ReLive - Drone Mission Review in Virtual Reality

Ricardo Mota
ricardo.r.mota@tecnico.ulisboa.pt

Instituto Superior Técnico, Lisboa, Portugal
May 2019

Abstract

Drones are becoming more common and widespread, with this the need to review their actions also increases as problems inevitably occur during their operation. These problems could be caused by a malfunction in the unit, human operator error or even caused by harsh environments. As such, there is a growing necessity of having a system that allows the users to investigate the root cause of the fail and help to correct it.

With a review system it's possible to get an overview of what the drone did and replicate it. We can also show information captured by the drone's sensors at a specific moment in time to better understand what was happening.

In this thesis we research, prepare the building blocks and develop a post mission review system applied to flying drones with focus on the user experience and interface, to better assess what happened and when, throughout the mission. The opportunity to create a new replay system allows us to experiment with a review system in an immersive virtual reality environment, that is not restricted by the small size of current monitors that clump together information in a single screen. In a virtual world we can overcome this due to the near unlimited space there is to organize our data and virtual screens with no need to spend time or money upgrading our real office space and monitors.

Keywords: Mission Review, Drone, Virtual Reality, Interface

1. Introduction

This thesis is focused on the replay mission interface of flying drones and the way you can review its actions and behavior after it has returned from the mission. The new system that will be used to review drone missions after they are concluded will be in a head mounted virtual reality (VR) environment, this work was proposed by Tekever, to research how post-mission review of their drone fleet would work in a different technological environment.

Tekever is a Portuguese tech company that specializes in hardware and software needed to create and develop manned or unmanned aerial vehicles and new technologies from the ground up.

1.1. Motivation

While on a mission the drone collects many types of data like the drone location, speed, video from the cameras, infrared data from the sensors, signals from ships, and many more that can also depend on the drone model.

These flying drones are used to survey the coastline, forests, and borders in partnership with national and European entities, the tasks performed by the drone are called missions, they can be as simple as sending it to a set destination or follow a passing ship detected by the camera.

The information captured during the mission such as video from the cameras and data from the sensors, can be reviewed afterwards to help drone operators revisit important moments, train new operators and detect and debug possible future or past problems that might not have been noticed the first time.

Nowadays flying drones are becoming more common and widespread. With this proliferation, errors during the usage will inevitably happen, these problems could have a more technical nature (a fault with the unit itself), a human nature (the pilot can cause the fault) or a hazard nature (the environment itself is an obstacle). As such, there is a growing necessity of having a system that allows the users to investigate the root cause of the fail and help to correct it.

1.2. Problem

There needs to be a way to review the state of the drone at any time during the recorded mission without losing context or information about that moment, it could be before or at the time of an incident to determine what happened or went wrong,
so a comprehensive replay system is needed that allows us to move freely backwards and forwards in time to better understand the circumstances of the drone.

The systems of today are full of information from the many sensors but that can be overwhelming to the user if he’s not sure from the start what he needs to look for in the replay, the user might not quickly find what he wants to review and that could be a problem if a time sensitive issue shows up, the two dimensional overview nature of the current system and its many graphs and parameters does not help the user to intuitively find what he needs.

Tekever’s current mission replay system is viewed on a single screen and uses a standard graphical user interface divided in four parts, showing the drone camera, a 2D map with the drone location, one with evolving data plots that show a given parameter as time moves forward and a list of all possible parameters that will be shown as data plots (see Figure 1). This system uses different files for the video and metadata of the sensors, the data time precision is not very accurate so it’s harder to properly review a mission but a revamp of the data gather and encoding is currently underway to correct the synchronization issue.

Visual clutter is a big concern in these types of systems because the physical monitor is of a fixed size and to show different types of data at the same time in the screen some opt to use a window that needs to be divided into smaller sized windows or viewports reducing their size and making it harder to analyze the data. Others may choose to create smaller extra windows to display additional information, but it also clutters the user work and visual space and makes it harder to manage different windows without hiding others accidentally or needing to rearrange them after a new one is created.

A possible solution to the visual clutter would be to buy as many monitors as necessary to accommodate the current workflow of a specific user, despite solving the visual clutter by spreading the windows to different screens. This option introduces more variables and possible problems, now each user of the replay system will need extra monitors and that costs money, the number of monitors might not be the same depending on the workflow of the person using it, and the number of monitors might increase as time goes on and new options are added to the program or if the workflow changes. In addition to this it also takes time for the monitors to arrive and be setup properly.

