Gesture Based UI Development Project

Introduction

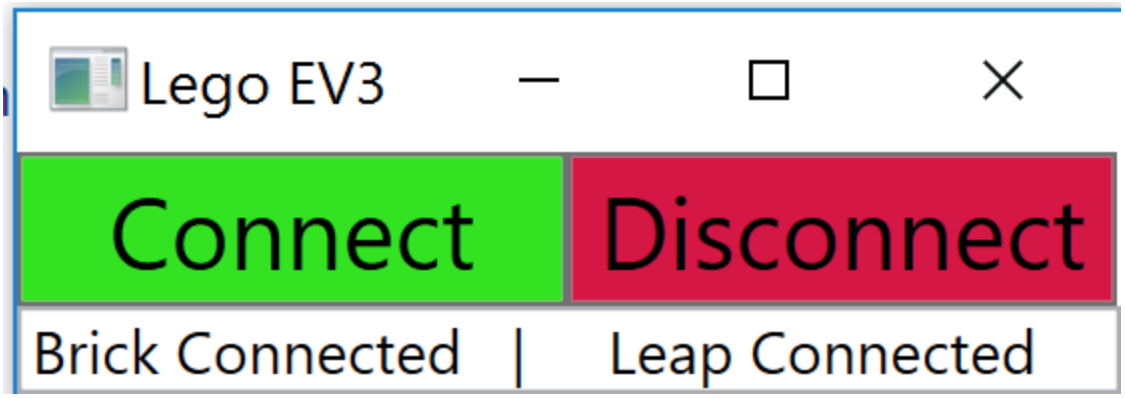
This project is for a Gesture Based UI Development module for 4th year Software Development at Galway Mayo Institute of Technology. The aim of this project was to develop an application that would demonstrate the use of gesture technology.

Purpose of the Application

The purpose of my application was to use the Leap Motion device to capture gestures and use those gestures to control another physical entity, in this case, a Lego Mindstorms EV3 robot. I wanted to come up with a number of gestures that would intuitively make sense with what motions they were mapped to on the robot. The chosen gestures are discussed in detail in a separate section.

The user interface is basic and contains just two buttons and a message box. The first button attempts to connect via Bluetooth to the Lego robot and connects to the Leap Motion. Once both are connected, a message is output to the UI. The second button is to disconnect to the robot and once again a message is output to the UI.

First UI screen when app starts



UI screen when robot and Leap have been connected

UI screen when robot has been disconnected

The Leap Motion can be programmed using a number of different programming languages and my initial idea was to use the JavaScript SDK provided by developers. However, in researching different languages to use, I found that C# was without a doubt the most popular. There are several APIs and plenty of documentation for C# so it seemed like the most logical solution was to program both the robot and the Leap Motion using C#.

I used Visual Studio 2015 to create a WPF application for the project. I downloaded the official Leap Motion SDK from the Leap Motion website found here (Leap Motion, 2017). The Leap Motion website contains extensive documentation for the C# SDK and the functional aspects of the Leap. It also contained examples of how to set up a Visual Studio project for using the Leap Motion.

I needed to create a reference in the WPF application to a .NET dll file in the SDK called LeapCSharp.NET4.0.dll. I also had to add a Post-build event in the project options to point to a second dll file called LeapC.dll from the SDK. I was then able to add an import to the project called “using Leap” and had access to all the libraries for the Leap.

In the program, I created an instance of a Controller which I used to create a newFrameHandler when the Connect button on the UI was pressed. The newFrameHandler is an async method that constantly updates. The gesture mapping I used from the SDK is discussed in detail under the *Gestures Identified as Appropriate for Application* section of this report.

For the Mindstorms programming, I used an unofficial SDK (referenced later). To add control of the brick to the WPF application, I added a reference to Lego.Ev3.Desktop which contains classes for things such as connections to the robot and dll files. Once this was added, I could use imports from the API in the application.

