Unity Transform Controller

Purpose

The aim of this project is to provide an alternate means of manipulating transforms within Unity within the editor, and at runtime. This tool will help users design levels in Unity as it enables manipulation of position and rotation concurrently in real time. While the same could be done with a keyboard, it is more intuitive to rotate an object by hand, and raise or lower a physical object to raise or lower the target transform. Furthermore, having the tangible user interface (TUI) eliminates the need for more key bindings on the keyboard, reducing cognitive load and user error (for example, miss-clicks). In like manner, the discovery and recall of this transform translation functionality within the app is anticipated to be more accessible given the effort to create the tool, and its physical presence.

In addition to benefits of using this tool in the editor, the TUI could be used at runtime to control a player. For example, a space exploration game would benefit from the increased freedom of movement, and the immersion of input offered by the TUI. Finally, this TUI may benefit users who are not yet familiar with transformations in 3D as they can apply their existing knowledge of manipulating real-world objects and view how the transform is modified within Unity. It follows that this TUI may be a useful learning tool.

List of Required Tools and Materials

The following tools and materials were used to create this prototype:

- Scissors and/or box cutter
- Cardboard
- 3 Popsicle sticks
- Hot glue gun
- Electrical tape
- 2 elastic bands
- One egg carton.

The following software was used:
- Unity
- Arduino IDE

The following libraries were used:
- Ardity
- i2cdevlib

Two cardboard cut layouts are available in the project repository [8] to support replication of the shroud.