1.3. Goals
The underlying application will need a rework in order to become a new type of replay system. For this, we will have to research and think of new ways to display the mission status and the program as a whole to better help the operators in reviewing their missions, for instance by changing their environment in virtual reality to better suit their needs or the mission.

Our goal with this project is to build a more intuitive and immersive interface for the drone mission replay system, it will be a virtual reality 3D application where the user will be able to replay a mission that has already finished with a timeline control that can go backward and forward in time, show the captured camera feeds and needed metadata, follow the drone’s position in space, a terrain view with contextual information that helps the user navigate and more easily detect anomalies that may have happened during the mission.

This virtual reality application will use a head mounted display immersing the user in the virtual world created by us that can be modified at will or use built in presets, it can be setup so the user feels more comfortable while using it by changing the environment around them to one that’s more familiar or liked by the user.

The main environment will consist of a control center where the user has a giant screen in front of them to monitor the most important part of the drone mission, the camera feed. Relevant information about objects and important locations that are found on the video can be added on top for more contextual information.

At the center we have a map of the current location and a drone position marker, to the sides we put sensor data that can be represented in history graph form or in more metric specific ways that better represent that type of data, for example a rotating compass. Also at the sides we can have mission relevant information in the form of written list logs or visually marked on the map if this information happens to be important points of interest or a trajectory that was planned beforehand (see Figure 2).

Information that is displayed on other virtual screens can be hidden to the side, outside of the
user field of view, so when the operator moves his head sideways that extra data is displayed to him without overloading the main part with more visual clutter, these screens could be set to show up on the main part if certain mission conditions are met as to warn the operator of a status change.

Not only is the visual part important but audio is also important when dealing with virtual reality to create a more believable and immersive environment, the special conditions that triggered the screen changing positions could instead be an audio warning that when played it alerted the operator to relevant information. This could be less distracting to the operator as it would be only a reminder that new information was available and not an active change done to the scene that could disrupt the mission review.

2. Related Work
While doing the research for this project we found many papers and articles that could be relevant, but mainly focused on the one that could help explain or justify the benefits of virtual reality and replayability in different fields.

Early our intention was to gather cases where their objective focused on using technology to help prevent future problems. This was done by showing doctors practicing sensitive procedures with a certain realism factor and feedback [8]. By using a drone to inspect a real disaster zone it was proven that drones can be useful for live damage assessment [6] and that this could be translated to a post mission review to better review the footage and sensor data captured by the drone.

Accidents can be a common occurrence in more dangerous workplaces, sometimes these accidents are known before hand as they have happened before but if a worker as never encountered that specific situation he might not be aware to prevent it, as is the case of the South African mining industry [14] that worked to avoid these accidents by using a simulated work environment that will lead to a mistake or accident and letting users explore the different scenarios to prepare them to better spot what’s wrong in the given scenario and avoid the accident.

Then we searched for work that specialized in replay or review, to get to know already existing solutions and how they are used and can help to develop a replay system application. Our oldest referenced work is from 2001 that is used to explore historic buildings [11] and can let people explore older monuments that do not exist anymore illustrating the use of a review system. A replay interface [13] is helpful to the user by letting him find and assess a situation with a quick glance and determine that where more worrying problems are appearing, also the South African mining industry example [14] discussed before is also a review system example.

Immersive virtual reality work that contain useful use cases and smart implementations that can help shape our own solution, this includes a airfield remote tower concept that features VR application to airfield management [7], recreated virtual workplace environments where the user can in a more familiar place [5] and [9], how to interact with large amounts of geotagged data and explore the virtual world [10], control a virtual ship with different remote people [1] and a virtual reality system implemented in a game to help spectate and replay matches [4].

In the interface side of things, a futuristic inspired application with an interactive interface [2] can help developers realize that many of the previously unused concepts thought to be impractical in an interface are now doable and user friendly. The potential of a visual solution with a higher field of regard can help users in comparison with a normal monitor [12] and an interface with modular components that monitor specific vehicle metrics [3] can be beneficial to users in their customizability.

3. Proposed Solution
A 3D virtual reality environment was created to immerse the user in the virtual world, it consists of a simulated command center workspace where the user has the ability to monitor and replay the drone’s mission.

