# Maestro Wind Turbine Maintenance Simulation

## Research Report - IMT&S

09.2025 - 01.2026


Andreas Degtjarow

**Research Supervision:** Yvens Rebouças Serpa

**Saxion SMART Coaches:** Dion Zwakenberg & Matthijs van der Meulen

# Abstract

With the rise of wind turbines in favor of sustainably generated energy, a growing need for maintenance work is due to keep them intact. Though wind turbine maintenance specialists are available, they cannot keep up with the increasing amount of wind turbines being built, the process also being dangerous. As such, drones are being developed to make this process safer and time efficient. The aim of the project covered in this research paper was to develop a maintenance simulation, with the focus of using the bristle blaster, which is an instrument used for such work. Both the drone and the human were to use it, emphasizing the similarities, benefits, and downsides. To achieve this, developing an application featuring haptic feedback plays an essential role, as it is key to simulate comparable sensations, when considering the real instruments haptics. This report covers the implementation of haptic feedback inside Unreal Engine 5.6.1, as well as other supporting features which add to the experience. In the end, we successfully developed a reusable, and adaptively changing haptic feedback system, that was capable of simulating the experience of using real instruments. In the tests conducted, participants confirmed that the simulation achieved this desired effect, leading to an improved quality of the developed product.

# Table of Contents

# 1. Introduction

Sustainability is a crucial topic in today's society (Kasinathan et al., 2022), leading to many sectors changing into reliable and reusable resources, which includes the energy sector. As a result, wind turbines are on the rise, being one of the most popular choices for sustainable energy generation (Arias et al., 2025). However, as with all complex machinery, wind turbines require regular maintenance to function safely and efficiently (BGB, 2024). This is time-consuming, resource heavy, and even dangerous, as it is done at high altitudes, often in remote places. To protect the maintenance workers, while also improving efficiency, drones are being used and developed as an alternative approach for this process (Omara et al., 2025).

On the topic of wind turbine maintenance, project Maestro Wind Turbine Maintenance aims to develop such a drone to solve this. However, development faces some challenges, as the drone would miss on important characteristics such as haptic feedback, while also requiring proper training prior to the use of such modern technology. For that, maintenance workers are skeptical about this solution, first for not being used to such technology, and second because of the lack of haptic feedback, which might considerably harm the quality of work. This is especially important for tools such as the bristle blaster, to which workers have already grown accustomed to the haptic feedback, leading to higher quality of work.

To circumvent this, the development of a simulation in a group effort was put in motion, helping maintenance workers become more familiar with the drone, and showing its increased efficiency. Goal of the simulation is to experience the same level of haptic feedback when using a drone, as when doing it by hand. As such, the worker would be able to use this instrument inside the simulation both as a human **(Human Mode, HM)**, as well as a drone **(Drone Mode, DM)**, making it easier for them to compare. The applied haptic feedback would be adaptive, changing the intensity based on how much the instrument is being pushed against a surface, which can lead to enhanced performance, as well as improved user experience (Rowland et al., 2024).

By also implementing different types of haptic feedback, to convey the proper feeling of using the bristle blaster, immersion can further be increased (Pretorius, 2024). Within HM, the worker would interact using handheld controllers, to give enough freedom of movement. Such controllers only have **Vibration Feedback features (VFB)**, which is what will be used for this scenario. While in DM it would also be possible to use the same handheld controllers, a flight stick with **Force Feedback features (FFB)** could be made compatible too, potentially leading to more intuitive controls for some tasks (Gibbs et al., 2022). However, the necessary hardware for FFB features could not be provided, and further research could not be conducted.

To support further immersion, this simulation was developed in **Virtual Reality (VR)**. Utilizing Unreal Engine 5.6.1 (UE), which is a game engine known for its VR specific features and realistic looking environments (Berrezueta-Guzman & Wagner, 2025), it will be easier to ensure the product stays immersive and well functional (El-Wajeh et al., 2022).

All this holds uncertainties and questions however, as UE does not support some of the necessary features and other threats also exist, such as wrong usage of haptic feedback, possibly harming the experience.

# 2. Problem Analysis

For adaptive haptic feedback to be implemented for both HM and DM, a common value needs to be calculated, with which to apply the haptic feedback onto the controllers. Though UE contains methods to detect collisions

with additional information, it will take certain configurations to be made for the interacting components, to be able to access them. To also make the code work with any mesh tied to the code, measurements need to be taken dynamically of the objects. Additionally, a method to acquire the **Texture Coordinates (UV)** of the collision, will also need to be researched, as this will be essential to make the bristle blaster have a visible effect on the interacted model.

Developing configurable tools and scripts will also be important, to allow easy access and tweaking for other team members. Regarding haptic feedback, there is also a phenomenon called “uncanny valley of haptics”, where the simulation feels less immersive despite the use of haptic feedback (Gibbs et al., 2022). Though it will be important to prevent this in the developed simulation, the task itself will be delegated to one of the designers of the team.

Both HM and DM are also very similar in their interactions, as they fulfil the same purpose, just in a different manner. As such, two different ways of functioning need to be developed, within a single interface inside UE, to keep the software architecture clean.

# 3. Research Question

How can a system be developed inside UE 5.6.1, which realistically simulates the feeling of using the bristle blaster in VR utilizing haptic feedback, within a clean architecture?

Sub-questions:

- How can adaptive and configurable haptic feedback be developed, in the context of UE 5.6.1?

- How can both HM and DM be implemented using a single interface?

# 4. Scope

Though the bristle blaster in use will have to be simulated realistically, especially in terms of haptic feedback, other parts such as the drone’s movement, do not hold the same requirement.

Due to time constraints and missing hardware, no development towards FFB features will be made, focusing solely on VFB instead.

Additionally, the simulation will cover only a single aspect of the possible processes that can be undertaken for maintenance on the wind turbine, namely repairing a layer of one of the wings. The development of further aspects of wind turbine maintenance is outside of scope.

Furthermore, it was not possible to prepare testing sessions with people who had real experience using the bristle blaster tool. Though such tests were requested, attempts at getting into contact were unsuccessful.

# 5. Development

The bristle blaster, and as such also the haptic feedback, consisted of multiple different pieces of logic which needed to be put together to make the feature work as intended. As such, the sections are set up based on the importance of the topic in the context of this report, rather than the sequence of development. Additionally, this section only covers the features, which are related directly to the bristle blaster simulation. Other parts, which were equally important, but unrelated to the bristle blaster, will be covered in another section.

It first introduces collision logic and haptic feedback implementation inside UE, as this is the topic that is focus of this research paper. Following up, mask painting will be covered, being the result which is supposed to be shown after using the bristle blaster.

# 5.1. Haptic Feedback

## 5.1.1. Adaptive Vibration Feedback

To get started with developing haptic feedback, the first goal was to apply any sort of vibration feedback, since this also gave an oversight of its capabilities and possible configurations. Using the "Haptic Feedback Effect Curve" asset from UE and adjusting its graphs *(Figure 1 & 2)*, only a function inside the VR character blueprint needed to be called, to result in one of the controllers vibrating. This function also possessed a scale value, which was then used to create adaptively changing intensities. At this stage, the two biggest questions were how to calculate the scale value properly, as well as how to determine which hand to apply this haptic feedback onto.

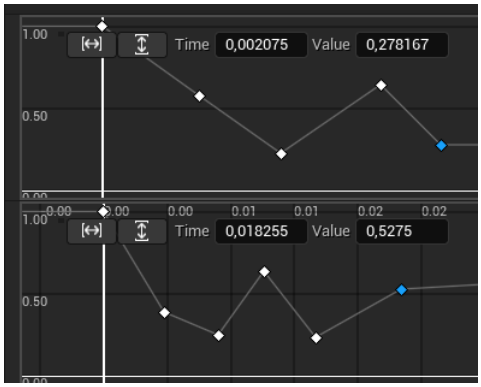


*Figure 1 Haptic Feedback Curve - Example Configuration*

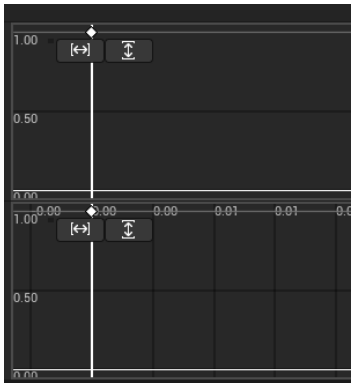


*Figure 2 Haptic Feedback used (Better Control)*

Thanks to the mask painting features implemented, which will be covered in the following sections, it was most convenient to start with the scale calculations, as the existing collision calculations could serve as basis for this new addition. Using the ray-casts for determining the UV coordinate of the collision, it was possible to extract the length of the vector from origin to the point of impact. This value could then be mapped from zero to one, based on the max length of the ray-cast vector *(Figure 3)*. The result of this calculation was the collision depth, and attaching it to the scale value, after parsing it into the VR character blueprint, the exact adaptive vibration feedback needed for this simulation was achieved.

