to ROS 11
Controlling turtlesim_node from Unity 11
User Guide 13
Menu 13
Connecting to Rosbridge 13
Localising the Robot 14
Manual Control 17
Voice Commands 18
Navigation 19
Tracking 20
Development Guide 21
Voice Commands 21
Custom ROS Subscribers 22
Troubleshooting 25
Deploy Failed on Visual Studio 25
Visual Studio NuGet Package Restore Failed Unable to Find Version 25
Unity WSATestCertificate is Expired 25
ROS# Project Newtonsoft.Json.dll Not Found Error 25
Unity Event Callbacks Not Fully Executing 25

About

With the continuous advancement of technology, robots are now capable of undertaking increasingly complex tasks, leading to a rise in human-robot interaction (HRI) across various industries, including manufacturing and healthcare. While HRI holds the potential to enhance efficiency and safety, it also presents significant hurdles that need to be overcome. These challenges encompass creating user-friendly interfaces for humans and mitigating the risks associated with accidents or malfunctions. The paramount importance of safety in HRI necessitates the development of robots that can interact safely with humans, operate within defined safety boundaries, and minimize the likelihood of mishaps. Fortunately, augmented reality (AR) technology has emerged as a promising solution to address these challenges. By offering a more natural and intuitive interface, AR can improve safety measures, facilitate seamless communication, provide immediate and unambiguous feedback, and even enable remote collaboration between humans and robots.

Broadly, this project aimed to develop software applications that utilise the versatile interactive capabilities of AR technology to ease collaboration between humans and robots. By providing intuitive interfaces that translate familiar human actions to robot commands and deliver immediate unambiguous feedback, AR technology can significantly increase workplace efficiency and safety compared to typical HRI communication methods such as command-line interfaces.

To this end, a project with the aim to develop a software application/suite that utilises AR to improve HRI was undertaken. This one year project was completed as part of the Queensland University of Technology's (QUT) EGH400 Research Project subject. For this project, access to a Microsoft HoloLens 2 AR headset and an AgileX Scout Mini mobile robot used by QUT Centre of Robotics was given. This project focused on the control of this mobile robot using the HoloLens 2.

The purpose of this document is to provide documentation for this project including installation, usage, development and deployment instructions. It will also provide instructions and resources for creating a similar Unity project and incorporating various useful features used in this project, such as localisation of real-world markers and connecting to ROS.

Getting the Project

This section will detail how to import the Unity project from the GitHub repository. It will also provide the steps to creating a Unity project like this from scratch.

Importing from GitHub Repository

The Unity project built for this project can be found in the following repository:

https://github.com/deltasprey/AR_for_HRI

The steps to import this project into the Unity Editor are as follows:

1. Download the folder from the GitHub link above and the Microsoft Mixed Reality Feature Tool.
2. Add the project into the Unity Hub list.
3. Open the project. A currently installed/preferred Unity version can be used with **UWP** selected as the target platform. Press **Change Version** and **Continue**.
4. In the MRTK Project Configurator click **Use OpenXR (Recommended)**. Close the **Project Settings** window and on the next window click **Skip until next session**.
5. In the **Assets** folder click on **NuGet**, select **Load on startup** and click **Apply**.
6. Click off and click back on the Unity Editor which should cause it to start compiling scripts. When the warning dialogue appears click **Yes** to restart the Editor. If it crashes just reopen the project from Unity Hub.
7. Go to **NuGet → Manage NuGet Packages** on the menu bar. Search for **"Microsoft.MixedReality.QR"** and install it if it isn't already installed.
8. Open the **Microsoft Mixed Reality Feature Tool**, select this project and select any packages that are currently installed. Click **Get Features, Validate**, **Import** and **Approve** to update these packages to the latest versions.
9. Complete the remaining steps of the **MRTK Project Configurator**. If it doesn't appear after the previous step, restart the editor.
10. Make sure the **OpenXR** and **Microsoft HoloLens feature group** checkboxes are checked under the **Universal Windows Platform settings** (tab with the windows logo) **→ Plug-in Providers**.
11. Click **Skip this step**, **Next** and **Done** to complete the setup.
12. If there are still errors and the editor was not restarted in steps 6 or 10, **restart** the editor now.
13. Switch platform by going to **File → Build Settings...**, selecting Universal Windows Platform and **Switch Platform** (if not done in step 3). Make sure the following settings are active:
    a. **Architecture**: ARM 64-bit
    b. **Build Type**: D3D Project
    c. **Target SDK Version**: Latest Installed
    d. **Minimum Platform Version**: 10.0.10240.0

    e. **Visual Studio Version**: Latest installed
    f. **Build and Run on**: Local Machine
    g. **Build configuration**: Release (there are known performance issues with Debug)

This project should now be buildable. To do so, follow the steps in the Building and Deploying to the HoloLens section below. If any errors remain or occur during building and deployment, refer to the Troubleshooting section.

Setting up Unity for HoloLens 2 Development

To create a similar project to this from scratch, the following steps and resources should prove useful.

Adding MRTK to Unity

1. Install **Visual Studio** with the following Workloads:
    a. .NET desktop development
    b. Desktop development with C++
    c. Universal Windows Platform development
    d. Game development with Unity

2. Install **Unity** with the latest Editor and **Universal Windows Platform** (UWP) module.

3. Create a new **3D Core** project.

4. Switch platform by going to **File → Build Settings...**, selecting Universal Windows Platform and **Switch Platform**. Make sure the following settings are active:
    a. **Architecture**: ARM 64-bit
    b. **Build Type**: D3D Project
    c. **Target SDK Version**: Latest Installed
    d. **Minimum Platform Version**: 10.0.10240.0
    e. **Visual Studio Version**: Latest installed
    f. **Build and Run on**: Local Machine
    g. **Build configuration**: Release (there are known performance issues with Debug)

5. Download and run the Microsoft Mixed Reality Feature Tool. (Optional) Follow the steps in the Mixed Reality Feature tool documentation linked below.

6. Press **Start.** Change the **Project Path:** to the folder containing the Unity project and press **Discover Features**. Get the following features:
    a. In *Mixed Reality Toolkit*
        i. ~~Mixed Reality Toolkit Extensions~~
        ii. Mixed Reality Toolkit Foundation
        iii. Mixed Reality Standard Assets
        iv. Mixed Reality Toolkit Tools (optional)
    b. In *Platform Support*
        i. Mixed Reality OpenXR Plugin
    c. In *Spatial Audio*

    i. Microsoft Spatializer (optional)

7. Click **Validate**, **Import** and **Approve**. **Exit** then go back to the Unity project.
8. After Unity has imported the packages, a warning appears asking to enable the backends by restarting the Editor. Select **Yes**.
9. If Unity crashes when trying to restart, don't worry, nothing is wrong. Just reopen the project from the Unity Hub.
10. The **MRTK Project Configurator** window should now be visible. Select **Unity OpenXR plugin (recommended)**.
11. This should open the **XR Plug-in Management** in the **Project Settings** window. Check the **OpenXR** checkbox under the **Universal Windows Platform settings → Plug-in Providers**.
12. Check the **Microsoft HoloLens feature group** checkbox.
13. Click the yellow exclamation mark next to the **OpenXR** checkbox or go to **XR Plug-in Management → Project Validation** in the **Project Settings** sidebar.
14. Select **Fix All**. Ignore any remaining warnings/errors for now.
15. Go to **XR Plug-in Management → OpenXR** in the **Project Settings** sidebar and add the following **Interactions Profiles**:
    a. Eye Gaze Interaction Profile
    b. Microsoft Hand Interaction Profile
16. Select the warning sign next to **Eye Gaze Interaction Profile** to bring back the **Project Validation** window. Make sure to be on the UWP platform tab and select **Fix All** to resolve the validation issues. Note that there may be issues that remain. In that case, select **Fix All** again, ignore any issues that are marked *Scene specific*, and then read the recommendations for the remaining issues (if any) and make any desired changes.
17. Close the **Project Settings** window and select **Next** on the **MRTK Project Configurator** window.
18. Change the **Audio spatializer:** to **Microsoft Spatializer** (if that feature has been added) and click **Apply** and **Next**.
19. If user interface (UI) elements are going to be used in the project, then **Import TMP Essentials**. Otherwise click **Skip This Step**.
20. Click **Done**.
21. Finally, go to **Mixed Reality → Toolkit → Add to Scene and Configure...**

Many of the steps above were taken from the following resources:

Mixed Reality Feature Tool documentation: https://learn.microsoft.com/en-us/windows/mixed-reality/develop/unity/welcome-to-mr-feature-tool

Unity HoloLens 2 beginner tutorial: https://learn.microsoft.com/en-us/training/paths/beginner-hololens-2-tutorials/

NuGet Package Manager

NuGet Package Manager adds access to packages that aren't available through distributors such as the Unity Asset Store. An example of its use is the installation of the **"Microsoft.MixedReality.QR"** package used to track QR codes using webcams. If the NuGet package manager is required in the Unity project, complete the following steps:

1. Download NuGet for Unity **.unitypackage** file from https://github.com/GlitchEnzo/NuGetForUnity/releases
2. In the **Assets** folder, **right click → Import Package → Custom Package.** Select the downloaded file and **import** it.
    a. If a **NuGet** tab doesn't appear on the menu bar, click the **NuGet** file in the **Assets** folder, enable **Load on startup** and **Apply**.
    b. If it still doesn't appear then restart (close and reopen) the Unity project.

Using Unity for HoloLens 2 Development

A few extra resources to aid in development.