I create an instance of a Brick which refers to the control brick located on the robot. When the Connect button is pressed on the UI there is an attempt to connect to the brick over Bluetooth on a specific port. The application contains a number of async methods that control the different motors of the robot. These methods are called from the newFrameHandler when certain gestures are detected by the Leap Motion.

There are two different ways that commands can be sent to the brick. One is a Direct Command which sends just one command at a time. There is also a Batch Command which can be used to send a group of commands to the brick at once. For movement commands, the application sends batch commands when more than one motor is being utilized at once. For other methods, where only one motor is being used, a direct command is sent.

The API contains specific commands for controlling the motors. There is the option to send continuous power to a motor or to send power for a certain amount of time in milliseconds. I decided to send power for a certain amount of time. This was because if there is a break in the connection between the application and the robot and there was continuous power being sent to the robot, it would continue to power the motor leaving the user no longer in control. However, this does result in more commands being sent to the robot but I decided that this was the better option.

Gestures Identified as Appropriate for Application

The Leap Motion can identify many gestures and motions which are specified in the official documentation (Leap Motion, 2017). It captures from the elbow to the fingers of one or both arms. The main individual object classes that can be recognized are as follows:

Hand

The hand object is a list returned from the frame of how many hands have been captured (up to two) by the sensors. Certain details can be obtained from the hand objects. Below is an example of some of these:

- GrabStrength – this is a range between 0-1 of whether the hand is open (0) or a closed fist (1)
- IsLeft/IsRight – identifies whether the hand is a left or right hand
- Direction – this can be used to get float values for the pitch, yaw and roll of the hand
- PinchStrength – a float value between 0-1 of whether the hand is making a pinching gesture (1) or not (0)
- WristPosition – a vector that contains the coordinates of the wrist position

Fingers

The finger object is a list of Fingers captured by the frame that are attached to a given hand. Each finger has an ID corresponding to it, a finger type and a count value.

Arm

The Arm object is gotten from a valid Hand object. Information that can be obtained from this object include whether an arm is equal to another arm, the direction of the arm and the position of the elbow.

Gestures

The Gesture class was included in the first version of the Leap Motion SDK and included gesture functions such as Circle, Key Tap, Screen Tap and Swipe. These gestures are now depreciated as the Leap Motion team felt that they were unreliable gestures to support in favour of gestures such as grabbing and pinching. (Leap Motion Team, 2016)

When deciding on the gestures that would be appropriate for this project, I experimented with the gestures that were available. I wanted to identify distinct gestures that would be unlikely to be confused with others. I also thought about what parts of the robot would be controlled and wanted to match similar gestures to the actual actions on the robot so that it would be quite intuitive to control.

The robot I started out this project with was a truck with 3 motors attached, two of which controlled each of the back wheels of the robot and the third which controlled the steering of both the front wheels.

(Lego Mindstorms)

I started out testing with one hand, using gestures to control the robot but it became apparent to me quite quickly that this was going to be a difficult and somewhat convoluted task. This was because using only one hand minimized how much I could control on the robot. An example of this was making the robot go in a forward centralised motion and steering the robot left and right. This was quite easy to do using one hand but after a few minutes of testing it with the robot, it became clear that I would need to add in gestures for reversing. This was when I decided to have two hands for controlling it.

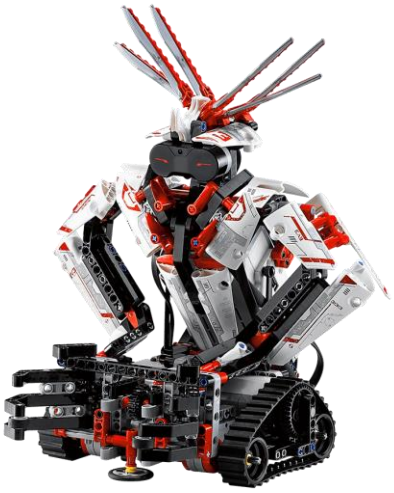
On the original robot, I used the following gestures:

- **Left hand roll anticlockwise and not in a fist** – send power command to steering motor clockwise steering the robot to the right
- **Left hand roll clockwise and not in a fist** – send power command to steering motor anticlockwise steering the robot to the left
- **Left hand fist** – stop power to steering motor
- **Right hand fist** – send power command to both back wheel motors clockwise sending the robot in a forward motion
- **Right hand pinch** – send power command to both back wheel motors anticlockwise sending the robot in a backward motion
- **Right hand no fist or pinch** – stop power to both back wheel motors

These gestures worked relatively well but I encountered some issues with both the gestures and the robot.