To figure out what hand to apply the haptic feedback onto, the existing VR character logic needed to be analyzed and expanded upon. Grabbed objects were being registered already, so there was no need to create a new layer of logic for it. Instead, this data was to be saved and checked upon, before applying the adaptive vibration feedback. As such, whenever a valid haptic tool was grabbed, the used hand was registered in an array, which would then be looped over for the haptic feedback application.

Additionally, the intensity of the haptic feedback should also depend on the position of the hand, relative to the top of the bristle blaster. For this, strings were used, which were then mapped to the references of the hand sockets, stored inside a dictionary on start of the simulation. This was done because no information about the socket was available by default, as it handled this interaction like any other grasp. Eventually, an OnGripped event was found which wasn't in use previously, that held the string name of the slot, leading to the solution mentioned.

After everything had been set up, and tests showed no signs of bugs, this feature was made final and ready to be used. Special attention was also given to the reusability aspect of this implementation, turning the logic into

a component with a configurable setup function, to make the adaptive haptic feedback easy to add for the drone version of the bristle blaster.

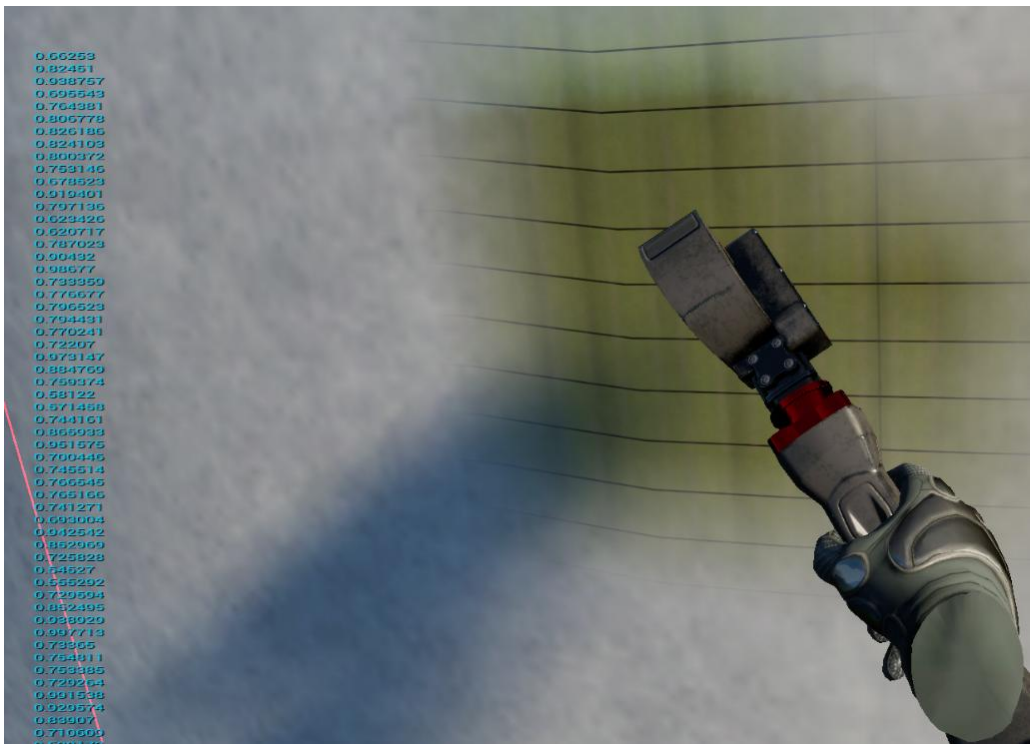


*Figure 3 Collision Scale - Print Results*

### 5.1.2. Activation Control for Bristle Blaster

To further enhance the realism of the bristle blaster instrument, it was decided to make it gradually activate first, before being able to scrape off the coating. By listening to a button press on the VR controllers, the bristle blaster would increase the scale value from zero to one, using a certain speed value. This would be used for both the rotation speed of the upper instrument part, as well as serve as additional multiplier for the opacity of the painted-on texture.

With this addition, the haptic feedback was also in need of adjustments. It was to be applied together with the button press, and using the scale, the intensity of the vibration should vary. This also meant that the lower limit of the previous bristle blaster collision feedback needed to be adjusted properly, as the transition between being only active, and actively colliding, should feel seamless. All this was a matter of adjusting some values, thanks to the logic implemented in previous iterations. Though this task could have been also done by a designer in the team, it was easier to implement this change right away, as this would prevent potential merge-conflicts when putting together branches of the repository.

## 5.2. Mask Painting

### 5.2.1. Developing the Core Logic

As the main maintenance process was using the bristle blaster to take off the coating, this implied blending different textures together onto the same surface. This process can also be described as mask painting, as the RGB values of a texture can be read to determine which texture should be painted, at which UV. UE supports such logic, as by activating an option called “Support UV From Hit Results” inside the project settings, exact UVs can be extracted from collisions such as ray-casts. To ease the workflow and be able to develop this feature quicker, a tutorial was followed (Aid, 2021), which explained and showcased how to properly implement the

exact feature needed for our simulation. This included creating two different materials, one for the paint applied and the other for the actual material used for the mesh, which would be blended with different attributes. The actual painting would happen on an asset called Render Target, which is a texture onto which various data can be stored and manipulated.

To be able to cause collisions to start painting, some sort of instrument needed to be created. As the bristle blaster model was not yet in development at this point, a quick placeholder was created, mimicking the instrument with basic shapes *(Figure 6)*. Though the first idea was to use trigger collisions to detect surfaces, tests showed that participants encountered bugs, where the instrument would either activate only after multiple tries, or not behave as expected, painting on a different spot than where the instrument collided with first. As such, these trigger colliders were replaced with ray-casts, which resulted in more precise collision detection, as well as easier management of the logic itself, as these were simpler to comprehend than the events of UE trigger colliders. They also held more hit information compared to the trigger collisions, and were fully customizable in terms of length, direction and origin position.

After all logic was implemented inside a blueprint of UE, it was then possible to paint upon the Render Target with simple colors, translating the collision information into UV space *(Figure 4)*. This blueprint was also reconstructed to be a component, able to be used by any other blueprint, to first decouple logic, and second also make it reusable.

By reading the R channel from the Render Target asset, different textures were then able to be blended on the material of the mesh *(Figure 5)*.

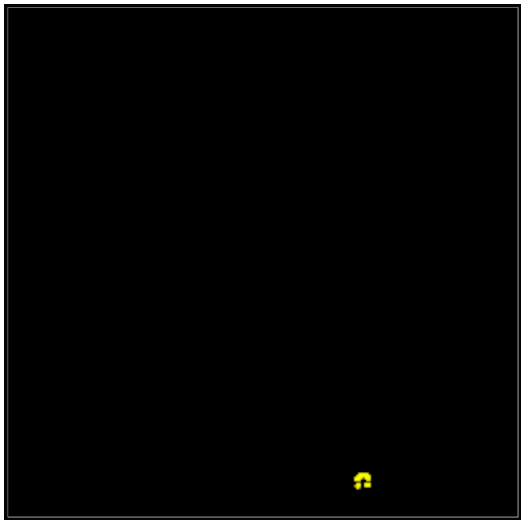

*Figure 4 Render Target used for Mask Painting*

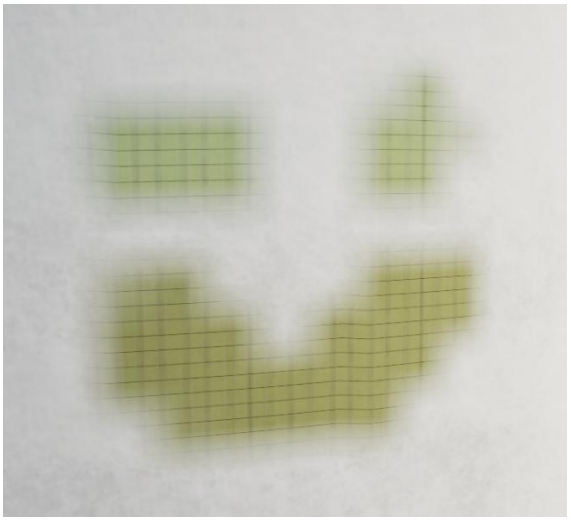

*Figure 5 Mask Painting blending two textures*

## 5.2.2. Adaptive Mask Painting

Since the bristle blaster introduced a scale based on the collision depth of the instrument, this was also an opportunity to introduce adaptive opacities of the painting applied. With this, the worker would not only be able to feel the force used through haptic feedback but also see the results properly on the worked-on surface *(Figure 6)*. During the development of the mask painting feature, an opacity parameter from the brush material was already exposed. What needed some adjustments though, was the exact scale value with which to apply the opacity with. One difficulty was that the exposed scalar parameter was not applying opacity in a linear, but rather in an exponentially growing way, which led to very small values feeling a lot stronger than they should. As such, the scale value needed additional mappings, one for the lower half of the values, and another for the upper half. This solution solved the problem mentioned, making it appear seamless during usage, combined with the haptic feedback received.