Unity API for HoloLens

Is useful sometimes for adding features or finding out what functions do what.

https://learn.microsoft.com/en-us/windows/mixed-reality/develop/unity/unity-development-overview?tabs=arr%2CD365%2Chl2

Useful Assets From the Unity Asset Store

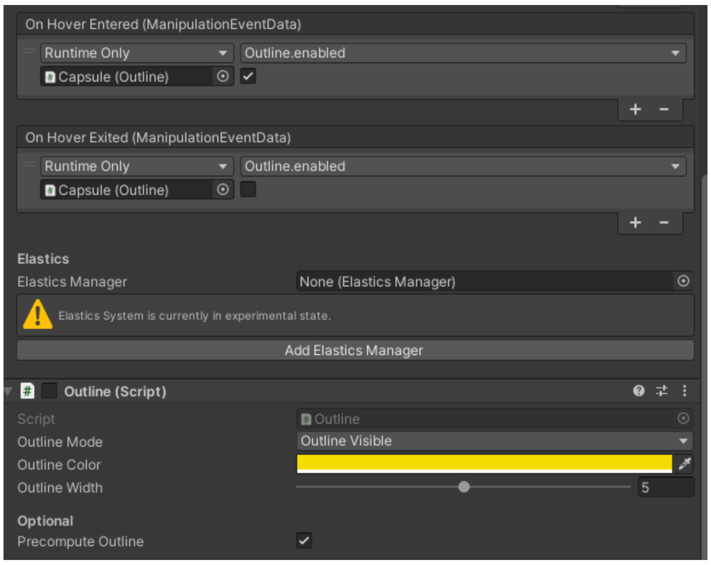
In-Game Debug Log for AR and VR devices by Tobiesen

Something encountered often when developing projects like this is that things which work in the Editor don't always work on the HoloLens. This package displays the debug logs and stack traces that would normally only be seen in the Console of the Unity Editor, which greatly speeds up debugging.

This project improved upon this packing by having it write to a scrollable UI.

Quick Outline by Chris Nolet

This package outlines in-game objects, obviously. Its best feature is that it can be turned on and off which can be used to provide intuitive visual feedback for when a virtual object has focus (is looked at or within grabbing distance). As such, it pairs well with Mixed Reality Toolkit's Object Manipulator script as shown below. However, one downside is that it completely obscures the objects on the HoloLens, even if the recordings show otherwise.

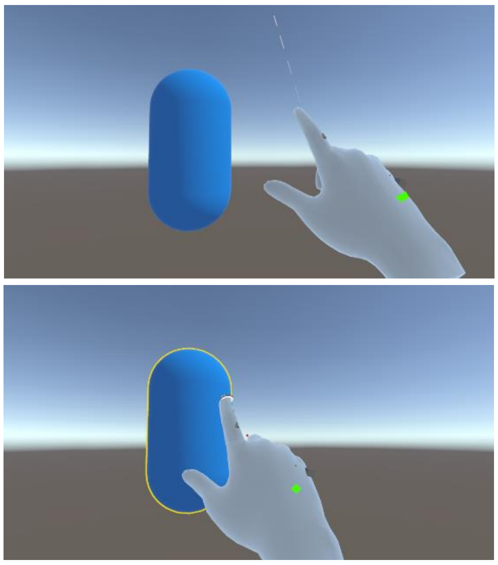

On Hover Entered (ManipulationEventData)
Runtime Only | Outline.enabled
Capsule (Outline) ✓

On Hover Exited (ManipulationEventData)
Runtime Only | Outline.enabled
Capsule (Outline)

Elastics
Elastics Manager: None (Elastics Manager)
Elastics System is currently in experimental state.
Add Elastics Manager

Outline (Script)
Script: Outline
Outline Mode: Outline Visible
Outline Color: 🟨
Outline Width: 5

Optional
Precompute Outline: ✓



[Free Casual Game SFX Pack](https://assetstore.unity.com/) by Dustyroom

While the Mixed Reality Toolkit has its own sounds that can be used, it never hurts to have more to supplement user feedback with these 50 audio clips.

Optimisation

The HoloLens 2 is basically two projectors powered by outdated smartphone hardware. As such, it cannot handle large amounts of computations that can be caused by things such as particle effects, large meshes, shaders and other high load calculations (things with lots of loops). The following article details methods of getting the best performance out of the Unity project to have a good AR experience.

https://learn.microsoft.com/en-us/windows/mixed-reality/mrtk-unity/mrtk2/performance/perf-getting-started?view=mrtkunity-2022-05

Building and Deploying to the HoloLens

1. Go to **File → Build Settings → Build**. Click **Add Open Scenes** to add the current scene.
2. Create a new folder called "**Builds**" in the project directory and **Select** that folder.
3. Once build is completed, open the **Builds** folder and open the **<Project Name>.sln** file in Visual Studio.
4. Configure Visual Studio for HoloLens by selecting the **Master** or **Release** (**Release** preferred) configuration and the **ARM64** architecture.
5. Click the deployment target drop-down and then do one of the following:
   a. If building and deploying via Wi-Fi, select **Remote Machine**.
   b. If building and deploying via USB, select **Device**.
6. Set the remote connection. On the menu bar, select **Project → Properties → Configuration Properties → Debugging**.
7. Click the "**Debugger to launch:**" drop down and then select **Remote Machine** if it's not selected already. Set the **Authentication Mode** to **Universal (Unencrypted protocol)**.
8. In the Machine Name field, enter the IP address of the HoloLens. To find the HoloLens' IP address go to **Settings → Updates & Security → For developers** and scroll to the bottom.
9. To deploy to the HoloLens and automatically start the app without the Visual Studio debugger attached, select **Debug → Start Without Debugging**. Or to deploy to the HoloLens without having the app start automatically, select **Build → Deploy Solution**.

Source: https://learn.microsoft.com/en-us/training/paths/beginner-hololens-2-tutorials/ (units 2 and 6).

Localisation of Markers

Both the HoloLens and the robot have their own independent world coordinate frames. Because of this, the robot has no idea where the HoloLens is. And more importantly, the HoloLens doesn't know where the robot is either. To solve this problem, there are two general industry standard solutions:

1. Place a marker in a known stationary location and use it to make the world coordinate frames of both the HoloLens and robot equal to each other. The poses of both are now the same in both world coordinate frames.
2. Place a marker on the robot and calculate the coordinate frame transform between the two world coordinate frames.

This project used the second option. To do so, a solution was found that could track QR codes (which were displayed on the robot's monitor).

Microsoft's QR Code Tracking Sample

Positives	Negatives
• Works almost out of the box. • Tracks stationary QR codes well. • Custom markers can be easily implemented.	• The refresh rate is low and sporadic, making it unsuitable for continuous tracking. • The code isn't clear/is difficult to modify.

Source: https://github.com/chgatla-microsoft/QRTracking

Setup:

1. Download the folder from the GitHub link above.
2. Add the project into the Unity Hub list.
3. Open the project. A currently installed/preferred Unity version can be used with **UWP** selected as the target platform. Press **Change Version** and **Continue**.
4. Press **Ignore** when the *enter safe mode* compilation error dialogue pops up.
5. In the MRTK Project Configurator click **Use OpenXR (Recommended**) and on the next window click **Skip until next session**.
6. In the **Assets** folder click on **NuGet** and select **Load on startup** and click **Apply**.
7. Click off and click back on the Unity Editor which should cause it to start compiling scripts. When the warning dialogue appears click **Yes** to restart the Editor. If it crashes just reopen the project from Unity Hub.
8. Click **NuGet** on the menu bar and **Manage NuGet Packages**.
9. Search for **"Microsoft.MixedReality.QR"** and install it if it isn't already installed.

10. Open the Microsoft Mixed Reality Feature Tool, select this project and select any packages that are currently installed **PLUS** the **Mixed Reality OpenXR Plugin** under **Platform Support (0 of 5)**. Click **Get Features, Import** and **Approve** to update these packages to the latest version.
11. Complete the remaining steps of the MRTK Project Configurator. Importing TMP Essentials can be skipped unless UI elements are desired.
12. Open the **SpatialGraphCoordinateSystem** script and comment out lines 75 and 76.
13. Paste in the following lines in the indicated positions and save the script:
    a. SpatialCoordinateSystem rootSpatialCoordinateSystem = PerceptionInterop.GetSceneCoordinateSystem(UnityEngine.Pose.identity) as SpatialCoordinateSystem; (**Directly below line 76)**.
    b. using Microsoft.MixedReality.OpenXR; **(Directly below line 5)**.
14. Open the **Build Settings** and change the Architecture to **ARM 64-bit**.

Connecting Unity to ROS

This is a core component of this project since a connection is required to link the HoloLens and robot. The robot already has ROS setup, and with the addition of the Rosbridge ROS package that creates a public WebSocket that is visible on the network, the HoloLens can connect to it wirelessly.

The following GitHub Unity project provided by EricVoll's UWP fork of the ROS# package by siemens allows for Unity UWP projects to connect to this WebSocket. This project seems to work without issues, it just needs the features from the Mixed Reality Feature Tool (specified above) added.

https://github.com/ericvoll/ros-sharp/tree/UWP

Controlling turtlesim_node from Unity

In ROS Hosting Machine:

1. (Oracle VM VirtualBox) Setup the network with three adaptors:
   a. NAT
   b. Host-only Adapter | Promiscuous Mode: **Allow VMs** | Cable Connected: **True**
   c. Bridged Adapter | Promiscuous Mode: **Allow VMs** | Cable Connected: **True**
2. Go through the ROS **Beginner Level** tutorials http://wiki.ros.org/ROS/Tutorials
3. Open up the Terminal and run the following commands to get Rosbridge:
   a. sudo apt-get install ros-noetic-rosbridge-suite
   b. source /opt/ros/<rosdistro>/setup.bash
   c. roslaunch rosbridge_server rosbridge_websocket.launch
4. If the last command (c.) doesn't work, try closing and reopening the Terminal or restart the ROS machine.
5. In a new Terminal window**, run** rosrun turtlesim turtlesim_node
6. Determine the IP address of this machine (run **ifconfig** in a new Terminal window OR look at the network connection settings).