1. The closed fist gesture made the robot move forward and the pinch gesture made the robot reverse. The Leap Motion would often confuse these two motions depending on the position of the hand when making this gesture causing the robot to jerk backward and forwards.
2. Steering the robot with just two gestures i.e. left hand roll to go left and right hand roll to go right meant that the robot was only ever going right or left and never in a straight forward direction.

I decided to rebuild the robot to allow for easier steering and more functionality.

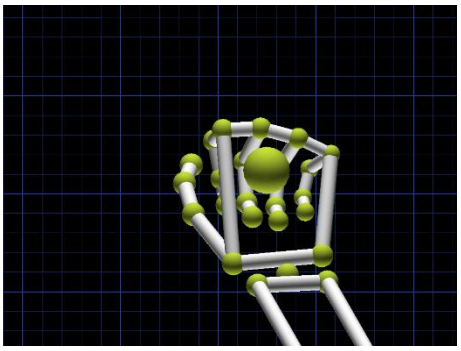


(Lego)

This new robot uses 2 tank tracks instead of 4 wheels and contains a gripper on the front of it that would allow it to pick up, carry and drop objects. I decided this would be a good combination to use with gestures.

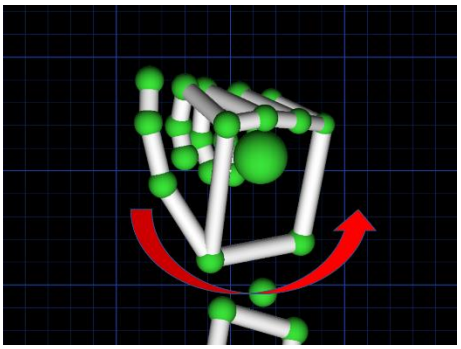
I looked to the Leap Motion SDK when deciding on appropriate gestures for controlling this robot. I wanted to avoid the previous issue of having similar gestures that could be detected by the Leap Motion incorrectly. Below are the gestures I decided on:

Moving Forward



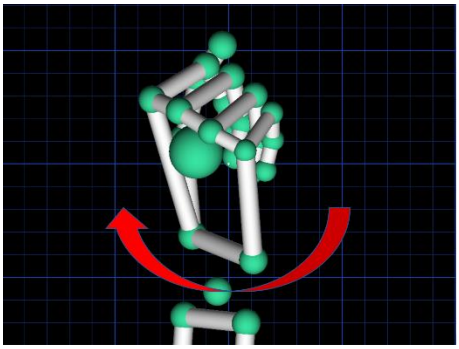
Right hand fist (centred)

This gesture sends power to both motors controlling the tank tracks on the wheel. I believe the centred position of the hand makes sense for the user to send the robot in a forward direction.



Right hand fist roll clockwise

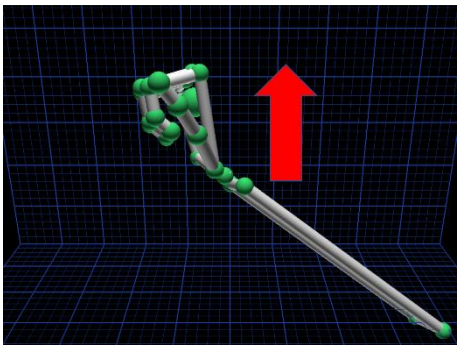
This gesture sends power to the left tank track motor sending the robot right.