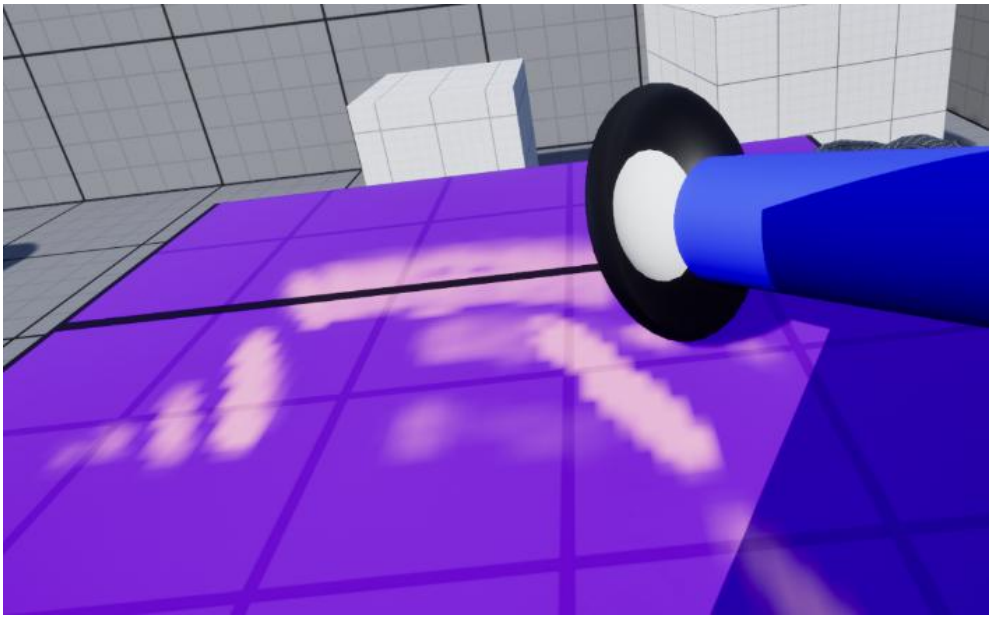

*Figure 6 Mask Painting applied adaptively (varying opacity) using placeholder model*

### 5.2.3. Progress Feedback for Patch Maintenance

As the render target already held data in form of pixel colors, the most convenient and straightforward way of keeping track of the current progress of maintenance was by evaluating the percentage of pixels, which have higher RGB values. Though blueprint does hold functions to read single pixels of a texture, this node is known for its heavy impact on performance, due to it running on the same thread as the main loop, also being CPU based. Since VR applications are especially sensitive to such performance heavy logic, custom code needed to be developed in form of a C++ script, to remain efficient.

With the help of an online post discussing such topics (Nicholas477, 2023), the main idea was to create a separate thread, in which to read the render target. After checking whether the provided render target instance is valid, and more than zero pixels could be read, a function called “ParallelFor” was used to iterate over every pixel. This function also took care of creating a new thread for processing. With each pixel being greater than zero on the R channel, an integer value was incremented by one, which was then used to calculate the percentage of the filled pixels, after the for-loop was finished. This approach resulted in a huge increase in performance, with which it was possible to keep track of the maintenance progress on each frame, whenever the render target was painted upon, without any apparent performance drops even inside VR.

## 5.3. Sound Logic

After recording the real sounds of the bristle blaster with the help of one of the designers, the results were edited into smaller sound files to be used for separate states, namely “Started”, “Stopped”, “Running”, and “Colliding”. Using a Meta Sound asset, which is a sound asset similar to a blueprint, logic was developed with which it was possible to manage the currently played audio. Benefits of using such an asset, opposed to other alternatives such as a Sound Cue, was that seamless transition was possible, as it used a separate thread for playing audio (MetaSounds: The Next Generation Sound Sources, 2026). This meant that there was no pause when switching from one audio file to another, while another sound was playing at the same time. To manage the states of the sounds, trigger inputs were used instead of variables, which were called inside the bristle blaster blueprint. This made the logic tidier and easier to comprehend, while also circumventing the issue of having to replay the audio with every variable switch, as this is not done automatically within the Meta Sound asset.

Attention was also given to the seamless sound transition between the states “Started” and “Stopped”, in case the instrument was turned off prematurely. Using the playback variable, which is available for every sound that is being played, a value was calculated that informed the sound played for the “Stopped” state, what start time the sound should have. This value could then simply be connected to the “Start Time” input of the sound, to make it work as intended. With this last addition for the bristle blaster, the instrument was finalized.

# 6. Methodology

At first, the necessary interactions were split up into smaller portions of functions, keeping in mind that those would have to be put together at a later point. As such, the bristle blaster itself had three main sub-functionalities that had to be developed, before the intended feature was ready to be used: Collision Depth Calculation, Mask Painting, and Haptic Feedback.

Throughout the development of each of the functionalities, the same process was repeated. First came the initial implementation, making sure the logic worked as intended. Following up this step, the feature was then handed over to the designers of the team, by merging those features into their branch in Git. These would then be implemented and tested, checking if this new addition behaves well within their own created environment. Each feature had its own conditions, to be able to determine whether the implementation was completed according to their own set of expectations.

This approach also helped to solve bugs and add improvements, forming the different iterations of all the logic. Only when all requested additions and issues were marked as finished, the next feature listed in the shared planning was started to be worked on.

## 6.1. Internal Iteration Testing

For the Collision Depth Calculation, the initial implementation and iteration tests during development were done by using simple print strings, and observing whether the percent value matched the visual depth of the actual instrument.

Mask Painting was handled similarly, where during implementation, the feature was tested on its configurability and results. It was checked whether painting different textures worked as expected, as well as if changing values had the assigned effects.

Lastly, for Haptic Feedback, the first iterations were checked by paying attention towards the noticeability of the intensity.

## 6.2. User Iteration Testing

Regarding the Collision Depth Calculation, when this feature was combined with VFB, participants were questioned whether the resulting effect matched the expectations, based on the current depth of the instrument. Through this, multiple ways of calculating the collision depth were explored, acquiring more precise values with each iteration.

When Mask Painting was merged with the designer branch, the values exposed for configurations were tweaked, based on the feedback gathered on the painting process. It was questioned whether it was easy to control, as well as if it felt nice to use.

When the Haptic Feedback feature was handed over to the designers, further testing was conducted with the same question for intensity, as prior with the internal testing, only adding whether the implemented VFB reminded them of using an instrument. With this feedback, the values controlling the intensity were changed, at various stages of collision depth.

## 6.3. Testing Procedure

Two variations of testing procedures were conducted, to acquire the data relevant to improve upon and confirm the success of the bristle blaster simulation. For all versions of testing, five different participants were selected to give feedback by answering the prepared questions. During the test, they were left to explore the environment independently and for however long they liked, only provided with helpful comments if they seemed to struggle for a minute or were confused as to what to do next. Though some of the participants have completed smaller testing with the project previously, most testers came with no expectations or prior experience of the simulation.

The first procedure only included going through the tutorial, followed up with an A/B questionnaire mostly filled with questions about the bristle blaster. The A group would experience the process without haptic feedback, while group B would do the same with haptic feedback included. In the tutorial, participants would not only get in contact with the bristle blaster and haptic feedback however, but also with other interactions such as moving, climbing, and drone control. Every interaction had accompanied videos or pictures next to them, showcasing what to do, and how to do it.