For our solution we use the Unity3D game development engine, it’s an engine we have used before, it has extensive official support documentation and community support because of its widespread use, it was also a Tekever requirement as the programming language C# is preferable and more compatible with current Tekever’s programming environment where C# is prevalent in many of their developed systems.

With Unity we have the option to export to many different platforms if needed in the future, as this engine has great support for many operating systems and virtual reality headsets, but for now we are only interested in the Windows build version
and we use the HTC Vive as our head mounted display (HMD)/virtual reality headset, in alternative to other HMDs like the Oculus Rift because of previous experiences with both headsets and the Vive providing more flexibility and fidelity when in use and while developing games and applications.

In addition to Unity3D we plan to use the Valve created SteamVR Unity plugin that helps create a layer of abstraction when dealing with Virtual Reality headsets and controllers to help with quicker application development.

Figure 3: Project architecture overview

The mission replay system (see Figure 3) is available after the mission is over and the drone has returned from its planned trajectory and concluded its goals, after a mission is concluded we have access to the mission file (.ts) that contains two data streams, one is the video feed (h264 video stream) from the onboard camera and the other is the timecoded data captured from sensors and payloads of the drone that may be needed while reviewing the mission.

Our project is comprised of many different modules (see Figure 4), each with a specific function that when working together create the ReLive mission review application. It starts by loading the mission file into the application where the video and data is sorted by the Data Handling module according to their timestamp and labeled for easier lookup when replaying the mission and showing a coherent state at each moment in time.

The State and Replay module are responsible for ensuring the accuracy of the current state data structures that are used by the various interface modules to show a visual representation of the scene and data gathered, for example the current video frame, sensor history needed for the graph and the drone location on the map.

A module created to alert the user of important events or changes is aptly named the Events module, and will keep track of predefined or user custom metric thresholds and criteria that trigger a certain event and notify the Interface module that is able to display the alert on the user’s field of view or play an alert sound.

The Interaction module connects to the SteamVR and if necessary the Vive headset and controllers directly to be able to respond to the user’s controller inputs and affect their environment by contacting the Interface module which in turn will need to contact the State and Replay module to alter the current effective time being reviewed and be put in a coherent state (see Fig. Figure 4), as the user can directly change the timeline by pointing and clicking and interact with the virtual screens by adding more, resizing and moving them. It’s also possible to change the state directly by using the controller hotkeys to rewind or go forward in time.

Figure 4: ReLive module architecture

The main use case (see Figure 2 that was illustrated earlier to give the reader a visual image of our concept). It is a control center where the user has a large screen in front of them to monitor the video camera feed of the drone mission, relevant information about the drone, detected objects and important locations that are found on the video can be added on top for more contextual information as to what the drone was recording at the time.

At the center we have the map with the drone location, mission relevant information can be displayed in the form of written list logs or visually marked on the map if this information happens to be important, for example points of interest or a trajectory that was planned beforehand.

Below the central part of the scene there is a timeline showing the duration of the mission similar to a video player timeline, and the user can go back and forward in time by using the controller or by pointing to a specific part of the timeline, this timeline also has relevant information color coded or with icons displayed, for instance if a ship is detected during the mission, instead of needing to look through all the footage to find that particular ship detection, it is parsed beforehand from the available data and displayed in the timeline at the specific time it is found to help the operator find relevant bits faster.

4. Implementation

This section discusses the implementation details and components needed for our solution and how they work together to complete the replay system.

The replay system consists mainly of two parts the interface and the backend that is responsible for data handling logic. Let’s start with the interface:
4.1. Video Player

A big part of a replay system is watching the footage captured by the drone’s camera, so in our implementation created inside a 3D virtual world, it fills a large part of the scene and the user’s field of view when looking in the video screen direction comparable to a cinema screen.

![Figure 5: ReLive’s video player with timeline shown below.](image)

Mission information is used to populate our scene elements (see Figure 5), above the video screen we put the name of the current mission being replayed, and the date and time of day it took place, at the bottom we show the video’s current time and total time the drone recorded for this mission.

Unity’s built-in component handles the video file loading in different platforms for us, but we still need to manage it with a Video Manager script that can speed up or slow down the video’s playback speed, that can jump forward and backward to an exact moment in time, this is called seeking. The module doesn’t support reverse video playback but we can still go back by jumping back multiple times.