Right hand fist roll anticlockwise

This gesture sends power to the right tank track motor sending the robot left.

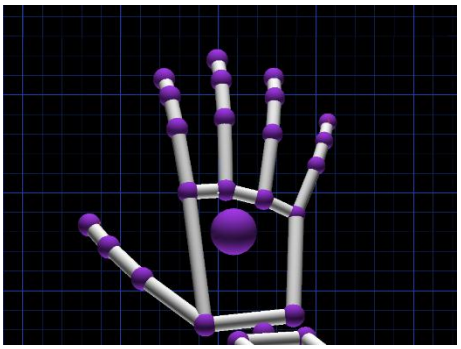
Rolling the fist to the left or right depending on what direction you want to send the robot seemed like an instinctive way to control direction.



Right hand fist raised (pitch)

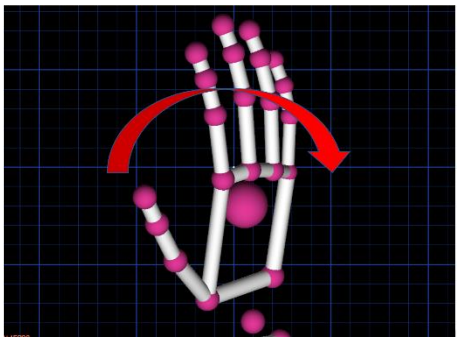
This gesture stops power to both tank track motors stopping the robot. Using a raised arm gesture to stop power to the motors is quite distinctive and could be compared to someone stopping traffic so I felt it was appropriate to use.

Moving Backward



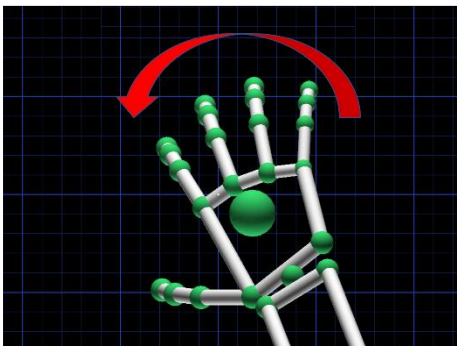
Right hand open palm (centred)

This gesture sends power to both motors in the reverse motion to control the tank tracks on the wheel. I decided on this gesture as it is similar in function to the forward motion but distinctive enough not to be confused for a fist.



Right hand open palm roll clockwise

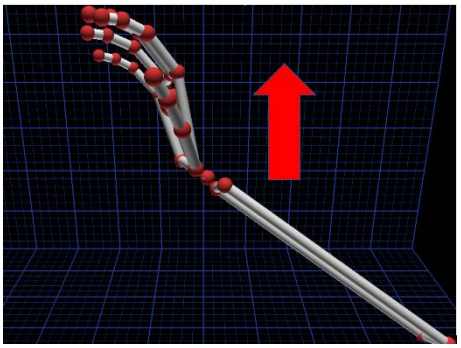
This gesture sends power in the reverse to the left tank track motor sending the robot backward and right.



Right hand open palm roll anticlockwise

This gesture sends power in the reverse to the right tank track motor sending the robot backward and left.

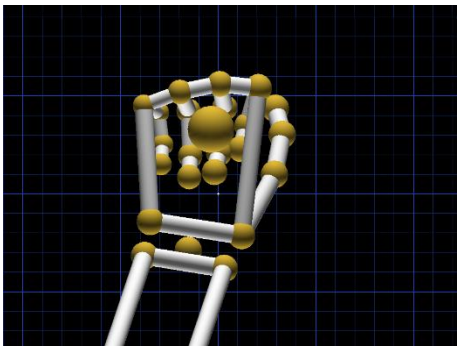
Like the forward motion gestures, rolling the palm to the left or right depending on what direction you want to send the robot seemed like a instinctive way to control it.