The tutorial starts by making them move their head and body. Turning around, they would see that the hall they are in, leads to a table with multiple cubes laying on top, which need to be picked up to progress further *(Figure 7)*. After this is completed, the wall in front of them disappears, showing a placeholder menu, which needs to be closed to proceed further. Doing this opens another wall, where a column is visible further down the hall, that needs to be climbed down upon using their hands *(Figure 8)*. After successfully doing so, they end up at the bottom of the hall, where they find the bristle blaster laying on another table, with an easy to spot purple wall nearby *(Figure 9)*. After 5% of the wall surface was coloured, the next wall would disappear, uncovering a spot looking similar to a landing pad for helicopters, with a TV set in front *(Figure 10)*. After moving on top of the pad, and pressing the button to take over control, the TV would turn on, showing the first-person view of the drone. After getting familiar with the drone controls, they will spot another purple wall slightly higher up, which needs to be painted on using the bristle blaster attached to the drone's arm. After painting another 5% of the surface, the level needs to be exited by opening and navigating the menu, which needs to be controlled using both hands. Afterwards, the session is finished, and the participants are given the set of questions. This test took between four to ten minutes for the participants to complete.

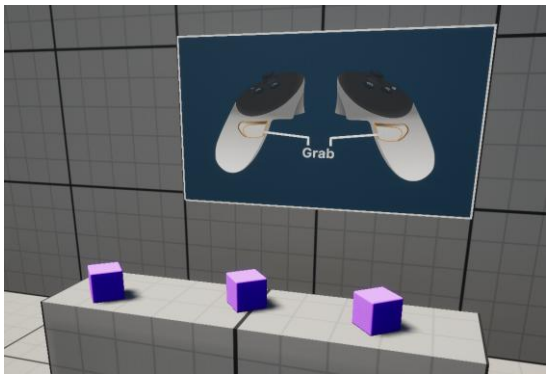


*Figure 7 Table with Grabbables*

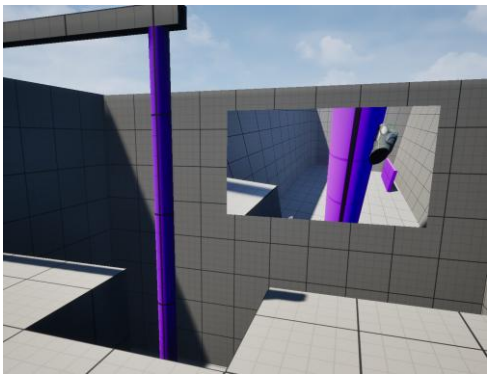

*Figure 8 Pillar for climbing down*

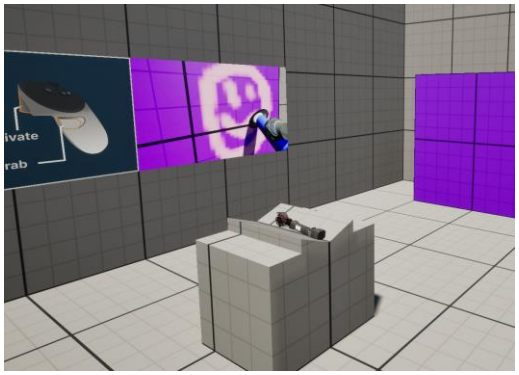

*Figure 9 HM Bristle Blaster Section*

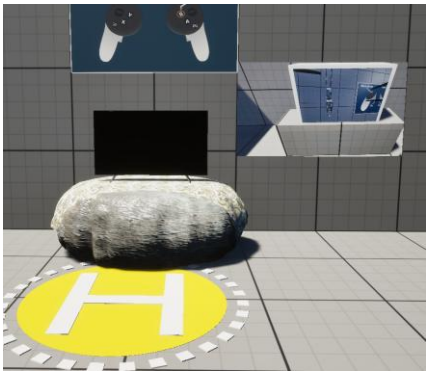

*Figure 10 DM Bristle Blaster Section*

The second procedure involved going through the tutorial, as well as the rest of the levels implemented. This meant they went through the same process as from the first procedure, after which they spawn in an office. There, they got to pick whether they want to do maintenance in HM *(Figure 11)* or in DM *(Figure 12)*, as well as if to do so onshore or offshore. Doing maintenance in HM, the participants would have to climb up a rope, to get to a patch marked by an outline, located on the wind turbine wing surface. By painting 70% of the surface, the rest of the patch automatically fills itself completely, after which the participants need to navigate back to the menu, and go back to the office. Doing the maintenance in DM has the same requirements, only that now, the participants are located at the bottom of the wind turbine, and maintenance needs to be done remotely. Switching between onshore and offshore would only switch the environments, as well as the model of the wind turbine, not the maintenance procedure itself. As such, participants were asked to go through the mode variations of maintenance only once. After completing both, the test was followed up by the questionnaire.

*Figure 11 HM Onshore Maintenance*

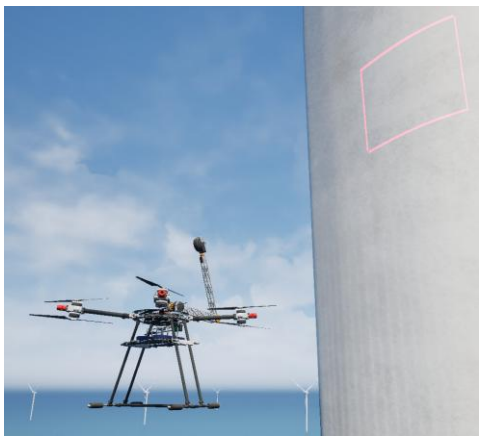

*Figure 12 DM Offshore Maintenance*

# 7. Testing Results

Concerning the A/B testing done for the haptic feedback, there were some important commonalities between both versions that should be mentioned. For one, half of the participants have never used any power tools before, which potentially lead to confusing values, when needing to evaluate the realism of the bristle blaster. Some would simply pick the average value for example, saying they are not able to tell for sure. A different trend which will be noticeable throughout each of the testing sessions, is the low score of the bristle blaster feature combined with the drone. Based on the feedback and explanations gathered from participants, this was not due to the bristle blaster itself and its features lacking in quality, but rather the controls and restrictions of movement for the drone being difficult to get used to.

## 7.1. Tutorial Testing A – No Haptic Feedback

The average of painting the walls using the bristle blaster has no overwhelming values *(Figure 13)*. Most participants would not proceed to colour the whole surface of the wall but instead paint simple symbols like smileys or hearts. In this context, it makes sense that most would have picked the average value, as they would not perceive their paintings as having covered much of the wall. What is noteworthy though, is that the human is leaning more towards the better half, while the drone version does the opposite. Especially since the two questions were asked right next to each other, a preference of the human version of the bristle blaster can be interpreted.


*Figure 13 Test A – Amount of Coloring on the Wall using the Bristle Blaster*

In terms of being realistic, the bristle blaster scored only positively *(Figure 14)*. Though having most responses be around the average means that there is room for improvement still, this is especially good to see, considering that haptic feedback was not yet included for these tests.

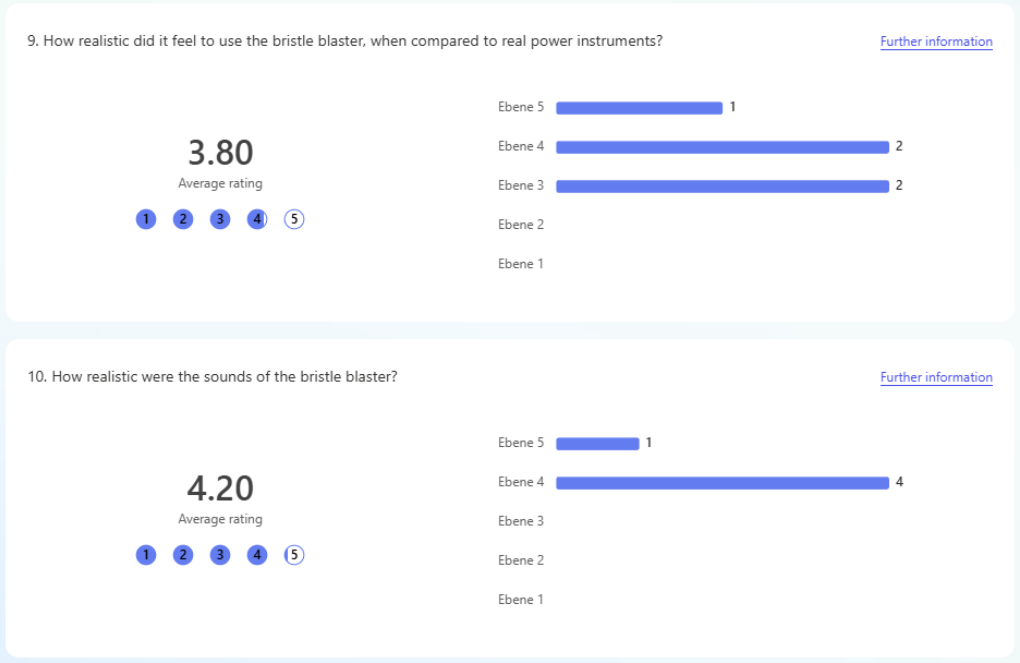


*Figure 14 Test A - How realistic did the Bristle Blaster feel*

When asked how fun and intuitive it was to use the bristle blaster, as well as what their least favourite feature was, it once again confirms the implications, of the DM bristle blaster being worse for the experience than the HM version *(Figure 15, 16 & 17)*. Additionally, the HM bristle blaster was able to score very high on the fun and intuitive side, meaning that the core functionality of the mask painting, combined with the collision logic, is very much liked even without haptic feedback.