Using the video manager’s features we added on screen buttons to control the timeline changing the current time of the video. The timeline is located below the main screen and shows a visual indicator that represents how far along the video playback is, it’s also possible to jump to anywhere in the video by clicking in the timeline.

This allows the user to quickly jump to their preferred time of the video to review a moment and analyze it, greatly combining the timeline jumps with visual events on the timeline, the timeline’s clickable area is bigger than it appears so it is easier to interact but still keeping the smaller look that it has.

![Figure 6: Timeline closeup with event indicators. Symbols showing (left) and minimalistic view (right) (a)](image)

Figure 6 shows different visual event indicators that can appear on the timeline itself to create a visual reference guide to when the events happen relative to other events or the current time. From left to right is the warning event, current time marker, info event, mission start marker and finally a danger event.

4.2. Events

Our solution has different types of time stamped events, these are extra ways to contextualize the captured data that occurs at a defined point in time, stored in the mission file.

The events types are:

- **Mission**: Represents the start of each mission step and helps convey the mission progress to the user.
- **Danger**: Best used for dangerous values that were in the data, where a set threshold has been exceeded and it is worth looking into.
- **Warning**: Can be a good complement to the danger event where a warning occurs before the extreme conditions are met, is also used to show object sightings on the map.
- **Info**: Its purpose is to show timed notes of the mission file and help guide the mission review. Can also serve the purpose of showing objects on the map that may not demand the same urgency as a warning.

They can be displayed on the screen inside the timeline (see Figure 5), in their own Mission List or Event List interface, placed on the map or as visual effects to the available modules in the scene to alert the user of an important change in the data.

4.3. UI Elements

![Figure 7: Mission List Component.](image)

**Mission List** (see Figure 7) is a component created by us that lets the user quickly determine the mission progress in a visual way. It shows the objectives and steps defined for the mission and highlights the current one based on the video player’s
time, this feature takes advantage of the Mission event type. The component also allows user interaction to quickly jump in time to the chosen mission objective by interacting with the mission step entry for a quick analysis of a certain part of the mission.

The Event List (see Figure 8) serves a similar purpose to the mission list but this time for the rest of the event types, it lets the user know what is going to happen next in the mission at a quick glance. When an event is taking place or a threshold is to be exceeded it is shown on the event list, they follow the same color scheme as their timeline counterparts, interacting with the event’s list entry jumps you to that moment in time.

Each event entry on the list shows the time when it happens during the mission. The entry has a short description of the event, or a note that refers to a moment in the mission. If it’s an event generated by a Databank condition it specifies which type of data was responsible for this event and the data value that met the criteria, at the right side of the entry we place the reason for that event and the value condition it exceeded. Currently the thresholds conditions we implemented are: “greater than” and “lower than” of chosen values.

After an event’s time has been passed it still lingers for 5 seconds to give the user context to what happened previously, this helps when jumping through the footage to review other sections of the mission where events that occurred are not instantly hidden.

To not clutter the scene needlessly, the event list is a component that allows scrolling to view all the other events that may not be showing at the start, as they will only happen later in the mission.

A script in the event list component is responsible for checking every event it has and checks to see if it should be hidden at the current time according to the 5 second condition. Also when in the time range of the event it should be highlighted for the user, the highlighted state of active events can be seen in Figure 8.

4.4. Data Elements

Data that was recorded and stored also needs to be shown to the user while he is reviewing the mission. We chose to have different ways to represent data visually because there isn’t a single solution that is best for all types of information.

Our goal is for the user to be able to clearly distinguish which data it is with a quick glance by choosing the appropriate component. It needs to have the data type’s name visible, the units for how it is measured and of course the current value. They can also exist in the form of a graph that shows the progress of those data values over time.

The user is able to place more data elements on the scene during mission review thanks to our object spawner component. This component is responsible for the creation and placement of data objects.

We want to be able to place objects around the user in a circular fashion, so a 3D oval shaped object, sphere that’s being scaled non uniformly, exists centered on the user position and encompasses the scene, then an invisible ray is cast from the user’s position in the direction he was pointing and when it hits the sphere we know where to place our intended object. After that, it rotates to face the user so it can be seen even in every position it is placed even if the user decides to place it directly on top or below his position.

This effect proved difficult at first because of the user’s position being inside the sphere (see Figure 10), the raycast was not being detected and it could be because of the way a 3D object geometry mesh is represented. Raycast hits to the backside of a mesh are not detected, this meant that when inside the sphere every polygon face of the sphere
is the backside to the user, so we had no collisions after casting our ray.