System Detail

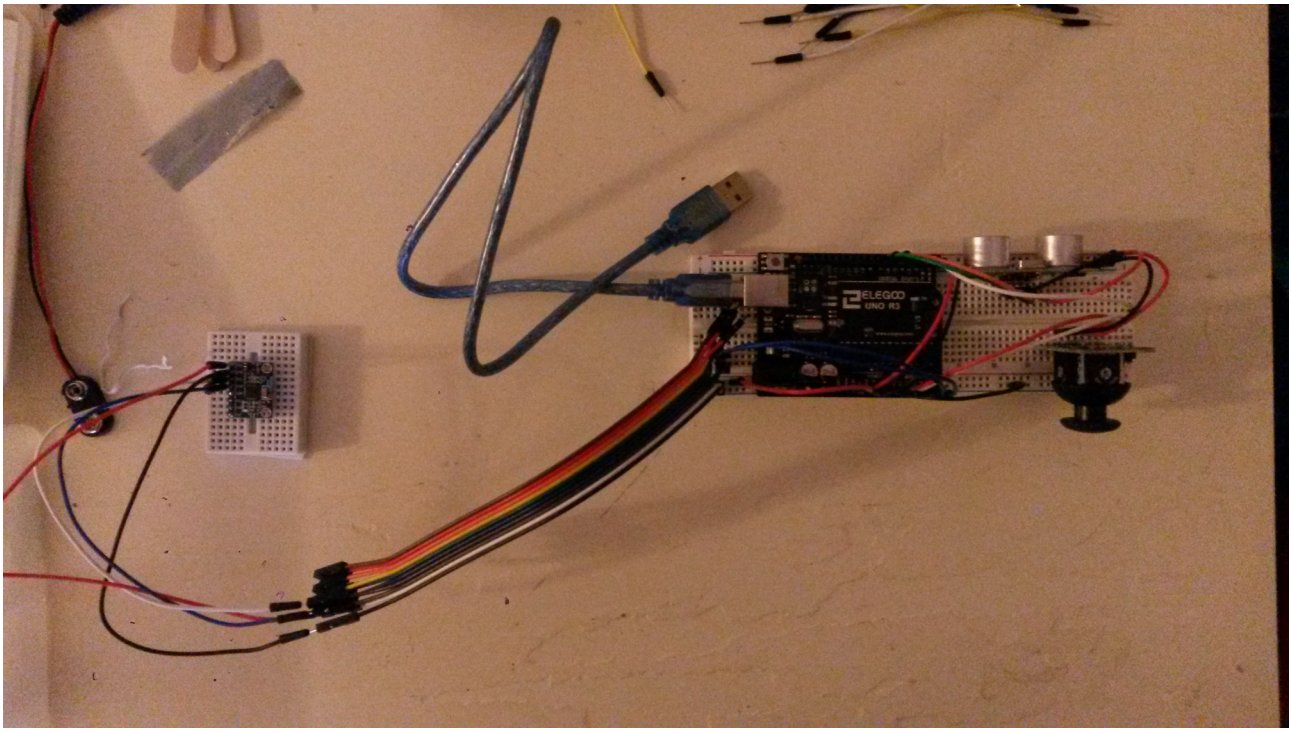


Figure 1: Circuit Implementation

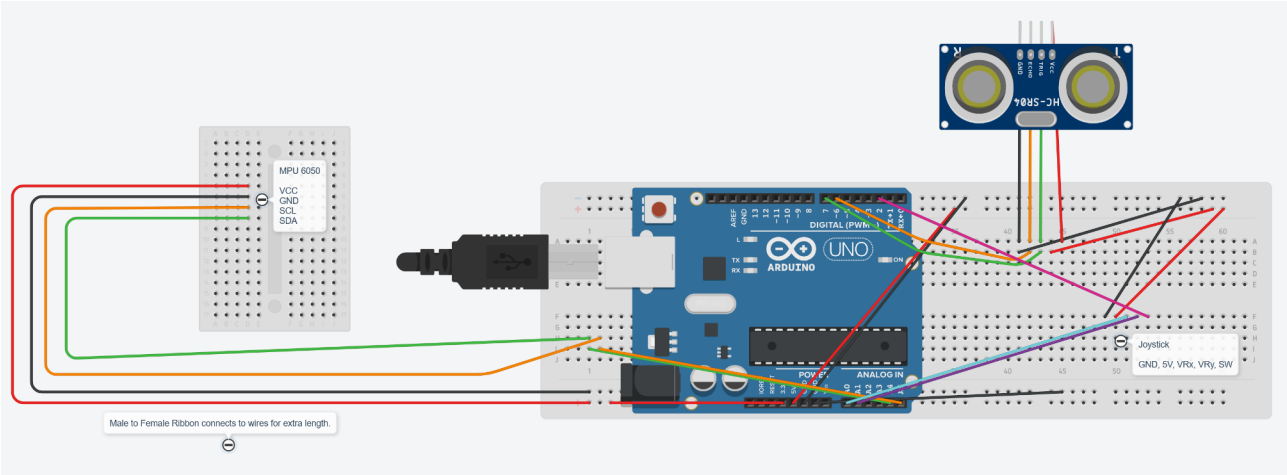


Figure 2: Circuit Design

As shown above (figure 1, and figure 2) there are three main components in addition to the Arduino Elegoo Uno R3: an MPU6050 motion processing unit (figure 3), a generic analogue joystick (figure 4), and an HC-SRO4 ultrasonic sensor. Figure 2 was created with Tinkercad, which doesn't have MPU 6050 or joysticks as components, so the ports are written in a note in the order which the wires plug into them (top-bottom, left-right). Also, the circuit design uses the convention that voltage always runs through red wires, and black wires always run to ground. It is helpful to tape the Arduino to the breadboard with electrical tape so that it doesn't shift around. After the circuit is assembled, you're ready to test it with some code.