*Figure 15 Test A - How fun it was to use the Bristle Blaster*

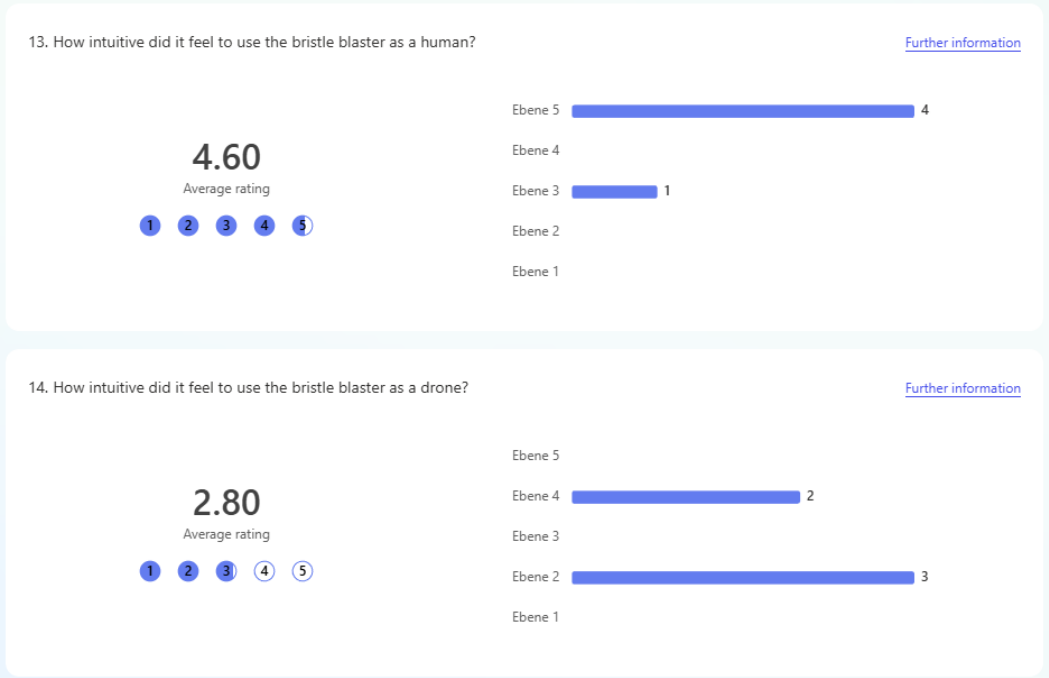


*Figure 16 Test A - How intuitive was the Bristle Blaster to use*


*Figure 17 Test A - Most favourite and least favourite Experience*

## 7.2. Tutorial Testing B – With Haptic Feedback

When asked about the colouring of the wall, it was answered similar to the A version *(Figure 18)*. Most people used the bristle blaster on the drone even less, making the difference in preference between the two modes even more apparent.


*Figure 18 Test B – Amount of Coloring on the Wall using the Bristle Blaster*

The realism of the bristle blaster was evaluated slightly lower by 0.4, than when tested without haptic feedback *(Figure 19)*. Though the difference is not high enough to be able to say that one version was better than the other, it is possible to assume that haptic feedback seemed to not add enough to the realism of the instrument, to be able to score higher points in a scale from 1 to 5. This is an interesting and unexpected outcome, as most people were positively surprised when experiencing the haptic feedback during testing, contradicting the actual results. Perhaps a higher scale from 1 to 10 would have made differences more apparent, something which is going to be mentioned in the recommendation to come. The rating itself is still fine though, as it shows that it was closer to realism with room for improvements, as also seen in the A version.

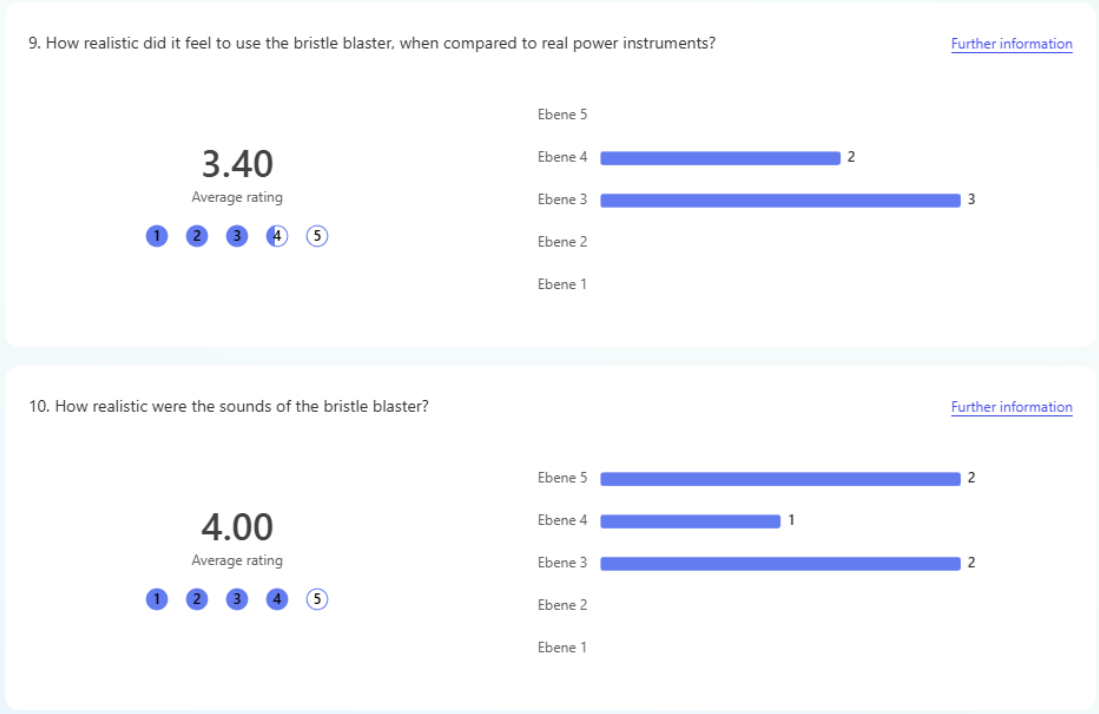


*Figure 19 Test B - How realistic did the Bristle Blaster feel*

In terms of fun and intuitiveness, the bristle blaster was able to score even higher when combined with haptic feedback *(Figure 20 & 21)*. As these values once again differ only by 0.4 on average for HM however, and even less for DM, this change does not seem to be attributed to this new addition. Still, it is a positive indicator for the general implementation of the bristle blaster simulation.


*Figure 20 Test B - How fun it was to use the Bristle Blaster*


*Figure 21 Test B - How intuitive was the Bristle Blaster to use*

Comparing both A and B versions, the preference for the favourite interaction always seems to be either climbing, or the HM bristle blaster, while the least favourite remains to be the DM version *(Figure 22)*. The toolbelt, which will be covered in the upcoming chapter together with climbing, was picked by one of the participants, due to it not being noticeable during testing.


*Figure 22 Test B - Most favourite and least favourite Experience*

## 7.3. Full Application Testing – Haptic Feedback included

Many questions were asked during the full test of the application, dedicating only very few sections for the bristle blaster. The results of those once again confirm that, which was made apparent in the A/B testing too. The bristle blaster remains to be very intuitive, also being one of the most favourite features of the overall simulation *(Figure 23 & 24)*.