The solution was to use a Unity plugin called ProBuilder that actually works on reversing the mesh and performs geometry operations to objects, it correctly inverted the polygons and the collision was detected.

4.5. Map
An important feature to have is a map so the user can follow the mission, with a birds’ eye view of the location. With the top-down view, we can provide additional spatial context to what is happening on the screen, how the mission progresses and how each event relates to the overall picture and other events.

To have a map inside Unity we used a plugin created by Mapbox, a service that specializes in maps for many different platforms, tools and use cases. It handles the loading and creation of a map in our scene and is customizable, we can choose the starting location and the map tiles around it appear on our map.

It can highlight roads, create buildings or show information about different points of interest in that area and stores terrain elevation data which can be used on the generated map.

![Map during a mission, with two warning events (in yellow).](image)

For our main case, we want the default look, photo captured from a satellite, no need for buildings as they don't look very good outside of large cities, and they would interfere with our placed objects. Mapbox also provides an easy way to place objects on the map given their GPS coordinates, making our life much easier when moving the map and keeping the objects where they should relative to the map.

The drone is placed on the map at the currently recorded location coordinates, we follow the drone in our map to give it focus when the user is not interacting with the map. The user can move the map to explore other areas, instead of the one where the drone is, to gain more insight on the mission.

Other events are also placed on the map (see fig. 11), such as the mission steps locations of which the current objective has a pulsating circle indicator, sightings appear on the map for the time span that they were detected and then are removed when the sighting time limit is reached. A trail is left behind by the drone through the places it passed on the map, making it easier to know the trajectory the drone took.

Besides the main central map, we also have a larger map overview to the side, its purpose is to better show the mission progress if the mission takes place between large distances and its scope can’t be captured by the more close-up central map.

4.6. Controls
Interaction with our mission review system can be done in two ways, the first being with a normal setup of a keyboard and mouse. Pointing and clicking to interact with the objects on the screen as you would on a previous review system, while also being able to use the mouse to look around the scene like you would a game.

This method is still supported because it greatly helped throughout the creation of our program with the scene prototyping, placement of components and with fixing problems during development.

Another method of interaction is with the Virtual reality headset and its handheld controller. Where the user can interact with the world using the controller as a pointer to simulate a point and click but in a 3D virtual world. Looking around in this virtual reality mode comes more natural to the user as he simply needs to move his head to explore the scene, much like in real life.

The mission review is meant to be done without leaving its starting position but the user can also move around the scene if he wants to have a closer look at the scene, for instance, the map.

While targeting a Data element the user is able to change its state with the directional controls, on the controller or the keyboard. We can change its visual style to another element, change the Data-bank it represents or move it.

When the user is not interacting with anything, he is able to control the video player playback speed, playing state and jump forward or backward.

4.7. Data Metrics
A replay system needs to be able to show the state at a moment in time for that the needed information that was recorded and now is going to be displayed needs to be well organized.

The current drone sensors record data at fixed time intervals to preserve storage space and band-
width and we could just develop a system that also replays at fixed intervals but that solution wouldn’t scale well in the future if the drone started to record some sensor data more frequently than others or if it recorded in irregular intervals.

So we decided to go the simpler way and store each data set as it is recorded with its timestamp stored along with data value, making it a more flexible system that supports as many or few data points as it needs. The data set entity we created is called a Databank.

The most common data types that need to be stored are number values, GPS positions, dates and written words, they can be represented by strings, floats and a combination of both.

As different types of data have important values that need to be monitored or taken into account, a Databank also supports thresholds that can be defined and then be used to alert the user that a specific data set might be interesting to look at the time the threshold was exceeded as it might suggest that something important happened. For example: if we are dealing with the altitude Databank, an altitude below a given limit or even 0 meters might be a catastrophic problem for that mission.

4.8. Data Interpolation

We have the data storage structure implemented, but now what happens in between the measured time intervals? See Figure 12.

![Figure 12: Interpolation scenario for altitude in meters (m). A is the previous value, C the next one and B is at the current time.](image)

There’s some options that we can choose from and they all have their use cases, such as:

- **Show no data**: when there’s nothing recorded in that time frame we guarantee that operator reviewing the mission just sees what the drone actually recorded, for example if tracking a ship’s beacon we see the actual position and timing when it was detected with no lingering effects.