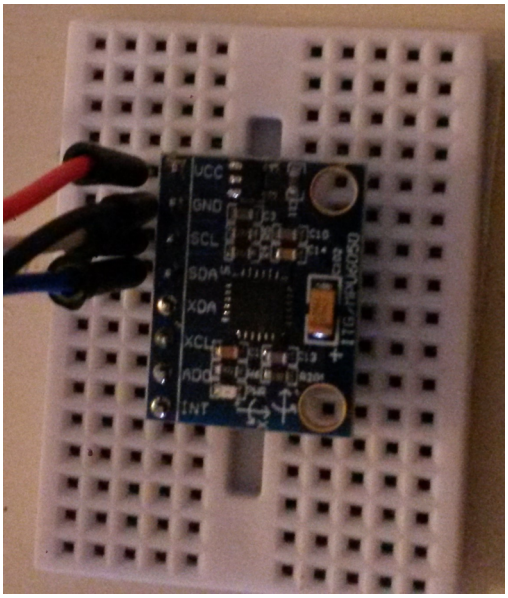


Figure 3: MPU 6050

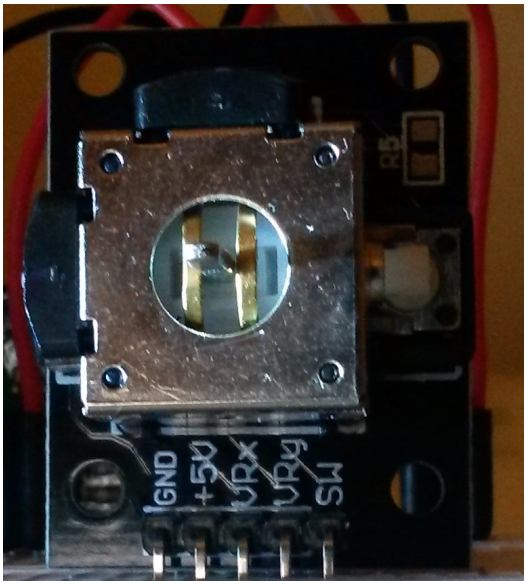


Figure 4: Joystick without thumb pad

To get rotation data from the MPU 6050, the i2cdevlib library collection [1] was used. Using the Arduino IDE, you need to import subsets of the library. By using the Sketch drop-down menu and selecting “Include Library > Add .ZIP Library…” you can add the required i2cdev and MPU6050 libraries from the Arduino folder of the i2cdevlib library collection. From these libraries I extracted the Quaternion input handling, setup, and associated variables, from the MPU6050_DMP6 example file to a method in my TransformController.ino file. The example file can be found in the Arduino IDE through the “File” drop-down menu by clicking “Examples > MPU6050 > MPU6050_DMP6.” That concludes the setup on the Arduino side for the MPU6050.

Next, joystick input was solved by finding an example [2] on the Arduino project hub. This is a very basic code snippet, so it was unmodified. As with the Quaternion handling, I extracted the main loop for joystick input to a method in my TransformController.ino file along with the required variables and setup method. The joystick handling is complete on the Arduino side.

Finally, the ultrasonic sensor is handled simply in kind to the joystick. I found an example of usage [3] and extracted the main loop to a method along with the required variables to my TransformController.ino file. To increase the granularity of the reading I calculate distance from the sensor in millimeters rather than inches or centimeters. Since this prototype setup has limited range of motion resulting from short wires this seemed like the best choice.

Once the proof of concept for each sensor is in place, they should be tested together to ensure everything is playing nicely together, and that no circuits were messed up along the way. After the serial output was printing nicely, I pulled the Ardity [4] Unity project and got the serial output logging to the console there. I am using Unity 2020.3.26f1 LTS with this setup. I faced a few hiccups to get Unity reading the serial stream from the Arduino. With the help from the Ardity troubleshooting guide [5] I got it working. I needed to change my .NET target in Unity’s project settings from 4.x to 2.0 and back, as well as change my architecture target in the build settings from x86_x64 to x86 and back to x86_x64 to get it working. Another gotcha is that the com port and

baud rate in Ardity's SerialController component must match your Arduino's com port and baud settings.

At long last, my TUI input was printing in the Unity console. But there were some problems with dropped and null messages. The baud rate was too high. In the end, I needed to use 9600 for the baud rate with this setup in order to get reliable input. With that done, I formatted my serial stream to be output with comma separated values so I could split the string and parse the values in Unity. I did an inverse lerp between the deadzone and extends of the raw sensor values to control translation, and assigned that translation vector and quaternion from the sensor directly to the target transform component.

Upon testing I immediately realized that the axes of my joystick and MPU6050 were not corresponding the axes in Unity. The orientation of those components in my TUI needed some transformation. Given my limited hardware resources, I was unable to solve this problem by rewiring and rearranging the circuits. In the end, the issue was solved during the assignment of values in Unity after parsing the serial input string.

With my setup working at runtime, I moved on and did a web search to see about handling the TUI input outside runtime in the Unity editor ("editor-time"). I found a couple cool forum threads discussing how to hook into the update loop of the Unity editor and handle custom input events. [6, 7] I adapted the snippets to my use case and it worked like a charm. Just like that, runtime and editor-time input handling for my TUI was complete (for this iteration).