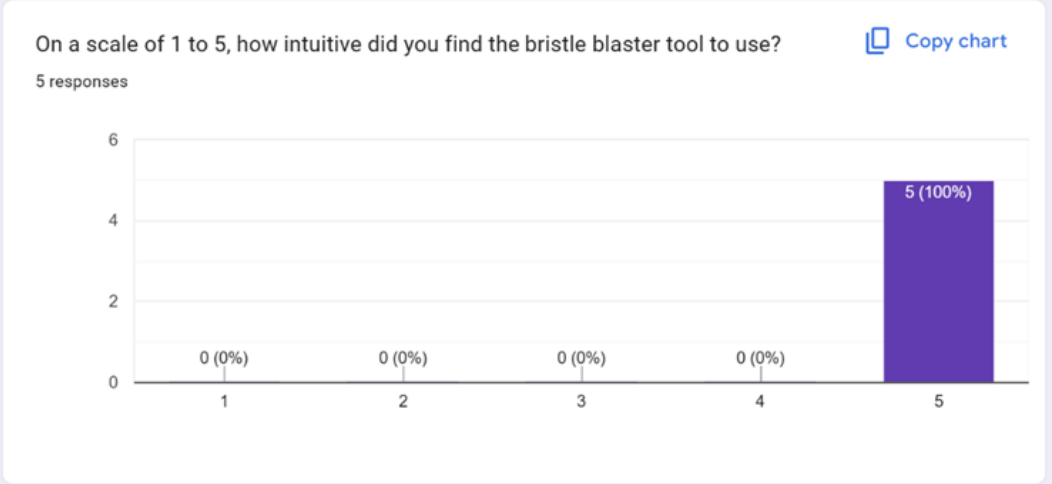


*Figure 23 Full Test - How intuitive was the Bristle Blaster to use*

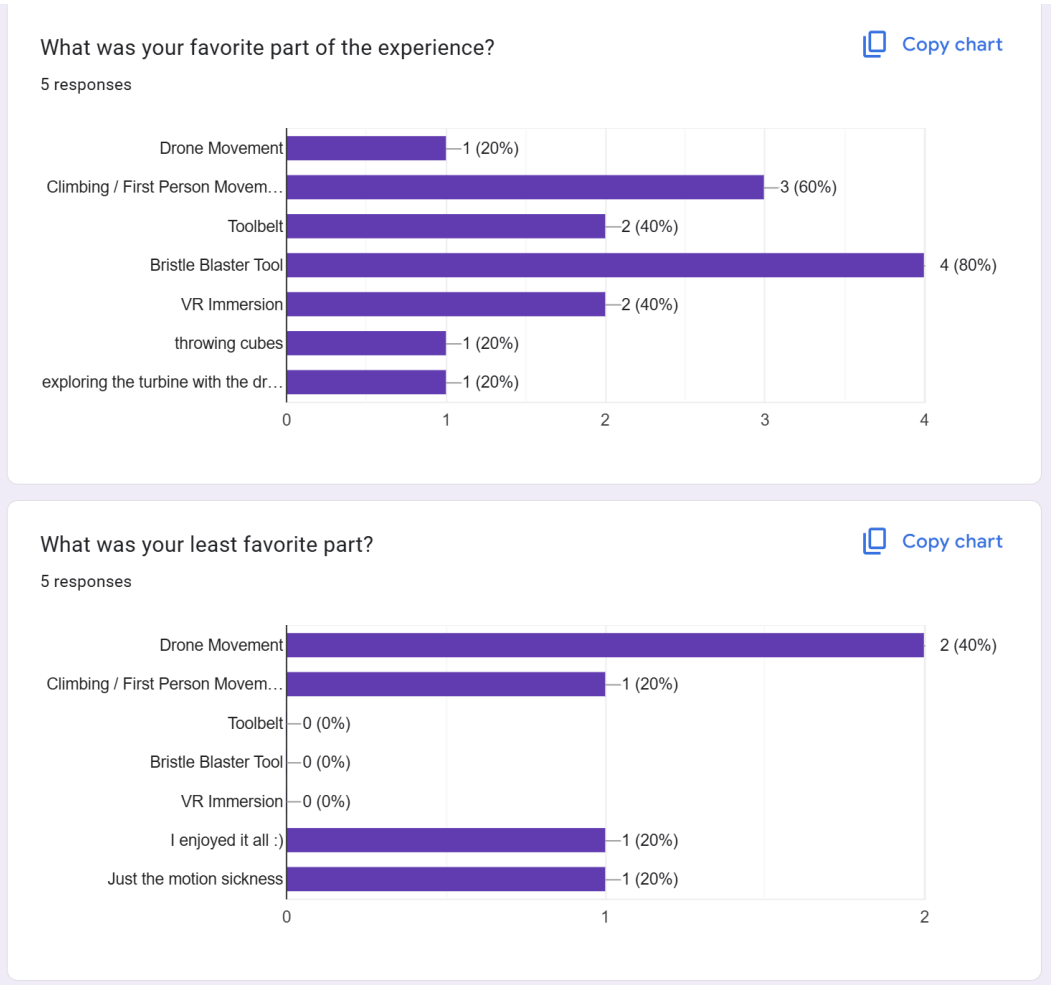


*Figure 24 Full Test - Most favourite and least favourite Experience*

# 8. Extras

This section covers additional features added, which helped improve the projects overall quality, but are otherwise unrelated to the bristle blaster implementation.

Regarding the content of this chapter, the text first begins explaining the project setup, as this formed the basis of the simulation. Afterwards, the climbing implementation will be described, being essential for navigating the wind turbine. Following up, the development of outline shaders will be covered, which made it easier for participants to figure out the spot on which maintenance was required. In the end, smaller features will be shown such as landscape shaders, and a toolbelt for items.

## 8.1. Project Setup

### 8.1.1. VR Expansion Plugin

The VR Expansion Plugin had features such as desktop-controlled VR characters, climbing logic which could be extended, and general settings that made VR projects run smoother, all of which we would need to save time and resources *(Figure 25)*.

The team decided to use a newer version of UE, for which no explicit plugin version was yet developed though. As such, importing needed to be done with care, since preconditions in the form of other plugins and assets had to be figured out, which would otherwise lead to the project becoming corrupted. Using the available template project with the plugin included, and iterating over the setup a few times, it was then possible to discover the missing imports and dependencies, to run our project.

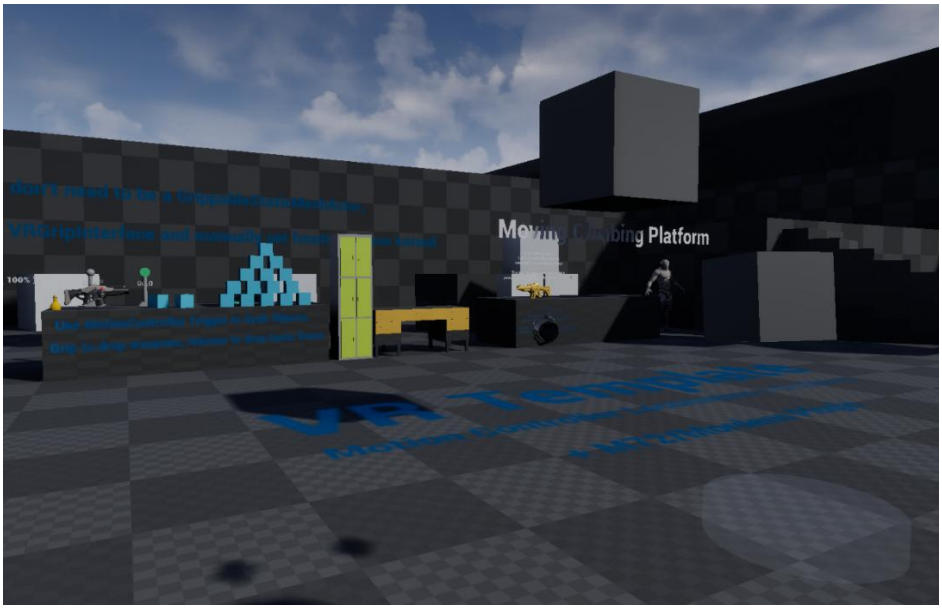


*Figure 25 Template Level showing VRE features*

## 8.2. Extending Climbing

### 8.2.1. Setting up Custom Logic

Climbing was added to make it possible to navigate through the wind turbine's wing, as the maintenance simulation would take place at high altitudes, being tied to a rope.