- **Show the most recent recorded value**: like the previous option the operator also sees real values that were recorded, it could be used to always show a ship’s last known position or for instance show a diagnostics sensor last recorded error rate

- **Show the linear interpolated value between the previous and next value**: this option creates possible values between two real values, proportionally to how closely we are to one of the points, providing a smoother experience for the replay user

- **Develop a more complex interpolation method**: this last option is the one with more potential for future work as a better interpolation model can be modeled specifically with a data set in mind could produce better and more coherent resulting values for that data in particular

When the mission is created it’s possible to choose which method will be used for that data type, with the default being the linear interpolation (lerp), it provides the smoother experience with no jumping values from time to time.

5. Evaluation

A review system will deal with many different types of missions, some that involve a larger volume of data to be processed than originally intended.

So to evaluate this project’s implementation and how it could scale with different amounts of data being processed and displayed at a time. We will test its performance when dealing with a scenario where the user wants to see the data displayed simultaneous on the scene.

This will also help us understand how the program deals with more extreme use cases and to figure out which future upgrades and feature additions are feasible on the current solution or if they need modifications before being implemented.

5.1. Additional Screens

Our current solution has a single video screen, because of the requirement that the drone only has one camera to record the footage. In the future different drone models could be used that have more video feeds recorded, so we test how adding a screen affects the frames per second (fps) of our program.

Unity’s video processing implementation is very taxing on the system’s processor. Comparing it to a popular media player program, VLC only uses about 10 to 15% of our system’s CPU to play a video. Where unity uses 20% of the CPU without the video, and an extra 30% just to play our video, making it near 50% total when running our whole scene. VLC is close to a third more efficient for playing video than unity, so a future VLC integration with Unity could prove beneficial.

In the future, if more than four videos need to be played in the scene at the same time, a different solution needs to be considered. Either not using unity’s own implementation, or scaling a less important screen to a less performance impactful resolution.
It's common for monitors to have only 60 hertz (Hz) or update 60 times per second, with VR it's said that to have a smooth experience a minimum of 90Hz is required. Making Unity's attempt to limit our update rate to 60 times a second not suitable for VR or testing. Fortunately, the community provides scripts to try and unlock the frame rate.

<table>
<thead>
<tr>
<th>Video</th>
<th>Total Time</th>
<th>Size</th>
<th>Resolution</th>
<th>Average FPS</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>1 minute</td>
<td>90 MB</td>
<td>1080p</td>
<td>199</td>
</tr>
<tr>
<td>B</td>
<td>5 minutes</td>
<td>1.8 GB</td>
<td>1080p</td>
<td>187</td>
</tr>
<tr>
<td>C</td>
<td>2.5 hours</td>
<td>2.24 GB</td>
<td>1080p</td>
<td>208</td>
</tr>
<tr>
<td>D</td>
<td>2 hours</td>
<td>3.11 GB</td>
<td>720p</td>
<td>212</td>
</tr>
</tbody>
</table>

Table 1: Video files used

Now we will start testing how the application handles different types of videos. These tests were done on the editor to speed up the testing but the percentage differences when tested with a compiled program build were in the same range.

When playing the videos, the average frames per second recorded were not perfectly stuck to that value, they oscillated around that value but could be 5 to 10 frames above or below at any given time.

Jumping ahead in the timeline has an immediate performance cost at the moment the skip is made, the performance drop range is 50-70 fps, then it instantly recovers (within 20 ms) to the normal recorded range.

Looking at Table 2, video file B is the densest file meaning each frame has more image information. Unity prefers displaying a less dense video file, and is able to run faster and that's what we see in the recorded performance information. It also prefers lower resolution files, such as the Video file D, but in this case, the performance gain is not that significant.

We also tested putting the files in different types of storage disks, predicting that the file on the faster SSD would allow the program to run faster than one stored in the slower Hard Disk Drive (HDD). But no noticeable difference was measured in their CPU usage or frames per second.

Table 2: Performance stats of one vs two screen

<table>
<thead>
<tr>
<th>Number of Videos</th>
<th>CPU Usage %</th>
<th>Average FPS</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>46 to 53</td>
<td>215</td>
</tr>
<tr>
<td>2</td>
<td>60 to 66</td>
<td>200</td>
</tr>
</tbody>
</table>

Extra video player screens are a possible improvement path in the future so we tested to see how another different video affected our program performance. Adding another screen increased the CPU load by 10% and there was a decrease of about 15 fps, a 7% decrease. If there's a need to replay footage from different cameras, using the unity video approach we can get to 4 screens while maintaining a smooth performance.