In Unity:

1. In the downloaded ros-sharp-UWP folder, navigate to **Unity3D → Assets**.
2. Copy the **RosSharp** folder to the **Assets** folder of the project. (Optional) The **README** and **Scenes** folder can be deleted.
3. In RosSharp/Plugins, **delete** the **System.Numerics** and **Newtonsoft.Json** dll's (they seem to prevent the project from being built).
4. In Unity, go to **Edit → Project Settings → Player → Other Settings → Configuration** and change **Api Compatibility Level** to **.NET Framework**.
5. Create an empty game object and call it "**Ros Connector**" or something similar. Add the **Ros Connector** and **Twist Publisher** scripts to this object.

6. Set the **Ros Connector** script to the **Web Socket UWP** Protocol and the IP address of the machine hosting ROS.
7. Set the **Twist Publisher** script Topic to “/turtle1/cmd_vel”.
8. Download the KeyboardControl.cs example script and add it to the **Ros Connector** game object.
9. Run the project and use Arrow/WSAD keys to move the turtle.

The above steps are compiled into an example project which can be found in the GitHub repository linked below. This project also subscribes to the ROS Topic “/turtle1/pose” using the custom subscriber script “TurtlePose.cs”. The data from it moves a game object in the scene as the Turtle Bot moves in the simulator.

https://github.com/deltasprey/Unity_RosBridge_TurtleSim_Demo

Many of the steps above were taken from the following resources. They provide more information but are also somewhat outdated.

Rosbridge tutorial: http://wiki.ros.org/rosbridge_suite/Tutorials/RunningRosbridge

ROS# wiki: https://github.com/siemens/ros-sharp/wiki (steps 1.1 and 1.2 recommended)

User Guide

This section will provide instructions for using the various features this project has implemented.

Menu

Can be opened and closed with the voice command, “Menu”. By interacting with the various toggles, graphical user interfaces (GUIs) and UI elements can be shown/hidden and functional components of this project can enabled/disabled. There are also two buttons with the following functionality:

Clear Markers: Removes all markers generated by the QR code localisation, navigation subsystem and user. This can also be triggered with the voice command, “Clear Markers”.

Exit: Quits the application. It is recommended that this button is used instead of the Home button on the HoloLens’ menu since it cleans the project up properly.

Connecting to Rosbridge

Firstly, the Rosbridge server must be started on the robot. This can be done remotely via SSH or by desktop sharing using NoMachine and opening a Terminal window (login details will be given once the user has been inducted on the use of the robot). To launch Rosbridge, enter the following command:

```
roslaunch rosbridge_server rosbridge_websocket.launch
```

This creates a localhost WebSocket on port 9090 by default. Because it is hosted on the local machine (robot), accessing it on the local network requires the IPV4 network address of the robot to be know. This can be found in the robot’s network settings or by opening a new Terminal window and running:

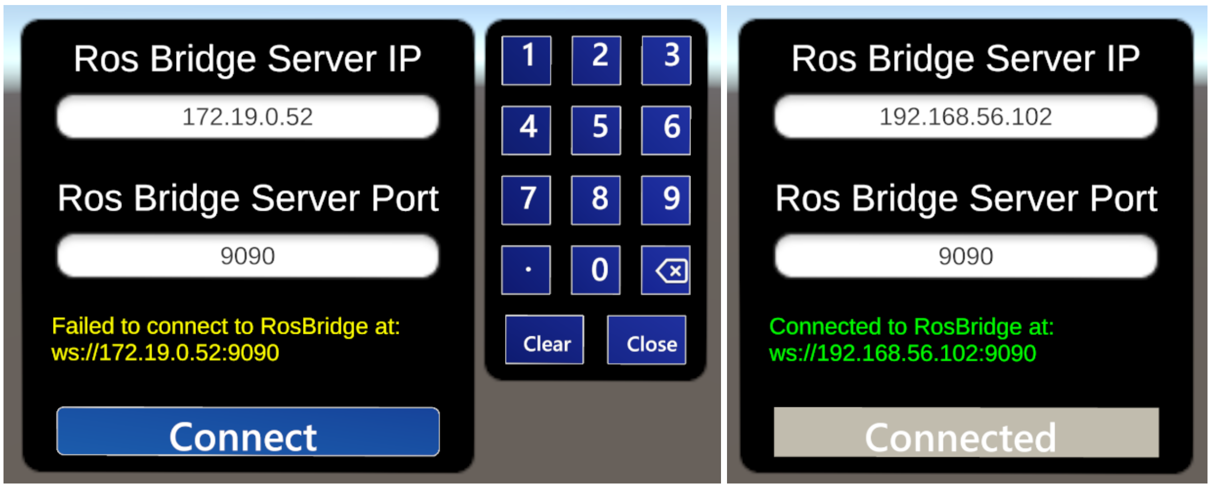
```
ifconfig
```

The WebSocket create by Rosbridge can now be connected to using the GUI shown below. Touching/Clicking the input fields will bring up a keypad that is used to enter the IP address and port of the Rosbridge WebSocket. Press the **Connect** button and the following sequence will happen.

If the IP address and port entered is correct, the GUI will look like the right image below with a connection confirmed message and the **Connect** button disabled.

Otherwise:

1. The confirmation text will be cleared and the **Connect** button will be temporarily disabled with the text changed to, “Connecting…”.
2. In the Unity Editor, the application will freeze for a few seconds. On the HoloLens, the displays will turn blank for a few seconds (no holograms visible) before returning to normal.
3. The GUI will look like the left image below with the confirmation text showing a connection failure and the **Connect** button reenabled with the text reverted.


Ros Bridge Server IP
172.19.0.52
Ros Bridge Server Port
9090
Failed to connect to RosBridge at:
ws://172.19.0.52:9090

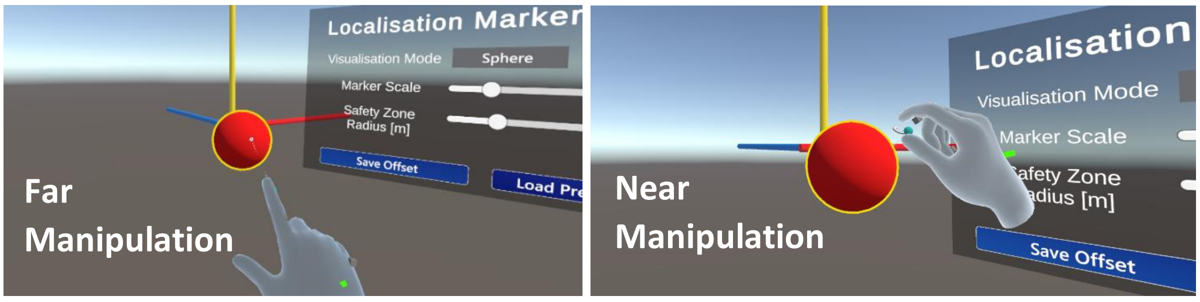
Ros Bridge Server IP
192.168.56.102
Ros Bridge Server Port
9090
Connected to RosBridge at:
ws://192.168.56.102:9090

The HoloLens and Robot must be on the same local network to establish a connection.

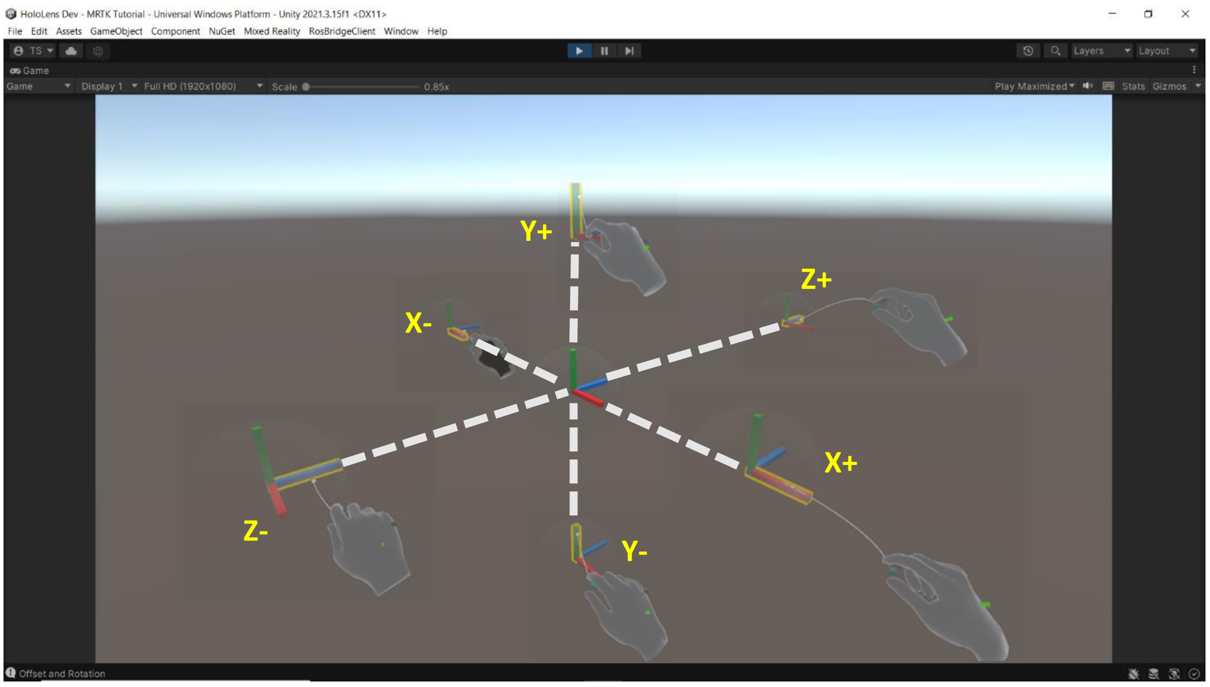
Localising the Robot

This subsystem implements the Microsoft QR code tracking sample with a custom marker that has two main components. The first is a three axis object that will always try to align its origin with the top left corner of any QR code. The second is a virtual marker that is moved to the robot's origin, which will be used to establish a coordinate frame transformation between the world coordinate frames of the HoloLens and robot.