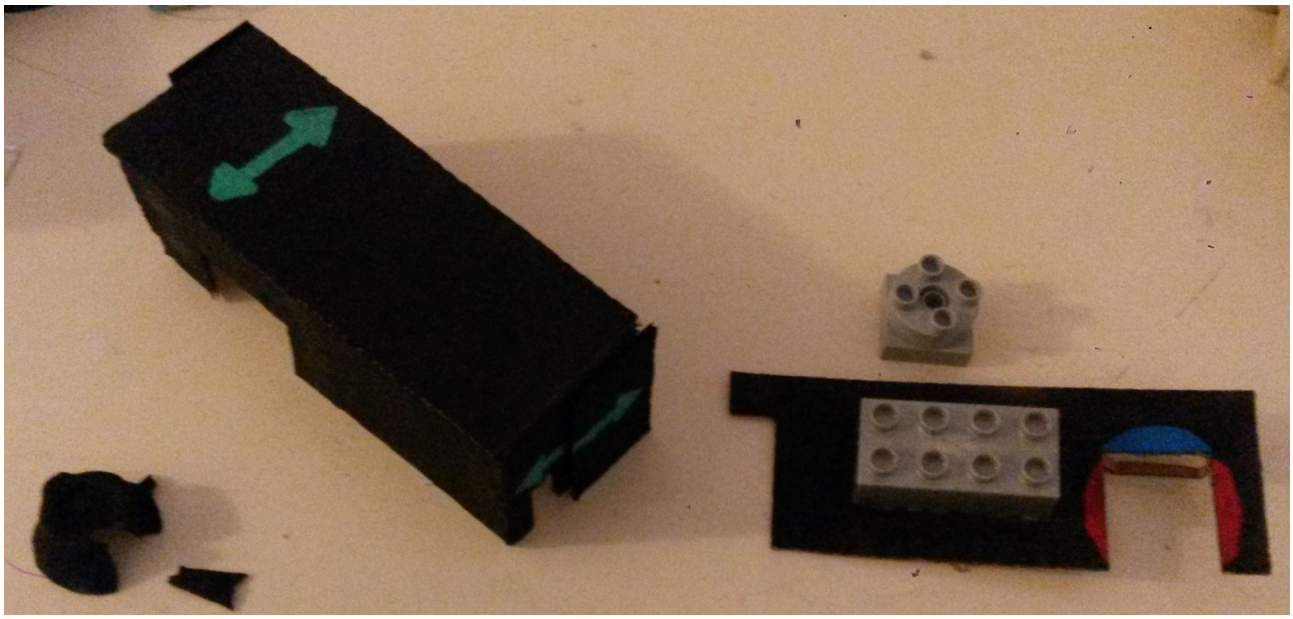
TUI Aesthetic and Tactile Design

To add affordance of the rotational input behaviour of the MPU6050 I attached popsicle sticks along the x, y, and z axes and painted them to match Unity's editor axes. Likewise, I painted around the joystick to correspond to the x and z axes which it maps. On the faces of the shroud, I painted bi-directional arrows in green to map to the y axis in Unity. To make the colour mapping of the axes pop out more, I painted the shroud black. In like-manner to the large shroud around the breadboard and Arduino, I added a smaller shroud to the MPU6050 to block the light and draw attention to the painted popsicle axes. I added a Duplo block mount with a pivot to the top of the main shroud, and another Duplo block to the underside of the breadboard mini so that the MPU6050 may be mounted to the top of the large controller. If there isn't need for a large range of motion, it may be preferable to have the rotation input and translation input bound together.

As this is a low-fidelity prototype, I felt that a boxy shape was acceptable. It provided a quick way to add and remove the shroud should any components need to be reconfigured. Given the shape of the breadboard and all the components in the TUI it was also prohibitively challenging to accomplish a more sleek design. I chose to make the shroud from cardboard as it is common and therefore creates an easy to reproduce prototype. The decision to use popsicle sticks was also grounded in accessibility, however, they are far more rigid. Popsicle sticks felt like a better choice for mounting to the rotation input device.

I would have preferred a different grip and orientation of the controller with the joystick mounted on it. However, I lacked the breadboards to get a more preferable orientation, as it felt more worthwhile to use the breadboard mini for the MPU6050.