Right hand open palm raised (pitch)

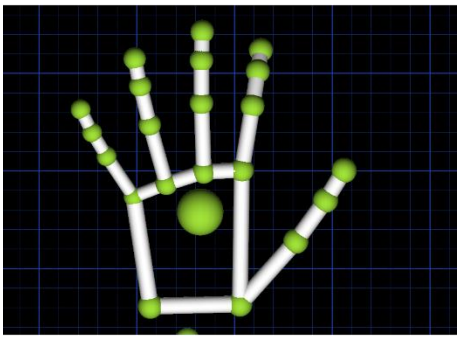
This gesture stops power to both tank track motors stopping the robot. As mentioned in the forward motion gesture section, it seemed to me that raising the arm to stop the robots' movement would be innate. I wanted to have both stop motions be similar so the user would not have many different motions to recall.

Controlling the Gripper



Left hand fist

This gesture sends power to the motor controlling the gripper on the front of the robot and makes it close.



Left hand open palm

This gesture sends power to the motor controlling the gripper in the reverse causing it to open.

The gripper is quite like a hand opening and closing and so I wanted the gestures controlling it to be as close to the actual motion as possible.

As I have discussed above, I chose each of the gestures using logic and reasoning. Some of the gestures, like the opening and closing of the hand to open and close the gripper on the robot, I chose because they were quite close to the actual motion that the user is carrying out and to me it appeared that it was the most intuitive way to control it. In other cases, such as powering the robot forward and backward, I wanted to have similar to each other so that the user would only have to think of one gesture and then carry out that gesture with either an open or closed hand. However, the gestures themselves are distinctive enough to each other that there would be less of a chance of the Leap Motion confusing the two gestures leading to the robot doing the opposite of what the user was trying to make it do.

Hardware Used

Leap Motion Controller

Leap Motion Controller is a gesture capture device that uses three infrared LEDs that generates pattern-less IR light and two monochromatic IR cameras that generate almost 200 frames per second of reflected data. There are several official SDKs for different programming languages for the Leap motion. The C# SDK is utilised for this project.

Out of the available gesture hardware available for this project, I decided on the Leap Motion for the following reasons.

1. I had used the MYO Connect in the past and had found it to be quite limited in what gestures it has. There are only 5 main gestures and I felt this would be quite restrictive for any project I might decide to do.

2. In an earlier assignment, as part of the Gesture Based UI Development, I had done research on several types of gesture technology. I had come across the Leap Motion Controller regarding its use in Virtual Reality. I was quite interested in experimenting with it and applying it to a real-life project.

3. In researching through the official documentation for the Leap Motion, I found there was a wide range of gestures it could capture and that the frame was updated so quickly (200 frames per second) so there would be very little latency in detecting gestures.

To set up the Leap Motion on my laptop I had to first download software from the official Leap Motion website. The software included the Leap Motion Control Panel which allows for calibration of the device and various settings. It also contained the Leap Motion Visualizer (used to create the gesture pictures in this report) and the Leap Motion App Home where you can download apps for use with the Leap Motion. The Leap Motion connects to the computer by USB. It comes with a long and a short USB cable for different uses.

To begin development with the Leap Motion, I had to decide what language I would be using to program it with and then download the SDK for that language, which in this case was C#. The SDK contained many libraries which could be used in the application.