Though climbing logic was already existing thanks to the plugin, it caused some unwanted behavior such as rope swinging. This is why custom logic handling the velocity was added on top of the base code, where the climbing movement was calculated and initiated. Damping the velocity helped to prevent the possibility of catapulting yourself in a direction, by swinging your arms in the opposite direction, while simultaneously letting go. Additional features were added, where the current height was compared with the lowest recorded height, to apply gravity to the worker to make them fall back down to that lowest height. This helped to show that this position is now the new spot where the rope is tied to. Climbing down further would then update this lowest height value.

### 8.2.2. Adding Boundaries & Adjustable Settings

After the first iteration, we had climbing with which it was possible to navigate up and down, as well as sideways, by holding onto a static object and moving arms. Through further testing, we then discovered an issue where participants would climb up too high, essentially climbing on top of the rope. As such, height boundaries were added, where the workers wouldn't be able to climb up any further. Combined with the workers falling back to their original altitude after climbing up, this would result in less confusion, as it is made clear in which direction they should move towards. One issue that remained though was the scenario of the workers climbing further down than necessary, accidentally skipping over the patch to do maintenance work on. For this reason, a button was implemented that pulled up the worker using more velocity logic, also resetting their original height used for the fall down feature.

All values such as the maximum height, the fall down speed, and toggles that made it possible to activate or deactivate these new functionalities, were added for easier testing and tweaking by other team members.

### 8.2.3. Rope Climbing

For increased immersion while climbing, a new, physics-based rope was implemented, and made compatible with the existing logic. The rope itself was developed by one of the designers of the team, using splines combined with physics constraint configurations. After applying code conventions and optimizing the logic of the newly added logic, the next step was to make it work with climbing. This posed an inherent issue however, as the climbing of the VR Expansion Plugin only supports static meshes, and not moveable, physics-based objects. As such, new logic needed to be implemented into the VR character, which allows such movements, while keeping the rope intact.

This implementation went through several iterations, as some approaches turned out to be very problematic. At first, climbing was attempted to be developed, by using the already existing functions of the VR Expansion plugin. This has caused issues, however, where the rope was moving and stuttering, due to the engine trying to resolve two different movements, that at the same time rely on each other. A solution to this was to distribute responsibilities between those two dependent objects, so that each of them affected only their own unique axis directions. As such, the rope was responsible for the side axis, meaning X and Y, while the player only resolved movement for the Z axis. This resulted in the correct and expected behavior when climbing the rope, even if the worker was now restricted in their freedom of movement, when using the rope. Additionally, further improving the immersion, climbing itself was turned to be velocity-based, instead of transform-based, resulting in a better feeling of weight *(Figure 26)*.

As testing was done, results also showed that the fall-down mechanic and the boundaries set for the maximum altitude, sometimes even led to confusion. In response, the previous additions were turned off, only keeping the logic that prevented the player from catapulting themselves. Thanks to the customizability features added in the previous iterations, this could be done by simply turning off the dedicated values for them.

*Figure 26 Rope in use*

## 8.3. Outline Shader

### 8.3.1. Creating a Component

A certain patch of the wind turbine wing would be subject of the simulated maintenance work, and to make it easier to find this exact patch, some sort of indicator needed to be implemented. As outlines could be used for more than just the patch, it was decided to create a reusable solution using components, with which one could outline any chosen mesh. By following a tutorial about outline implementation in UE (University, 2024), a new material was created, that could be applied as overlay, onto any object. This base was then further developed, to support reusability within other classes, as well as a timer that allowed delayed activation upon entering the area. A setup function holding all variables as parameters, such as the detection collider or timer length, was added to the blueprint component, to make it easily configurable.

### 8.3.2. New Post-Processing Outlines

Though the original outline shader worked as intended, it had a major downside. The edges of the wind turbine maintenance patch used for the shader were seamless, which made the outline Z-fight with the rest of the wind turbine *(Figure 27)*. This made the patch look bugged, leading to a decrease of immersion for the application, which is why an alternative approach was researched instead.

The shader previously created was dependent on the visibility of the object, and what was needed instead was an outline shader that was able to paint through walls. A new material shader based on post-processing offered to be a solution for this, since it held more configuration nodes for exactly this purpose. Following a tutorial made development of this new outline easier (Games, 2024), and all that needed to be done, was to make it work with the component and timer logic created, in the previous iteration of the outline. For this, material references were replaced, and instead of setting the overlay texture of the owner of the component, the custom depth stencil value would be set to either zero or one. This would then result in the wished-for outline, provided the current level holds a post-processing volume *(Figure 28)*.

*Figure 27 Old outline shader Z-Fighting*

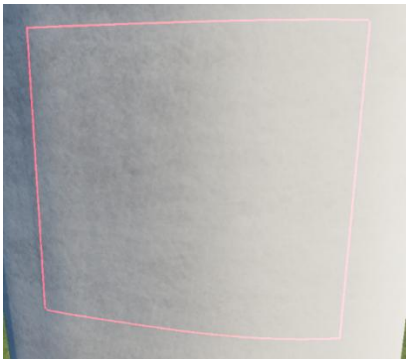

*Figure 28 New Post-Processing Outline Shader*

# 8.4. Polishing

## 8.4.1. Belt Implementation

The bristle blaster needed a checkpoint, to which instruments like itself could return to, in case it gets tossed away by accident. For this reason, belt logic was implemented inside the character, which manages all the items holding a specific tag. The belt itself would only consist of varying amounts of slots, determining how many items can be considered as belt items.

At the start of the level, all items are gathered and sorted into the slots mentioned. If more items exist than slots, only the ones called first would be assigned as such. If the belt object was grabbed, it could be used as usual, and when letting it go, a timer would start, which would reset the item back to the slot position on finish.

This was the basis for the logic of the belt, which is simple in the concept. The actual implementation was not though, as different methods of position manipulation had to be tested at first, combined with certain physics settings such as damping, mass, and velocity. The first iteration of the belt was transform-based, using dynamic reparenting of objects. Especially the latter turned out to be difficult to handle though, as a lot of unexpected behavior made the objects interact in unpredictable and difficult to explain ways. The transform logic on the other hand worked fine but still resulted in less dynamic and more static movement of the instrument. In response to all those newly found insights, the system was developed further without using dynamic reparenting, and by setting the position in a velocity-based way instead of transform-based, utilizing the physics engine of UE. After fixing smaller bugs that were found from testing session with other students related to resetting, the belt was finished and ready to be used *(Figure 29)*.

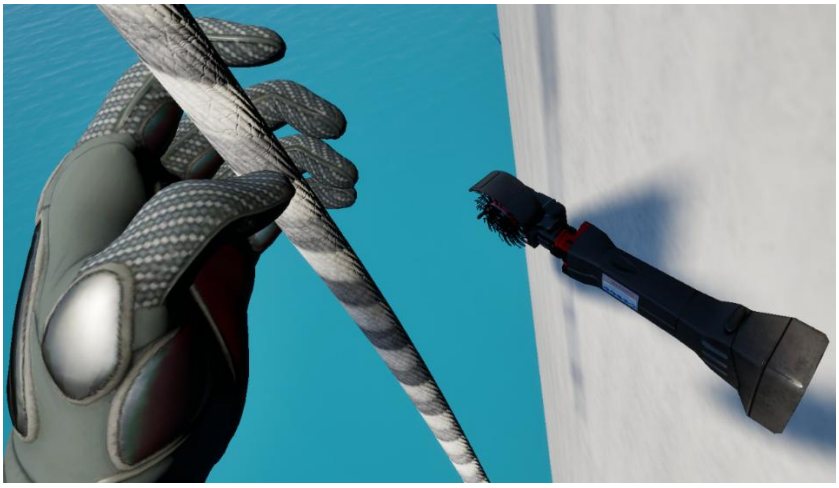

*Figure 29 Bristle Blaster attached to Worker as Belt Item*

### 8.4.2. Improving Landscape Diversity

A landscape material was created to increase the quality of the simulation environment. This was done by one of the designers of our team, and made the level created look closer to a finished part of the simulation. However, the textures themselves looked very repetitive in their color and patterns, resulting in a cartoonish look *(Figure 30)*. As we were aiming for a more realistic appearance, it was decided to bring in some variation into the landscape, using shader logic.