Shroud and Axis Fabrication



Use the two cardboard cut layouts below to prepare the cardboard shroud for the breadboard. The original files are available in the project repo in the “./images” folder. [8]

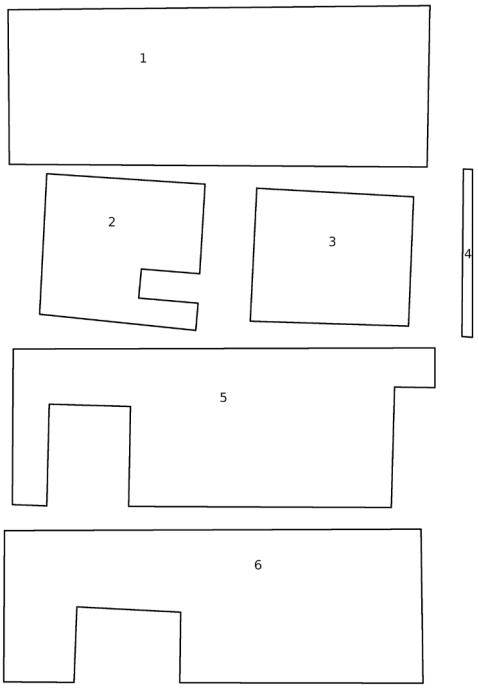


Figure 6: Cardboard cut layout 1

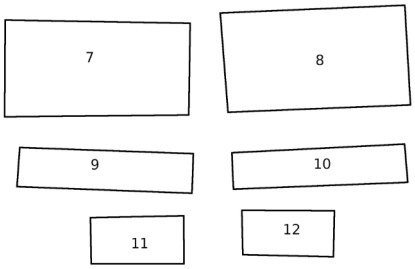


Figure 5: Cardboard cut layout 2

1. Print out the two cardboard cut layout images from the project repo [8] and cut along the lines to prepare the pieces.
2. Glue piece 1, 2, 3, and 6 together at right angles so that the cutout in 2 allows the USB cable from the Arduino through, and the cutout in 6 allows the ultrasonic sensor through.
3. Bend 9 and 10 into the corners between 1, 2, and 3 so that 5 can rest along the edges of 2, 3, 9 and 10 while butting up against 1. Glue 9 and 10 in.
4. Glue 11 and 12 together so they form a thicker rectangle. Then, glue them to the center of 1, between 9 and 10 to help support 5 when it rests on the edges of 2, 3, 9, 10, 11 and 12.
5. Glue 7 and 8 along the outside of 2 and 3 so that they come flush to the top edge of 1. This provides prevents 5 from slipping side-to-side off the inset edges you just made.
6. Glue 4 along the top of the joystick cutout in 5 so that it hangs underneath the joystick board to provide support.

Figure 7: Inside shroud

Figure 8: Joystick support

To create the shroud for the mini breadboard:

1. Cut out one cup of the egg carton.
2. Slice it to the desired height.
3. Make a small cut in one side of the cup which the wires fit through, saving the cutout to tape back over with electrical tape.
4. Paint the shroud black.
5. Tape the circular shroud over the MPU and wires, securing the wires to the shroud with a piece of electrical tape to prevent them from pulling out during use.

Figure 9: Mini shroud

To create the axes for the MPU:

1. Score a popsicle stick at one third of the length on both sides until it snaps evenly.
2. Hot glue the 1/3rd length stick to the center of one popsicle stick so their faces are attached.
3. Glue the breadboard mini in the center of the final popsicle stick.

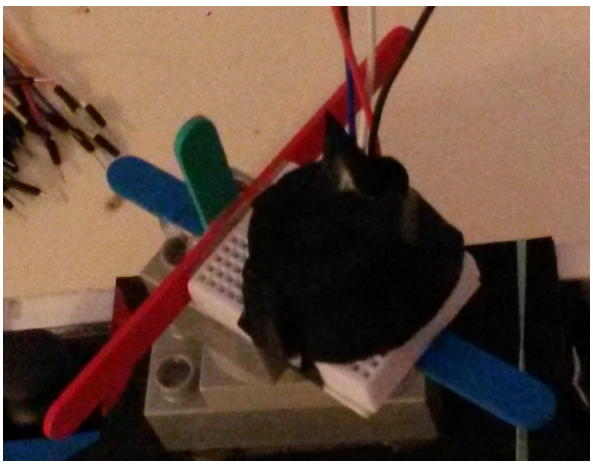


Figure 10: MPU Axes

4. Glue the base of the 1/3rd length popsicle stick and its perpendicular counterpart so that the small piece points up along the backside of the breadboard mini.

To match the axes of the popsicle sticks, joystick, and ultrasonic sensor to the axes in Unity, use red for the x-axis, blue for the z-axis, and green for the y-axis. This is left/right, forward/backward, and up/down respectively.