Regarding screen size, despite already having a large screen we can't predict if it's the perfect size for all future missions so we tested changing its size. We scaled it to 10 times the normal size, testing a very extreme possible scenario and it only reduced our performance by about 15 frames per second. So if needed we can increase the screen size without worrying about causing the performance to degrade and reach unpleasant levels.

At the end of this round of testing we ran some of the tests to compare the results to a compiled build version and the performance increased, it added 80 to 130 fps depending on the test case, making it even better and nowhere close the 90hz undesirable limit.

5.2. Additional Data Elements

Databanks are used to store the recorded sensor information and after they can be displayed on screen with a DataElements component. In this section we will test how adding more interface components affects the performance of our program.

Our scene already starts with 6 data interface components, we created a grid that spawns the desired amount of components. The first test will spawn 25 elements in a 5 by 5 grid, and the second one will have 49 elements in a 7 by 7 grid (see Figure 13), this last case is an extreme case that's unlikely to happen as that amount of elements on screen at the same time is very distracting. It is used to assess the performance change when adding extra elements.

Each element is requesting its interpolated value one time per frame and updating itself on screen.

When we test the 25 grid array we get an average of 160 FPS, changing to the 49 grid we get 110 FPS, both are around 200 FPS when the visual elements are disabled.

These values we are seeing, are starting to get close to the 90 FPS limit we talked about earlier so we check the compiled build version to see if
we would have problems and the values increased significantly. The grid with 25 elements increased from 160 to 250 FPS and the 49 elements one increased from 110 to 185 FPS, out of the lower intended performance limit and into a smooth stable range.

5.3. Data Frequency
Data frequency is also a parameter which can be tested, how many drone data points do we store per minute. First, we tested with 10 per minute, having real information recorded once every 6 seconds, then we tested with 60 points per minute, or once per second and the performance stayed the same.

This could be because we are already interpolating values many times per second, and having more points of real data doesn’t affect how many we need to consider to interpolate. It just changes which two points we use when interpolating, but having a faster recording will mean we can display more accurate data if the drone can record it that fast with no loss to performance.

Different source videos from Table 1 did not have a significant impact on the performance of data frequency based on the video file time, not considering the already measured impact from the previous testing.

Moving the map only has a performance impact when panning results in an empty tile that needs to be fetched and loaded from the server. This loading of tiles lowers the performance into undesirable levels below the 90 FPS range into the 50s or even small hangs, but it’s the way Mapbox loads new information so we can’t avoid it.

In this section we tested different scenarios that could expand our application’s functionality, by adding more video screens, increase the number of data elements on the scene and raise the data recorded rate. All of them proved successful while keeping the program functional and within satisfying performance levels.

6. Conclusions
This project started with the challenge of developing a new flying drone mission review system in virtual reality as an alternative to the current simple non-3D window graphical user interface applications. We were able to list some limitations of the current solution and ways to improve it with our solution.

Our application was created with a virtual world in mind, and the user is able to review the footage captured during a mission, while having an overview of its progression via the Mission list that tracks the objective progress. The Event list is used to observe detailed information about coming events and the central map for a positional overview on how each event relates to one another and how far along the drone has come since the start.

The video timeline gives additional insight into the mission, as the user knows how long it will take to complete, but also for the information that an event will happen around a certain part of the mission just from a simple symbol placed in the timeline.

Interface elements were created to facilitate the examination of different types of sensor data and can be arranged in a way that better helps the user in their review.

Data interpolation is used to smooth out parts of the mission review where data might not have been collected or the interval between each data point stored is too large and a stuttering review experience is avoided.

Putting it all together, we have a review system that is able to handle increasing amounts of data, update their values hundred times per second while displaying the values and graphic elements changing on the scene at the same time. It also deals well with small and large file video files where total times can reach hours with no noticeable performance difference.

Our solution has the capability to be expanded with future iterations by having more video screens with different files playing. It can support faster-stored data interval rates where the time gap between data points stored is much smaller, preparing it for a future where sensor recorded data requires a more precise measure.

References