Lego Mindstorms EV3 Robot

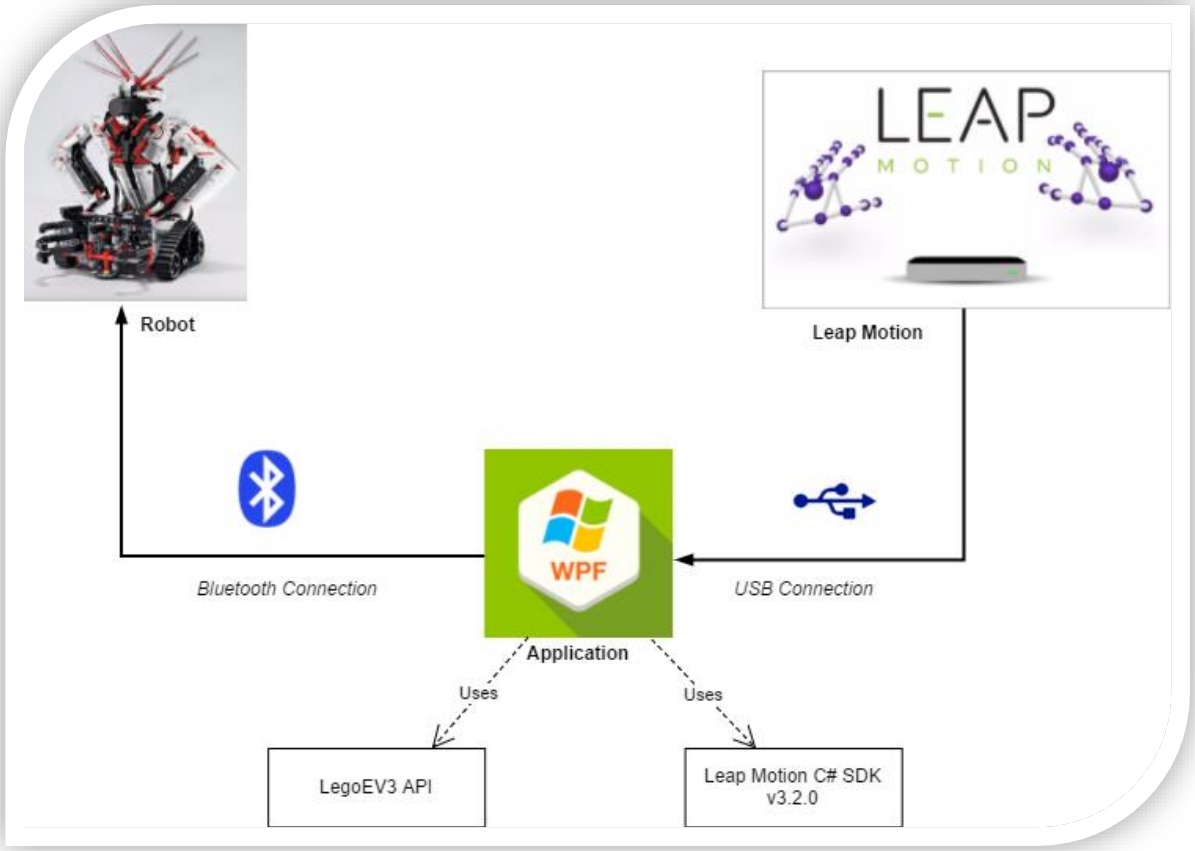
My original idea for this project was to use a robotic arm that would emulate the gestures detected by the Leap Motion. However, I realised that the robotic arm did not have a hand like structure but instead had a pincer. I felt this limited the scope of what I could do with the wide range of gestures that can be captured by the Leap Motion. Instead, I decided to use the Leap Motion in conjunction with the Lego Mindstorms EV3 Robot which opened up a wide number of possible controls.

Lego Mindstorms EV3 is a Lego robotic kit which can be used to build many types of robots with Lego blocks, motors, sensors and a programmable control brick. The official way of programming the Mindstorms EV3 is using a GUI software using drag and drop blocks. For this project, I programmed the brick in C# using an unofficial API which is available here (Peek, 2015).

For the final robot build, I used two large motors to control the tank track wheels for the robots' movement. I used a medium sized motor to control the gripper on the front of the robot. The Lego Mindstorms website provides guidelines on how to build different types of robots and the one I used can be found here.

There are two ways of connecting the robot to the computer, through Wi-Fi and Bluetooth. To connect to the robot using Wi-Fi, a USB Wi-Fi dongle must be connected to the robots control brick in the designated port. Lego only supports one brand and type of USB dongle, the NETGEAR N150 Wireless Adapter (WNA1100) which is no longer in production and is difficult to track down online. Therefore, I decided that using the Bluetooth connection was quite adequate for this project.