I had a Duplo piece that can pivot, so I hot glued a small Duplo block to the bottom of the breadboard mini, and a larger block to the top of the large shroud. This allows me to mount the MPU to the top of the large shroud with the option of having the pivot.

The shroud over the breadboard and Arduino should not be taped so that the components may be reconfigured. Also, this allows the unit to be disassembled and assembled for transportation. To keep the large shroud together, use two elastics around each end.

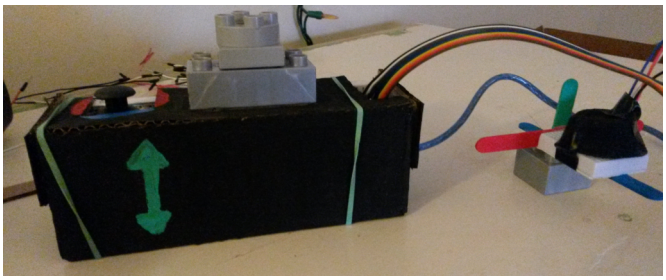


Figure 12: TUI Assembled

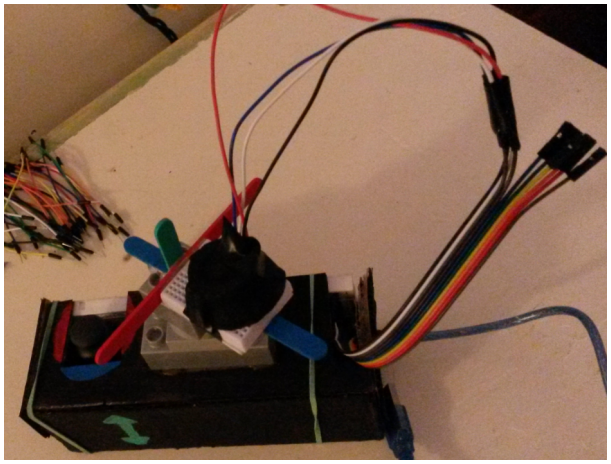


Figure 11: MPU Mounted with Duplo

Further Work

Should I continue to work on this project, I will integrate a wireless adapter to transmit input from the MPU6050. Being bound by wires is not ideal as it restricts the range of motion of the input devices. Likely a the NRF24L01 wireless RF transceiver module would be a good fit. [8] The design is lacking these components as I did not have them at my disposal. Likewise, the orientation and grip of the joystick could be improved with the addition of another breadboard mini. This would create need for a third breadboard mini to house the ultrasonic sensor, so that it can continue to be manipulated with the joystick hand. Both the joystick and ultrasonic sensor would benefit from a wireless setup as well. However, priority for wireless capability should be given to the MPU6050 for freedom in motion if resources continue to be limited.

References

[1] Rowberg, J. (2022, February 7). *I2C Device Library*. GitHub. https://github.com/jrowberg/i2cdevlib

[2] *How to Use a Joystick with Serial Monitor*. (n.d.). Arduino Project Hub. https://create.arduino.cc/projecthub/MisterBotBreak/how-to-use-a-joystick-with-serial-monitor-1f04f0

[3] *Arduino - Ultrasonic Sensor - Tutorialspoint*. (n.d.). Www.tutorialspoint.com. https://www.tutorialspoint.com/arduino/arduino_ultrasonic_sensor.htm

[4] *Ardity: Arduino + Unity over COM ports*. (n.d.). Ardity.dwilches.com. https://ardity.dwilches.com/

[5] *[4.6] EditorApplication.modifierKeysChanged, how to find out which key was pressed*. (n.d.). Unity Forum. Retrieved February 6, 2022, from https://forum.unity.com/threads/4-6-editorapplication-modifierkeyschanged-how-to-find-out-which-key-was-pressed.357367/

[6] *How do i get a callback every frame in edit mode - Unity Answers*. (n.d.). Answers.unity.com. Retrieved February 6, 2022, from https://answers.unity.com/questions/39313/how-do-i-get-a-callback-every-frame-in-edit-mode.html

[7] *Arduino Wireless Network with Multiple NRF24L01 Modules*. (2018, July 31). How to Mechatronics. https://howtomechatronics.com/tutorials/arduino/how-to-build-an-arduino-wireless-network-with-multiple-nrf24l01-modules/

[8] McManus, G. (2022, February 7). Project repository. https://gitlab.csc.uvic.ca/gmcmanus/unitytransformcontroller_csc485e_project1