There are four methods of moving this virtual marker to the desired location. The first method uses gesture recognition and hand tracking to move the virtual marker in 3D space. This is done by setting the Visualisation Mode on the custom marker's GUI to **Sphere** then either pointing a hand ray at the virtual marker or moving a hand within grabbing distance and performing a pinch gesture (shown below), at which point the marker can be moved freely.



The second method is similar to the first with the Visualisation Mode set to **Coord Frame** this time. Using gesture recognition and hand tracking, the virtual marker can be moved on one axis at a time as shown below.


HoloLens Dev - MRTK Tutorial - Universal Windows Platform - Unity 2021.3.15f1 <DX11>
File Edit Assets GameObject Component NuGet Mixed Reality RosBridgeClient Window Help
Game
Display 1
Full HD (1920x1080)
Scale 0.85x
Play Maximized
Stats Gizmos
Y+
Z+
X-
X+
Z-
Y-
Offset and Rotation

By pressing the **Show Complex Controls** button on the GUI, the third method can be accessed. This method is currently the only method of rotating the virtual marker after it has been instantiated.

The virtual marker is moved by entering the position and rotating offsets into the input fields or by incrementing/decrementing the offsets for each individual axis using the buttons on either side of the input field. The increment/decrement amounts (**Step**) can be selected from dropdowns or incremented/ decremented using the buttons on either side.

Incrementing/decrementing the virtual marker's position offset on a given axis moves the virtual marker on that axis relative to the direction it's facing. Or in other words, it moves with respect to its local coordinate frame as shown in the image above.

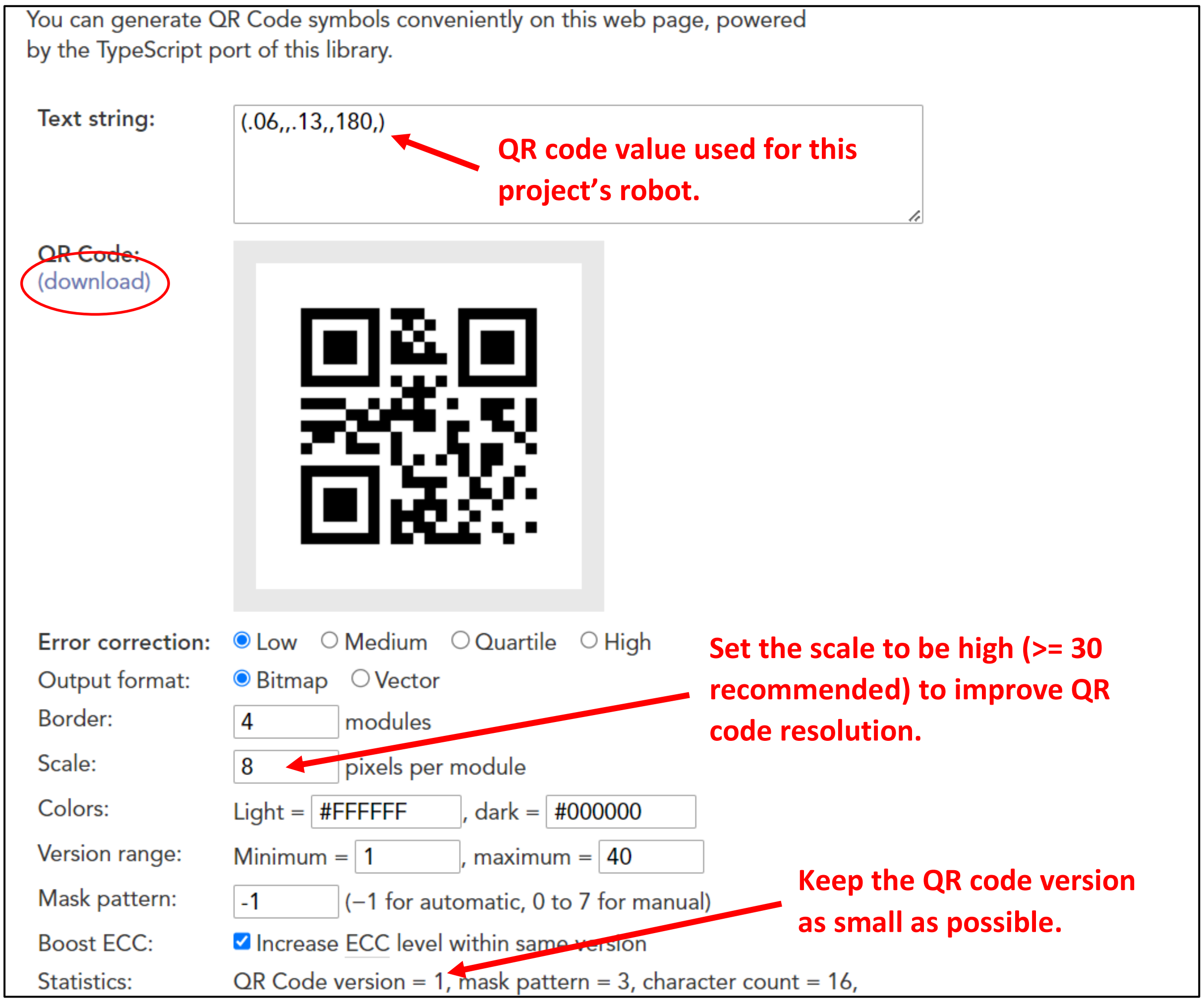
Incrementing/decrementing the virtual marker's rotation offset on a given axis works slightly differently. The x and z rotation axes are the HoloLens' world coordinate axes, which was done based on the assumption that the robot will be driven on a flat surface. The y axis rotation is the virtual marker's rotation relative to the QR code axis object.

The fourth method is performed automatically when the custom marker is instantiated by reading the value of the QR code, which can be in one of two formats:

1. `(x,y,z)` → This offsets the virtual marker from the axis object.
2. `(x,y,z,rx,ry,rz)` → This rotates the virtual marker then performs the offset from 1.

The formats above can be put into a QR code generator such as the one shown in the image below. Ideally, the QR code version should be kept as low as possible as the increased complexity from higher versions makes the marker harder to track. The complexity increases when the QR code's value has more data. To reduce data, a few shorthand's were introduced.

1. All zero values can be omitted but the number of commas must remain the same. For example, (1,0,2) can be shortened to (1,,2).
2. Values between -1 and 1 (not inclusive) can have the leading zero omitted. (0.1,0,-0.2) can be shortened to (.1,,-.2) for example.



You can generate QR Code symbols conveniently on this web page, powered by the TypeScript port of this library.

Text string: (.06,,.13,,180,)

QR code value used for this project's robot.

QR Code: (download)

Error correction: [✓ Low] Medium Quartile High

Output format: [✓ Bitmap] Vector

Border: 4 modules

Scale: 8 pixels per module

Set the scale to be high (>= 30 recommended) to improve QR code resolution.

Colors: Light = #FFFFFF, dark = #000000

Version range: Minimum = 1, maximum = 40

Mask pattern: -1 (−1 for automatic, 0 to 7 for manual)

Boost ECC: [✓] Increase ECC level within same version

Statistics: QR Code version = 1, mask pattern = 3, character count = 16,

Keep the QR code version as small as possible.

Source: https://www.nayuki.io/page/qr-code-generator-library

To add the QR code to the robot, press the **(download)** button shown above. It's easier if this webpage is opened on the robot: otherwise, transfer the downloaded image to the robot, **right click** the image and click **Set as Wallpaper**. Finally, open a Terminal window and enter the following command:

```
gsettings set org.gnome.desktop.background picture-options 'scaled'
```

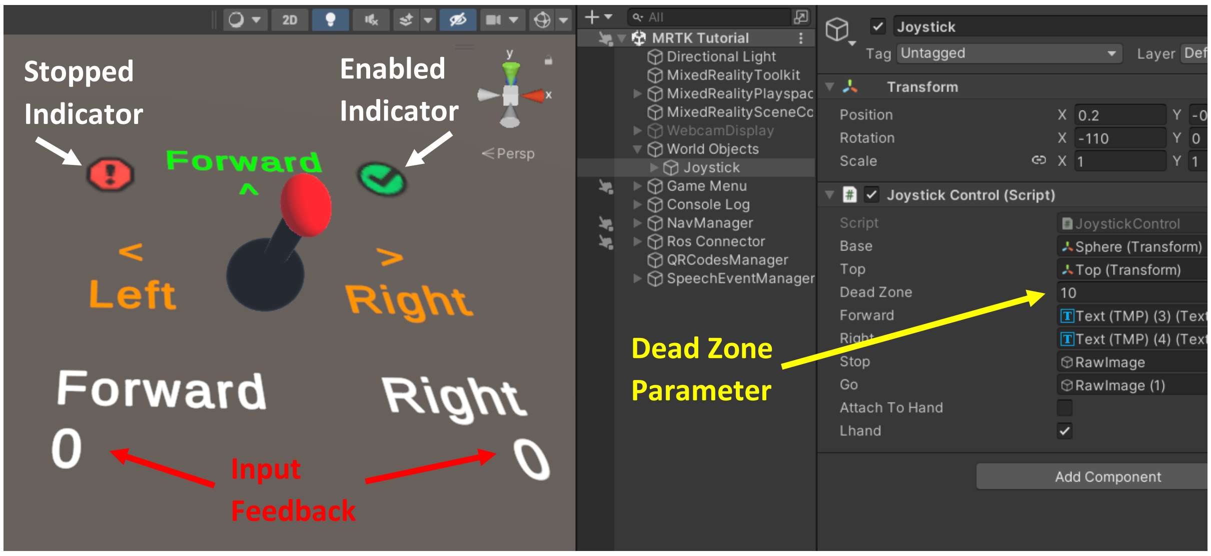
When a desired position offset and rotation offset is achieved with any of the previous methods, the **Save Offset** button on the GUI can be used to save this offset, which can be loaded using **Load Previous Offset** button if the position changes or the application is restarted.