Following a tutorial (Academy, 2023), macro variation was implemented into the material of the landscape, allowing configurations to be made on values such as the color to be blended with, the noise texture applied on top, tiling of different environments and more. Since these configurations needed to be applied to multiple textures, this logic was created as a material function asset, to make it easily reusable and configurable, while also maintaining a clean structure inside the graph. In the end, the result of this was a landscape which looked more realistic, thanks to its newly added variations *(Figure 31)*.

*Figure 30 Landscape with no Macro Variation Material*

*Figure 31 Landscape with Macro Variation Material applied*

# 9. Conclusion & Discussion

Looking at the results of the testing sessions, the bristle blaster simulation turned out to be a success, being the most well-liked implementation among participants, together with the climbing feature. Special attention was also paid to haptic feedback and its reusable logic, all of which also served to answer the research questions of this paper.

By using different scales for activation, collision, and transition, it was possible to implement adaptive vibration feedback into the project, which could easily be configured, by changing the exposed values of the developed class in UE. Since the engine already possesses features with which it is possible to apply basic haptic feedback, the focus was put towards how to make it adaptive, based on its surroundings and the interactions done, using the scales mentioned.

To keep the logic of the bristle blaster and its haptic feedback usable for both HM and DM, the logic was turned into a blueprint component, which is an asset of UE that can be reassigned to any blueprint class. By creating a proper setup function called on simulation start, which replaced all the instance data, this component was made manageable and configurable.

In conclusion, by iterating over the functionalities of the bristle blaster multiple times, using the feedback gathered as basis, the instrument was able to get closer to realism, within the boundaries of using VFB and the hardware provided. As reusability was kept in mind from the start of development, all related features were implemented using modular logic and easily replaceable variables. All this contributed to the clean architecture of the bristle blaster system implementation, within the environment of UE.

# 10. Recommendations

As an engineer you will always be confronted with the question of whether to implement a new functionality using C++ or blueprint, when using UE. Though it is more efficient to do so in C++, especially prior to building the project, this also comes with a substantial downside in terms of workflow for other team members, as well as potential risk for memory loss, if not handled with care. Functions implemented in C++, which are then used in blueprint for example, may result in errors if not built prior to launching the project. As such, it is recommended to have close communication with other team members working on the project when it comes to implementing new C++ scripts, to prevent the loss of time and resources. Otherwise, blueprint is recommended as a well-supported way of programming inside UE, which shouldn’t be neglected in favor of the typical programming environment.

Furthermore, it is recommended to hold testing sessions with people who have experience in using the real bristle blaster. Though the results of the testing done with others were mostly positive, experienced participants may lead to a wide range of new insights, possibly enhancing the bristle blaster experience. Additionally, the assessment score for questions should be doubled, going from a maximum of five to ten instead, to gather more precise data.

Implementing FFB features using compatible hardware is also something that should be considered for future development. Especially for the drone, this would make the interaction more interesting and unique, as opposed to using the bristle blaster in HM. For this, a closer look should be taken at hardware from Moza, as they provide software which can be accessed and adjusted using the developer mode access. This should only be done with enough time at hand however, as UE does not currently support FFB features, and low-level programming will be required to make it work as intended.

Lastly, when dealing with adaptive haptic feedback in UE, it is important to keep in mind, that different people may experience the sensation in various ways. Subtle differences, also regarding the type of vibration feedback,

may be noticeable for some, while for others it feels the same. As such, it is important to hold regular testing sessions with different people, when it comes to implementing haptic feedback. This ensures that the intended effect is noticeable, even for participants with less sensitivity.

# 11. Reflection

The team followed a well-structured planning with a good set of priorities, leading to us being able to polish most of our implementations. Additionally, with the help of good communication, everyone was well-informed about the current progress of the project, available tasks and encountered challenges. Regarding myself, I sometimes had difficulties though, getting used to creating an issue for new additions first, before starting to work on it. Throughout the project I have developed a habit of doing that most of the time now, which is something I will need to keep doing in future projects, to assist in maintaining the same quality of teamwork.

Obvious improvements that could have been made are the tests conducted for my research. By preparing and carrying them out sooner, more data could have been collected and possibly be used to acquire clearer results, which could have been used to better showcase the effectiveness of the haptic feedback implementation. This would have been especially useful regarding A/B testing, as having conducted it sooner would have made it possible to have more participants available for these sessions.

Looking at the development process, though reusability was always kept in mind, there were also instances where implemented functionality needed to be converted into blueprint components, from priorly developed blueprint classes. This could have been prevented if UML diagrams or flowcharts had been created prior to development, to get an overview of how the functionalities would communicate with each other.

Despite those flaws, however, the project turned out to be a success, both in terms of being a product, as well as offering opportunities to grow from personally.

# 12. References


Academy, G. D. (2023, 5 7). *The Secret to Hide TEXTURE REPETITION in Unreal Engine 5: 4 PRO TIPS - UE5 Tutorial*. Retrieved from YouTube: https://www.youtube.com/watch?v=zY8AtjM2Jxg

Aid, T. A. (2021, 9 29). *Render Targets Part 1: Painting Mesh Dynamically in Game // Unreal Engine Livestream*. Retrieved from YouTube: https://www.youtube.com/watch?v=299LOiYvfi4

Arias, A., Moreira, M. T., & Feijoo, G. (2025). *Wind farms through the lens of sustainability and circularity: Integrating environmental, economic, and social dimensions.* Retrieved from https://www.sciencedirect.com/science/article/pii/S2352550925001472

Berrezueta-Guzman, S., & Wagner, S. (2025). Immersive Multiplayer VR: Unreal Engine's Strengths, Limitations, and Future Prospects. *IEEE Access, 13*, 85597-85612. https://doi.org/https://doi.org/10.1109/access.2025.3570166

BGB. (2024, March 27). *Wind Turbine Maintenance: A Complete Guide*. Retrieved from BGB: https://www.bgbinnovation.com/knowledge/news-and-articles/wind-turbine-maintenance-guide

El-Wajeh, Y. A., Hatton, P. V., & Lee, N. J. (2022). Unreal Engine 5 and immersive surgical training: translating advances in gaming technology into extended-reality surgical simulation training programmes. *British Journal of Surgery, 109*(5), 470-471. https://doi.org/https://doi.org/10.1093/bjs/znac015

Games, M. (2024, 8 18). *Complete Interaction System (Items, Outline Effect, On Hovered & UI) - Unreal Engine 5 Tutorial*. Retrieved from YouTube: https://www.youtube.com/watch?v=imeEPY8RjWI

Gibbs, J. K., Gillies, M., & Pan, X. (2022). A comparison of the effects of haptic and visual feedback on presence in virtual reality. *International Journal of Human-Computer Studies, 157*. https://doi.org/https://doi.org/10.1016/j.ijhcs.2021.102717

Kasinathan, P., Pugazhendhi, R., E. R., Ramachandaramurthy, V. K., Ramanathan, V., Subramanian, S., . . . Alsharif, M. H. (2022). Realization of Sustainable Development Goals with Disruptive Technologies by Integrating Industry 5.0, Society 5.0, Smart Cities and Villages. *Sustainability, 14*(22), 15258. https://doi.org/https://doi.org/10.3390/su142215258

*MetaSounds: The Next Generation Sound Sources*. (2026). Retrieved from dev.epicgames.com: https://dev.epicgames.com/documentation/en-us/unreal-engine/metasounds-the-next-generation-sound-sources-in-unreal-engine

Nicholas477. (2023, 9 15). *Asynchronously Reading Render Targets Using Fences*. Retrieved from nicholas477.github.io: https://nicholas477.github.io/blog/2023/reading-rt/

Omara, A., Nasser, A., Alsayed, A., & Nabawy, M. R. (2025). *Remote Wind Turbine Inspections: Exploring the Potential of Multimodal Drones.* Retrieved from https://doi.org/10.3390/drones9010004

Pretorius, L. (2024). *bird.marketing/blog*. Retrieved from Bird: https://bird.marketing/blog/digital-marketing/guide/ux-design-principles/use-feedback-loops-ux-design/

Rowland, D., Davis, B., Higgins, T., & Fey, A. M. (2024). Enhancing User Performance by Adaptively Changing Haptic Feedback Cues in a Fitts's Law Task. *IEEE Transactions on Haptics, 17*(1), 92-99. https://doi.org/10.1109/TOH.2024.3358188

University, U. (2024, 6 2). *How To Hightlight Any Object With An Outline Unreal Engine 5 Tutorial*. Retrieved from YouTube: https://www.youtube.com/watch?v=4-a8QZIf1Dc