Solution Architecture



The WPF application runs in Visual Studio and connects to the Leap Motion via USB. The Leap Motion returns the detected gesture information to the application. The application is connected to the robot control brick via Bluetooth. Once it reads the information returned from the Leap Motion, it sends commands to the robot. The SDK, API and application are discussed further under the section *Purpose of the Application*.

Conclusions & Recommendations

This was a interesting project to work on and I thoroughly enjoy doing it. At the start of it I was unsure how I was going to go about making the various parts of hardware work together. I also made a few changes along the way about the hardware and programming languages I was going to use. In the end, I was happy with the result and began to see practical uses for what began as just an idea of making a robot move. If I was to do this project again or had more time to work on it I would like to see how much further the robot could be optimized for practical use.

I believe that it would be possible to incorporate this kind of technology, i.e. gesture technology and some sort of motorized entity, into assistive technology for people suffering from disabilities and/or the elderly. In its present state, the robot moves around on a surface and can be used to pick up objects, carry them and drop them off elsewhere. An alternative type of robot build could include an extra mechanism to raise and lower the robot so that it could reach objects from various heights and be raised up to a level where the user could retrieve the item from the robot without having to bend down.

There are also sensors that come in the Mindstorms EV3 robotic kit which could be used to optimize the robot. One such sensor is the EV3 Color Sensor which could be used to design certain colour coded tracks for the robot to follow where the user could send the robot to a location without having it in sight at all times. The robot could be programmed to only follow in the line of the specific colour.

In a similar way, the EV3 Infrared Sensor could be used to detect objects or obstructions and avoid them. The EV3 Touch Sensor could be used in unison with the existing gripper so that when an object is picked up, the touch sensor would be triggered, alerting the user that the item was successfully grasped.

In terms of the gestures, it became clear to me from experimenting with different gestures, that some are more comfortable than others. Some gestures, such as rolling the wrist inward, can be awkward in some situations and so I strove to make the range of roll small enough that it would detect the motion without the wrist having to be rolled to an extreme position.

I also realized that for this application to be practical, the user should not have to learn off a broad range of different movements. I felt it was important to have the smallest number of gestures possible to do the most actions. I also wanted to ensure that any gestures used in the program were as close as possible to the actual motion the robot would be carrying out so that rather than having to learn off gestures, they would be intuitive.

In order to test this idea, I had 2 6-year-old children try out controlling the robot with gestures. I explained to them that their left hand controlled the gripper and their right hand controlled the robot moving. If they wanted to go forward they were to make a fist and open that fist to go backward. The gripper control sunk in instantaneously and they could open and close it. The movement control forward and backward took a little longer but once they knew which was which, the rolling motions to turn also became quite natural for them.

The Leap Motion is being targeted as a Virtual Reality technology but I found from working on this project that it also has a place in the real world.

References

Leap Motion. (2017). *C# SDK Documentation*. Retrieved from Leap Motion: https://developer.leapmotion.com/documentation/v2/csharp/index.html

Leap Motion. (2017). *Desktop Setup.* Retrieved from Leap Motion: https://www.leapmotion.com/setup/desktop/windows

Leap Motion Team. (2016, April 11). *Leap Motion Community*. Retrieved from Leap Motion: https://community.leapmotion.com/t/deprecated-gestures-api/4968

Lego. (n.d.). *GRIPP3R*. Retrieved from Lego.com: https://lc-www-live-s.legocdn.com/r/www/r/mindstorms/-/media/franchises/mindstorms%202014/robots/product_gripp3r_mainstage.png?l.r2=-523486258

Lego Mindstorms. (n.d.). *RAC3 Truck*. Retrieved from Lego.com: https://lc-www-live-s.legocdn.com/r/www/r/mindstorms/-/media/franchises/mindstorms%202014/robots/product_rac3r_mainstage.png?l.r2=-940854035

Peek, B. (2015, December 19). *LEGO MINDSTORMS EV3 API for .NET*. Retrieved from Github Repository: https://github.com/BrianPeek/legoev3