Note that this offset will be lost if the application is updated.

Manual Control

IMPORTANT: To stop the robot's motion using the voice command, "Stop". To reenable the joystick controller, use the voice command, "Command", followed by the voice command, "Restart".

This is performed solely using the joystick in this application, which has three ways of being used.

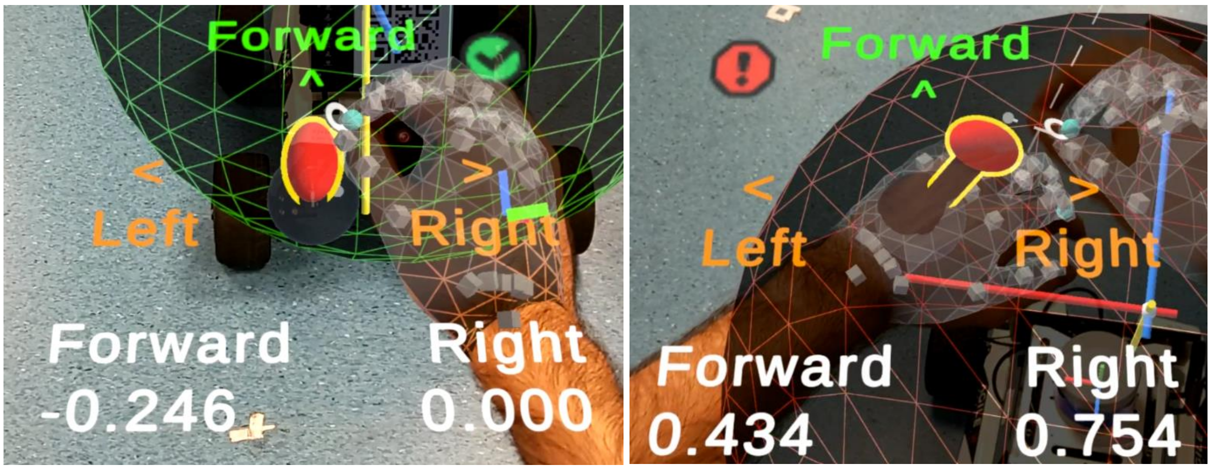


In all three methods, this controller is used by performing a pinch gesture on the red top of the joystick, which can then be moved using hand tracking. The forward direction is also constantly aligned to the direction the user's head is pointing.

The three methods differ by how the joystick itself is positioned and the number of hands used. By toggling off the **Joystick Controller Tracks Hand** option in the Menu, the first method has the joystick remaining as a stationary object that is manipulated using either of the user's hands.

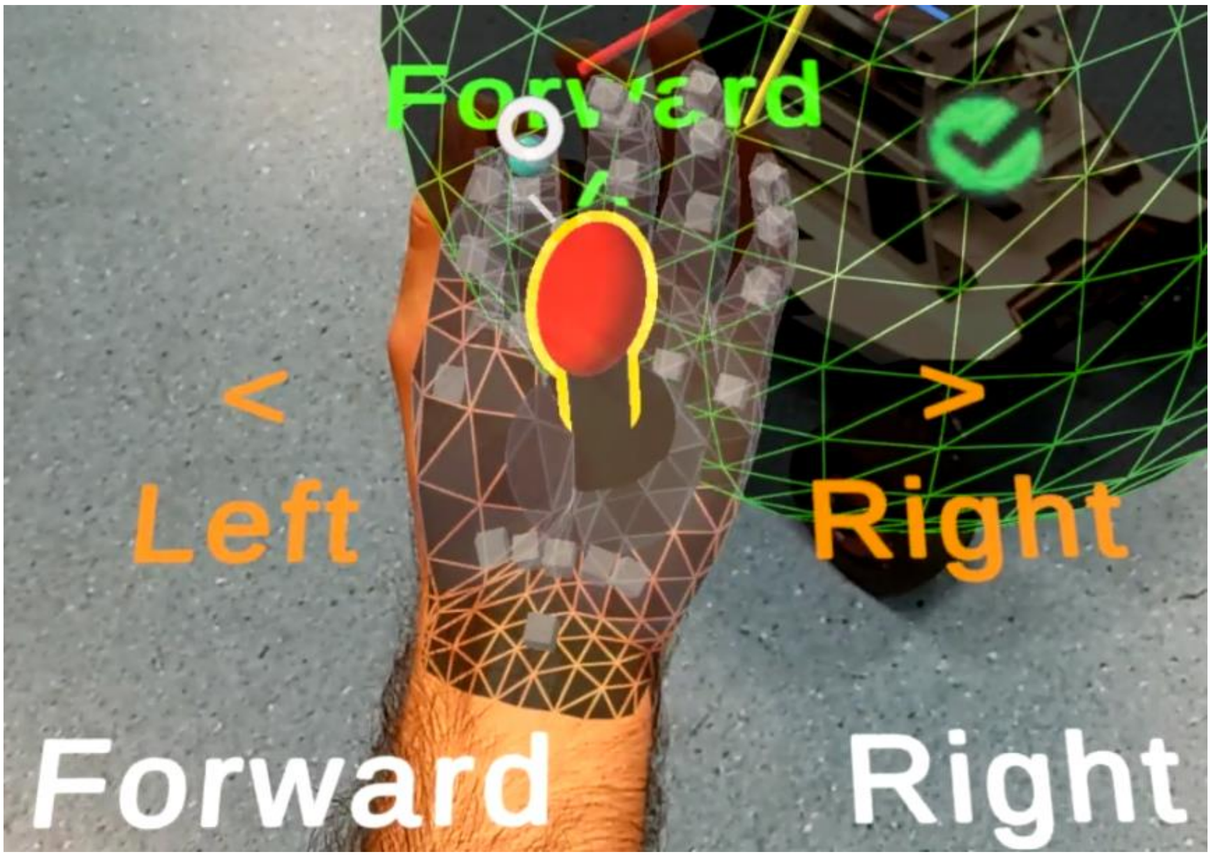
By toggling on the **Joystick Controller Tracks Hand** option in the Menu, the joystick's position will be constrained to one of the user's hands, allowing them to move about and manipulate the joystick with their other hand.

The user can select which hand the controller tracks to by clicking the toggle next to the **Joystick Controller Tracks Hand** option.



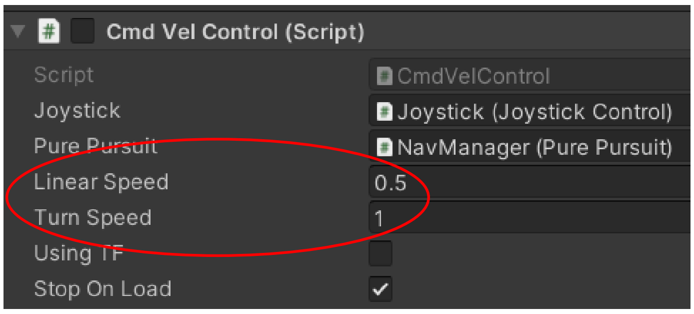
With this same setup, the user can also control the robot with just one hand. By facing the user's palm with the joystick attached towards their head and performing a pinch gesture, the joystick can be manipulated when the user rotates their wrist.

Be mindful that performing a pinch gesture while the user is looking at their wrist can cause the HoloLens' menu to open.



Adjusting the robot's maximum speed and turn rate is done by changing the public **Linear Speed** and **Turn Rate** variables in the **CmdVelControl** script.

Note that if these values are too low, the robot's motion won't be very fluid.



Cmd Vel Control (Script)

Script	CmdVelControl
Joystick	Joystick (Joystick Control)
Pure Pursuit	NavManager (Pure Pursuit)
Linear Speed	0.5
Turn Speed	1
Using TF	☐
Stop On Load	✓

Voice Commands

There are two types of voice commands in this application, safe and unsafe. Unsafe voice commands have functionality such as enabling robot movement, which could cause someone harm if those commands are unintentionally heard by the HoloLens.

These voice commands will only have an effect if used after the voice command, "Command", which will give the user visual (as shown below) and auditory feedback when it takes effect.

Listening For Command...

Safe voice commands are those that have non-hazardous functionality or require immediate action once spoken (the "Stop" command in particular). No prerequisite is required for these voice commands, a full list of which is shown in the table below. The full list of voice commands can also be found in the application by toggling the **Show Voice Commands UI** option in the Menu.

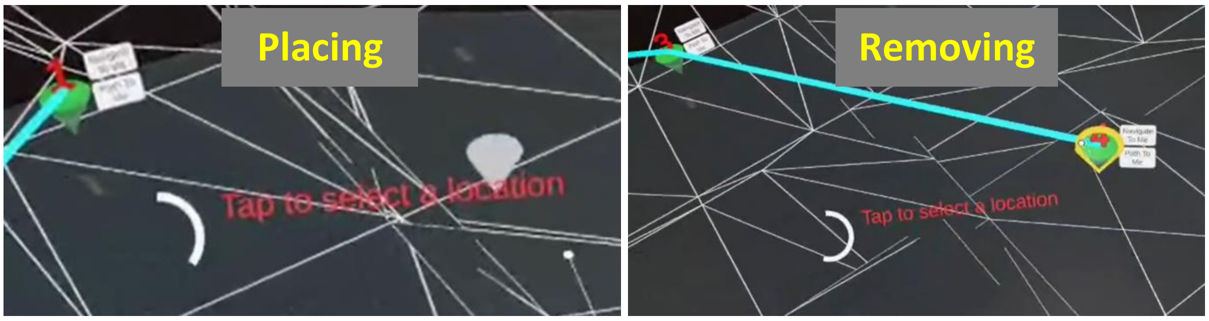
Safe Commands	**Function**		**Unsafe Commands**	**Function**
Stop	Floods the ROS messages with stop messages and ignores any other inputs.		Restart	Allows ROS messages to deliver inputs to the robot.
Command	Allows **Unsafe Commands** to take effect.		Follow Me	Enables a feature of the navigation subsystem that commands the robot to follow the user.
Menu	Toggles the Menu GUI.			
Select	Generic Mixed Reality Toolkit command that clicks focused UI elements.			
Toggle Navigation	Toggles the Navigation subsystem (detailed below).			
Clear Markers	Explained above in Menu.			

Navigation

Can be toggled on and off in the Menu or with the voice command, "Toggle Navigation". Doing so will enable the HoloLens' Spatial Awareness and the functionality to place navigation markers on the Spatial Awareness layer.

Markers can be placed by doing a pinch and hold gesture for one second while looking at a surface. The marker will be positioned at the point where the user's gaze meets a surface using eye tracking.

During the pinch and hold gesture, a ghost marker will appear indicating the position the marker will be placed and a radial indicator will inform the user of how long to hold the pinch gesture for. This operation can be cancelled anytime during the hold by the user separating their fingers.



Markers can also be removed by doing a pinch and hold gesture while pointing a hand ray at a marker. There will be no ghost marker, but the marker will be outlined, the radial indicator will still be present (as shown in the image above), and the operation can still be cancelled at any time.

Each of these navigation markers has two buttons that can be clicked by pointing a hand ray at them and performing a pinch gesture. The **Navigate To Me** button instructs the robot to pure pursuit to that marker, while the **Path To Me** button instructs the robot to pure pursuit to each successive marker in numerical order (indicated by the number above each marker) until it reaches the selected marker.

An additional "navigation" feature that uses the pure pursuit system is an option to follow the user as they move around. This is activated using the voice commands, "Command" and, "Follow Me" (unsafe command), and doesn't require the navigation system to active.

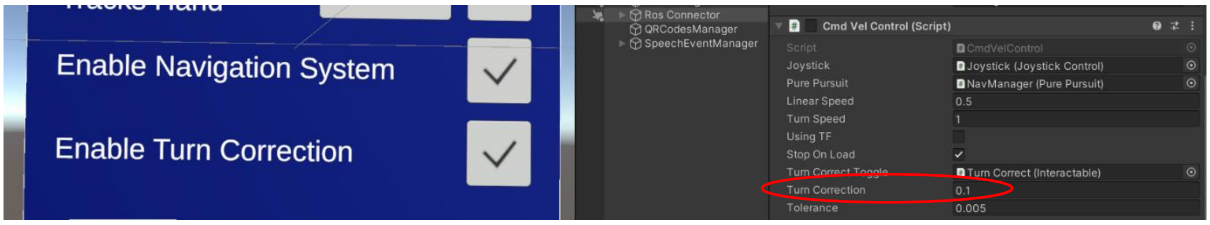
If the **Stop** command is invoked at any point during the operation of the pure pursuit system (due to voice command, safety zone, etc), the system will cancel any further operations with no option to resume them. They must be manually restarted by clicking the button's again or saying the voice commands.

It is also worth mentioned that using the joystick will override movement commands by the pure pursuit system but will **NOT** cancel its operations. They will resume when the user stops interacting with the joystick unless a **Stop** command has been invoked before then.

The solution for placing navigation markers using the HoloLens' spatial awareness system was adapted from this article by Joost van Schaik. Included was this GitHub demo project containing scripts used in this project.

Tracking

The robot's odometry tends to be rather inaccurate especially when making turns. Some imprecise testing found that the robot's IMU odometry has an undershoot drift of approximately 0.1 degrees per degree turned when turning on the spot. The **CmdVelControl** script includes an option to apply this value to the robot's rotation feedback and tune it through the editor. This option can also be toggled during runtime using the **Enable Turn Correction** toggle in the Menu.



Enable Navigation System ✓

Enable Turn Correction ✓

Ros Connector
QRCodesManager
SpeechEventManager

Cmd Vel Control (Script)

Script	CmdVelControl
Joystick	Joystick (Joystick Control)
Pure Pursuit	NavManager (Pure Pursuit)
Linear Speed	0.5
Turn Speed	1
Using TF	
Stop On Load	✓
Turn Correct Toggle	Turn Correct (Interactable)
Turn Correction	0.1
Tolerance	0.005

An attempt was made to use the robot's navigation stack by subscribing to the "/tf" ROS Topic, but worse results were achieved. The implementation is still in the project and can be enabled by setting the **Using TF** Boolean in the **CmdVelControl** script to **true**. The robot's navigation stack is enabled by opening a new Terminal window on the robot and running the following command:

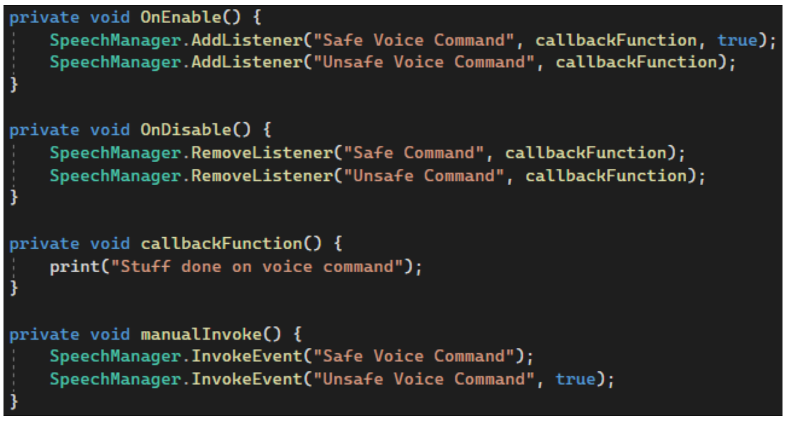
```
roslaunch qcr_agilex_payload slam_nav.launch
```

Development Guide

This section will provide details on how to extend upon currently implemented features in this project.

Voice Commands

The **SpeechManager** script provides a handy interface for assigning any voice command to a callback function. A typical implementation using this is shown below.

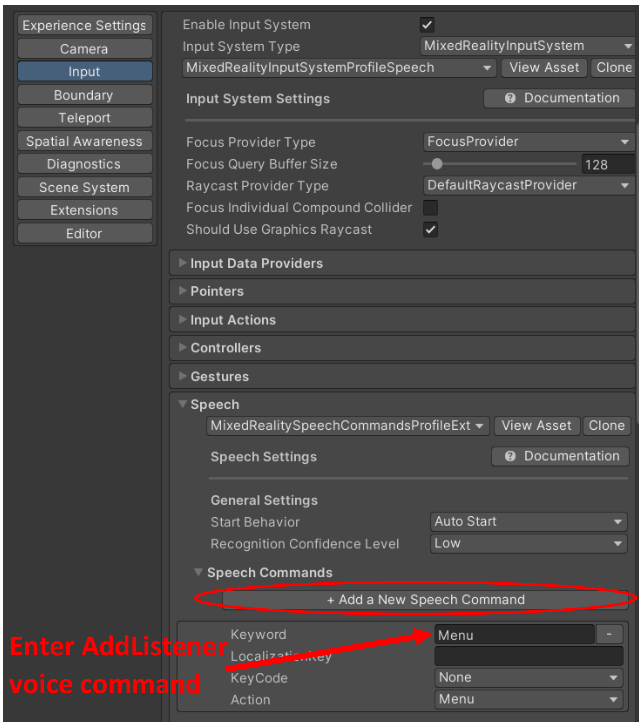


```
private void OnEnable() {
    SpeechManager.AddListener("Safe Voice Command", callbackFunction, true);
    SpeechManager.AddListener("Unsafe Voice Command", callbackFunction);
}

private void OnDisable() {
    SpeechManager.RemoveListener("Safe Command", callbackFunction);
    SpeechManager.RemoveListener("Unsafe Command", callbackFunction);
}

private void callbackFunction() {
    print("Stuff done on voice command");
}

private void manualInvoke() {
    SpeechManager.InvokeEvent("Safe Voice Command");
    SpeechManager.InvokeEvent("Unsafe Voice Command", true);
}
```

To assign a voice command to a callback function, the **AddListener** function is used. This function takes three parameters: the **voice command** as a String, the **callback function** and a Boolean set to **true** if the **voice command** is safe, otherwise it can be set to **false** or left blank. Multiple callback functions can be assigned to a single voice command, and multiple voice commands can be assigned to a single callback function, as shown above.

To unassign a callback function from a voice command, the **RemoveListener** function is used. This function takes two parameters: the **voice command** as a String and the **callback function**.

To manually invoke the callback function(s) assigned to a voice command, the **InvokeEvent** function is used. This function takes two parameters: the **voice command** as a String and a Boolean which can be set to **false** or left blank if the **voice command** is safe, otherwise it must be set to **true**. This function can be called from any **MonoBehaviour** script.



Enable Input System: ✓
Input System Type: MixedRealityInputSystem
MixedRealityInputSystemProfileSpeech

Input System Settings

Focus Provider Type: FocusProvider
Focus Query Buffer Size: 128
Raycast Provider Type: DefaultRaycastProvider
Focus Individual Compound Collider: ☐
Should Use Graphics Raycast: ✓

Speech
MixedRealitySpeechCommandsProfileExt

Speech Settings

General Settings
Start Behavior: Auto Start
Recognition Confidence Level: Low

Speech Commands
Keyword: Menu
Localizati[illegible]Key:
KeyCode: None
Action: Menu

The other required step is to add the voice command to Mixed Reality Toolkit's Input Speech profile. Once on the screen shown above, click the **+ Add a New Speech Command** button and enter the voice command from the **AddListener** function call(s) into the **Keyword** field.

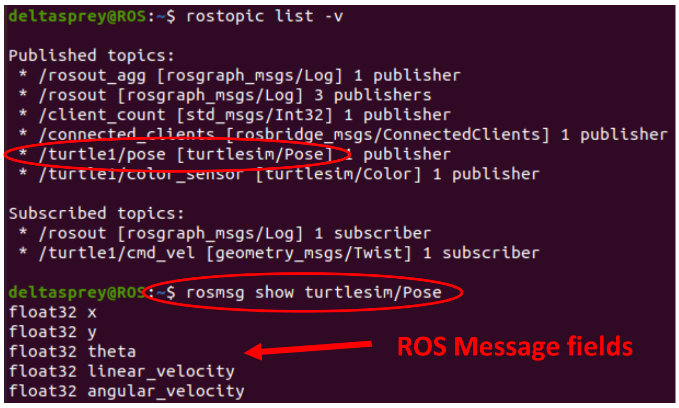
Custom ROS Subscribers

Subscribing to a ROS Topic requires reading its ROS Message. For example, the ROS Topic, "/turtle1/pose" has the ROS Message, "turtlesim/Pose". This can be found by running the following commands:

```
rostopic list -v
```

OR

```
rostopic type <ROS Topic>
```

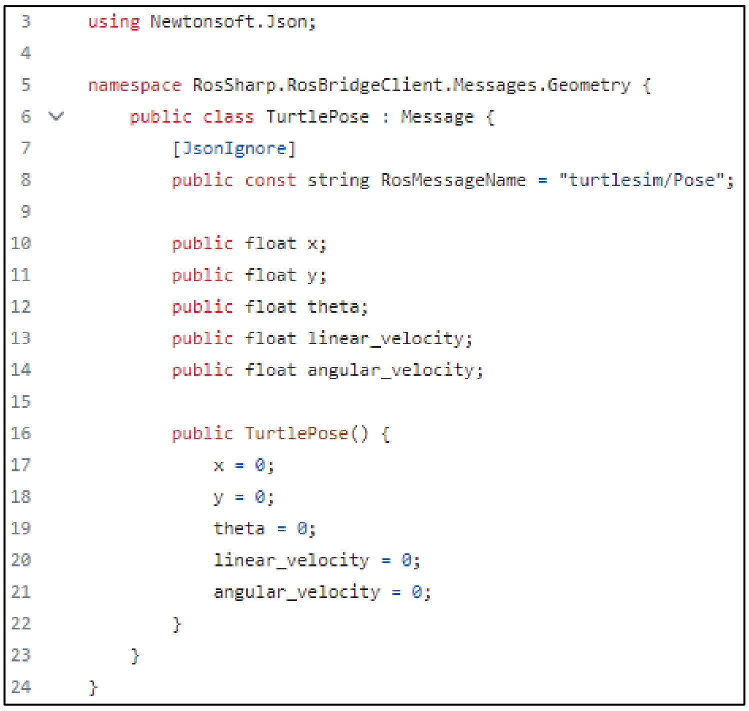


```
deltasprey@ROS:~$ rostopic list -v

Published topics:
 * /rosout_agg [rosgraph_msgs/Log] 1 publisher
 * /rosout [rosgraph_msgs/Log] 3 publishers
 * /client_count [std_msgs/Int32] 1 publisher
 * /connected_clients [rosbridge_msgs/ConnectedClients] 1 publisher
 * /turtle1/pose [turtlesim/Pose] 1 publisher
 * /turtle1/color_sensor [turtlesim/Color] 1 publisher

Subscribed topics:
 * /rosout [rosgraph_msgs/Log] 1 subscriber
 * /turtle1/cmd_vel [geometry_msgs/Twist] 1 subscriber

deltasprey@ROS:~$ rosmsg show turtlesim/Pose
float32 x
float32 y
float32 theta
float32 linear_velocity
float32 angular_velocity
```

To read from a ROS Message, the subscriber must have fields for each field in the Message. To see what the fields are, run the following command (example shown in image above):

```
rosmsg show <ROS Message>
```

As seen in the image above, the ROS Message, "turtlesim/Pose" has five fields of type **float32**. To subscribe to this Message using the ROS# package, the following script is used:



```
using Newtonsoft.Json;

namespace RosSharp.RosBridgeClient.Messages.Geometry {
    public class TurtlePose : Message {
        [JsonIgnore]
        public const string RosMessageName = "turtlesim/Pose";

        public float x;
        public float y;
        public float theta;
        public float linear_velocity;
        public float angular_velocity;

        public TurtlePose() {
            x = 0;
            y = 0;
            theta = 0;
            linear_velocity = 0;
            angular_velocity = 0;
        }
    }
}
```

For convention, the **namespace** should be in the **Messages** or an extended namespace of it in **RosSharp.RosBridgeClient**. For example, **Messages.Geometry**.

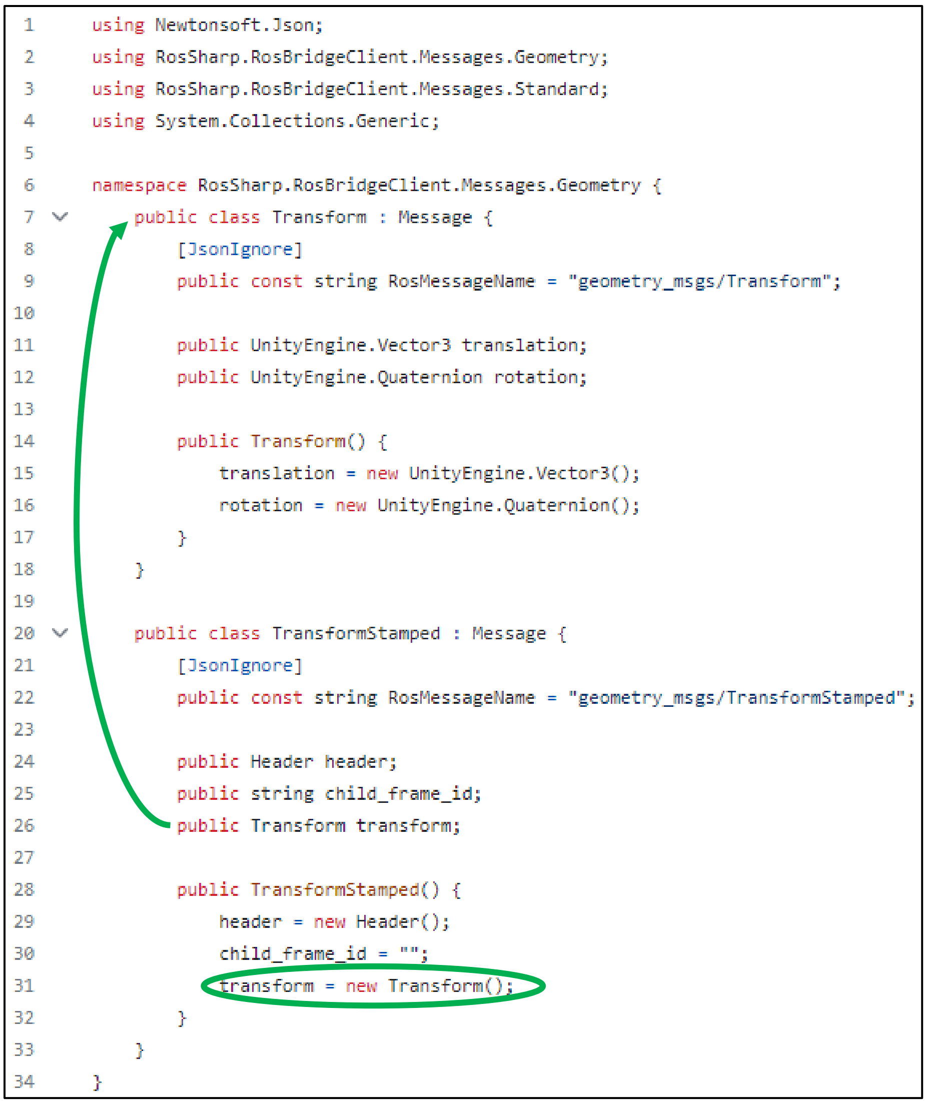
The class must be an extension of the **Message** class.

The **ROSMessageName** must be identical to the ROS Message name of the ROS Topic being subscribed to.

The same number and types of fields in the ROS Message must be declared.

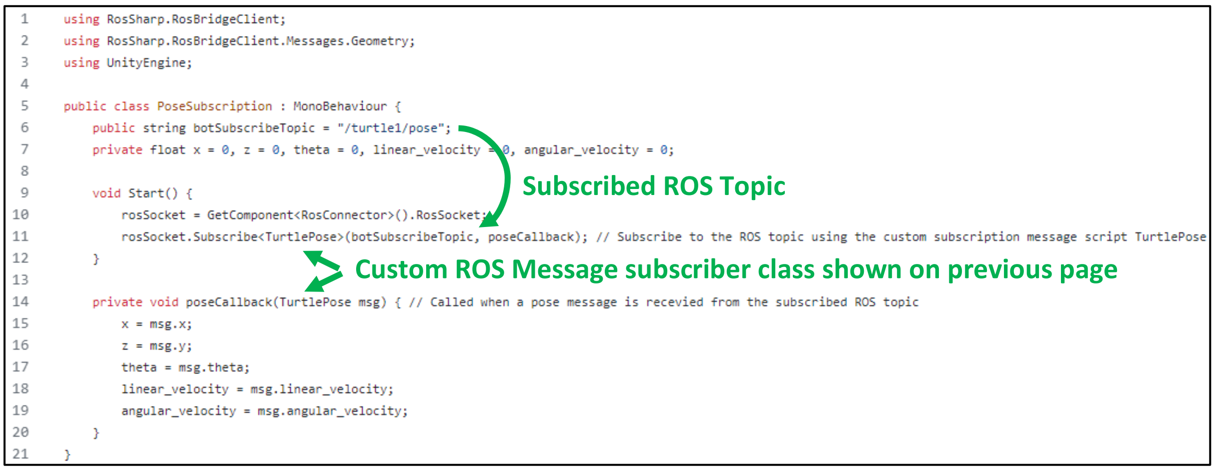
Initialisation of these fields is required and done in the class constructor.

For more complicated ROS Topics, many of their ROS Messages will be combinations of other ROS Messages that may already be implemented in the ROS# **Messages** namespace. The example below shows a custom subscriber used in this project, which creates a subscriber for the ROS Message, "geometry_msgs/Transform". Below it, there is a subscriber for the ROS Message, "geometry_msgs/TransformStamped", which extends the **Transform** subscriber.



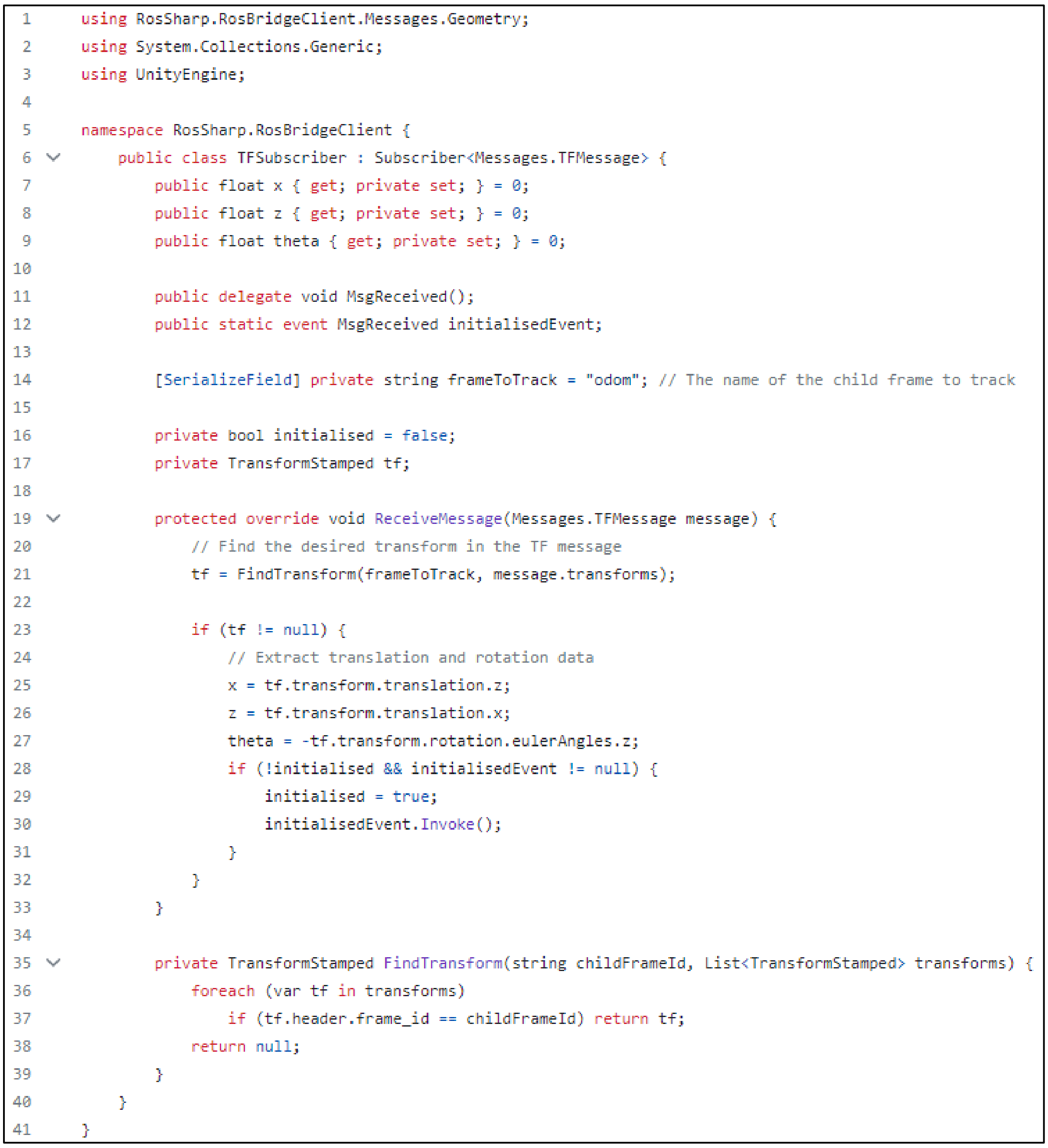
```
using Newtonsoft.Json;
using RosSharp.RosBridgeClient.Messages.Geometry;
using RosSharp.RosBridgeClient.Messages.Standard;
using System.Collections.Generic;

namespace RosSharp.RosBridgeClient.Messages.Geometry {
    public class Transform : Message {
        [JsonIgnore]
        public const string RosMessageName = "geometry_msgs/Transform";

        public UnityEngine.Vector3 translation;
        public UnityEngine.Quaternion rotation;

        public Transform() {
            translation = new UnityEngine.Vector3();
            rotation = new UnityEngine.Quaternion();
        }
    }

    public class TransformStamped : Message {
        [JsonIgnore]
        public const string RosMessageName = "geometry_msgs/TransformStamped";

        public Header header;
        public string child_frame_id;
        public Transform transform;

        public TransformStamped() {
            header = new Header();
            child_frame_id = "";
            transform = new Transform();
        }
    }
}
```

Because these subscribers are declared in a namespace, they don’t have to be added to the Scene to function. To make them function, the **RosSocket** from the **RosConnector** script must explicitly subscribe to the ROS Topic that uses the implemented ROS Message subscriber (using a script in the scene). An example is shown in the Rosbridge demonstration project (bottom of linked section).



```
using RosSharp.RosBridgeClient;
using RosSharp.RosBridgeClient.Messages.Geometry;
using UnityEngine;

public class PoseSubscription : MonoBehaviour {
    public string botSubscribeTopic = "/turtle1/pose";
    private float x = 0, z = 0, theta = 0, linear_velocity = 0, angular_velocity = 0;

    void Start() {
        rosSocket = GetComponent<RosConnector>().RosSocket;
        rosSocket.Subscribe<TurtlePose>(botSubscribeTopic, poseCallback); // Subscribe to the ROS topic using the custom subscription message script TurtlePose
    }

    private void poseCallback(TurtlePose msg) { // Called when a pose message is recevied from the subscribed ROS topic
        x = msg.x;
        z = msg.y;
        theta = msg.theta;
        linear_velocity = msg.linear_velocity;
        angular_velocity = msg.angular_velocity;
    }
}
```

Another method of making the custom subscriber's function is creating a script in the scene with a class that extends the **Subscriber** class in the **RosSharp.RosBridgeClient** namespace. Receiving messages is done by overriding the **ReceiveMessage** function as shown in the script below.



```
using RosSharp.RosBridgeClient.Messages.Geometry;
using System.Collections.Generic;
using UnityEngine;

namespace RosSharp.RosBridgeClient {
    public class TFSubscriber : Subscriber<Messages.TFMessage> {
        public float x { get; private set; } = 0;
        public float z { get; private set; } = 0;
        public float theta { get; private set; } = 0;

        public delegate void MsgReceived();
        public static event MsgReceived initialisedEvent;

        [SerializeField] private string frameToTrack = "odom"; // The name of the child frame to track

        private bool initialised = false;
        private TransformStamped tf;

        protected override void ReceiveMessage(Messages.TFMessage message) {
            // Find the desired transform in the TF message
            tf = FindTransform(frameToTrack, message.transforms);

            if (tf != null) {
                // Extract translation and rotation data
                x = tf.transform.translation.z;
                z = tf.transform.translation.x;
                theta = -tf.transform.rotation.eulerAngles.z;
                if (!initialised && initialisedEvent != null) {
                    initialised = true;
                    initialisedEvent.Invoke();
                }
            }
        }

        private TransformStamped FindTransform(string childFrameId, List<TransformStamped> transforms) {
            foreach (var tf in transforms)
                if (tf.header.frame_id == childFrameId) return tf;
            return null;
        }
    }
}
```

This script subscribes to the "/tf" ROS Topic used in this project which has the ROS Message, "tf2_msgs/TFMessage". This message comprises a list of the **TransformStamped** custom ROS Message subscriber shown in the previous page.

Troubleshooting

Deploy Failed on Visual Studio

This can occur after Visual Studio is updated, which causes it to lose the Machine Name configuration property. To fix, go to **Project → Properties → Configuration Properties → Debugging** and add the IP of the HoloLens back into the **Machine Name** field.

This can also occur if the Unity project is built on a previous version of Visual Studio and then a Deploy is attempted on a later version. To fix this, navigate to the folder where the Unity project was built and **delete** all the files there. **Rebuild** the Unity project and it should now be deployable. Make sure to have also followed the steps in the previous paragraph.

Visual Studio NuGet Package Restore Failed Unable to Find Version

Visual Studio doesn't check the internet for NuGet packages by default which causes errors if the packages aren't on your computer. To fix this in Visual Studio, go to **Tools → Options → NuGet Package Manager → Packages Sources**. Click the green plus button in the top right corner then enter the following into the fields.

Name: nuget.org

Source: https://www.nuget.org/api/v2/

Press **Update**, then in **NuGet Package Manager** click on **General → Clear All NuGet Storage**. Press **Ok** and the project should now compile without errors.

Unity WSATestCertificate is Expired

Go to **Edit → Project Settings → Player → Publishing Settings**. Under **Certificates** click the button above Create and delete the WSATestCertificate.pfx file. Close the file explorer window and then click **Create**. Leave everything at default and create a new certificate.

ROS# Project Newtonsoft.Json.dll Not Found Error

This can sometimes be fixed by closing and reopening the Unity project. When this isn't the case, try installing NuGet by following the steps above.

Unity Event Callbacks Not Fully Executing

The most surefire way to solve this is to change the value of a private Boolean variable, which is checked by a conditional statement in **Update()**. The formerly blocking code will be successfully executed inside this conditional statement. Be sure to reset the Boolean variable at the end.