Optimal Control Frequencies for Large Number of Virtual Agents in Augmented Reality Applications

Bradford A. Towle Jr.[*]
Florida Polytechnic University, Lakeland, FL 33805

Abstract

With the rise of augmented reality (AR) applications, more devices of different computation capabilities are employing this technology. Currently, AR applications are limited to a small number of virtual agents, but in the future, this will change. A virtual agent is any entity within the program that must periodically run logic and has some graphical effect. This journal article explores the optimal control frequencies for virtual agents across three common AR platforms and compares the results. This experiment uses a stair-step stress test and records the framerate at a 10Hz cycle. Special care has been taken to reduce extenuating factors that may consume CPU cycles, isolating the only change to the increase in virtual agents. These tests were run five times for six different frequencies on the HoloLens 1, HoloLens 2, and the Android Note 8.

Key Words: Augmented reality, AR, framerate, MRTK, unity3D, HoloLens.

1 Introduction

Augmented Reality (AR) is a promising new technology making large gains in how people interact with computers and mobile devices. Augmented reality allows a computer program to combine computer-generated and real-world content [12]. This combination is usually accomplished by adding to an image or video taken in the real world from the user's perspective. The technology is expanding rapidly with subtle integration into everyday uses, especially in mobile applications. Three main reasons for this expansion are emphasis on reality, mobile device CPU improvement, and unique visualization capabilities.

The most crucial distinction of AR is that it does not occlude or replace the user's environment. Virtual reality (VR) will enclose the user in a completely virtual environment and capture all their visual/audio senses. Augmented reality, enhances the real world, allowing the user easier interaction with people and their surroundings. This feature is most noted with social applications as people begin focusing on real-life interactions and moving away from the computer screen. Another element fueling the AR enhancement is the increase in mobile computing capabilities. Smartphones now have enough processing power to run AR applications without noticeable delays. This means every new mobile device can now be used as a platform for AR. App developers have noticed this trend and begun incorporating AR elements into their programs, thus creating a subtle shift toward the widespread incorporation of augmented reality. The last reason for AR expanding popularity is the unique capability of integrating real-world items into the program.

While not all devices are capable of this yet, many AR devices can scan the surrounding objects, allowing them to affect the program's outcome. This scanning capability, coupled with the vast visualization potential of AR, has made it popular in the medical field for training, simulation, and a visual reference for doctors.

Since AR has become popular due to the aforementioned points, large game engines, including Unity 3D and Unreal, have championed its development. While this software is well optimized, the programmers that use them only sometimes employ maximum efficiency within their code, especially compared to hand-coding a device driver from scratch. This phenomenon is especially true for control scripts of virtual agents. Virtual agents are any virtual entity that must be updated regularly and have some graphical effect on the application. This definition can include characters, user panels, or visual effects.

To date, this has not been a pressing issue, as most AR applications employ small numbers of virtual agents. However, as the field of AR expands, the number of virtual agents will also scale. This increase in virtual agents could be problematic since larger CPU usage corresponds to lower framerates; lower framerates correspond to motion and simulator sickness in both AR and VR. This paper outlines a stair-step test to determine the optimal frequency for controlling virtual agents across three different AR platforms. The paper is broken down into the following sections: related work, platform description, testing methodology, problems encountered, results, future work, and conclusion.

2 Related Work

This paper considers virtual agent to mean any projected visualization in an augmented reality application that must change, move, or adapt to outside stimuli, thus, requiring a control script to function. The term virtual agent conjures

[*] 4700 Research Way: ARC 2230. Email: btowle@floridapoly.edu.

mental images of brightly colored cartoon characters running around. While such virtual agents have been used in AR applications to assist with user interaction [3, 4, 6, 7, 11], not all virtual agents are 3D game characters. Many virtual agents will be informational elements that change and adapt. For example, a virtual agent may be nothing more than a label or text message that appears identifying a desired product in the grocery store [1]. Another example in the realm of health applications is visualizing different organs during surgery or education [2, 5, 10]. This visualization may need to change, update, or provide some form of response due to the user's action requiring some form of control script. Another important field in AR is bridging the gap between robotics and humans [8, 9]. These applications will employ virtual agents to represent robots or robot-human scenarios and goals. Again, these informational virtual agents will need a control script to update, move, and adapt to different input from the user.

Currently many of the augmented reality applications only focus on one or two virtual agents at once. Therefore, processing power for their control scripts is not a large concern; however, as the field grows expanding the scale of these applications, it is foreseeable that an AR application may have hundreds of virtual agents running simultaneously. Due to this expectation, the experiment described in the next section seeks to determine the best frequency to control virtual agents.

3 Platforms

Three different platforms were chosen for the stair-step agent test:

- The Android Smart Phone
- HoloLens 1
- HoloLens 2

3.1 Android Mobile Device

Mobile devices and smartphones have become ubiquitous within society. The devices can now run augmented reality applications, and game engines can build applications for these platforms. This capability means that the general public now owns a device capable of AR; therefore, its performance with a large number of virtual agents should be tested. The Android phone tested was the Samsung Galaxy Note 8. This was purposefully done to represent a newer phone, but not the newest one on the market. Any performance issues observed with this phone would indicate a potential systemic problem for AR applications on modern phones.

3.2 HoloLens 1 – (PC Architecture)

The HoloLens 1 was one of the first head-mounted-display (HMD) AR platforms. It was also one of the first to have spatial awareness. A powerful feature where the device can map the physical environment and use that to occlude virtual objects. This device uses a typical x86 PC architecture and provides the next generation of AR functionality, such as gesture recognition, spatial awareness/mapping, and accurate localization for user position. The device is fully supported by Unity 3D game engine and is a solid baseline to compare against newer HMDs.

3.3 HoloLens 2 – (ARM Architecture)

Arguably one of the most advanced HMDs available and has a generational improvement in capability compared to the HoloLens 1. It is one of the best technologies available for AR applications and uses the ARM architecture instead of the standard x86 PC architecture. Any problems observed with the Hololens 2 would indicate that current hardware is not capable of running a large number of virtual agents in an AR setting.

4 Testing Methodology

4.1 The Experiment

The experiment outlined in this paper was run on all three platforms. The program used in this experiment was written with Unity 2020.3 and used the Mixed Reality Tool Kit (MRTK 2.0). This program would create a new virtual agent at a frequency of 2 Hz for five seconds and then wait for five seconds to determine if the system was stable. The above sequence of spawning and waiting would repeat until there were 100 agents, during which the frame rate was recorded to a file at the frequency of 10 Hz. The frame rate was calculated using the unscaled delta time property to provide the most accurate values possible (Equation 1).

UnscaledDeltTime is an independent interval in seconds from the last frame to the current [13].

$$FrameRate = \frac{1}{UnscaledDeltaTime}$$

Equation 1: Equation used to calculate framerate

The program recorded the framerate every tenth of a second and kept the file writer open to minimize computational overhead. The program would only append information, never delete, or search through the file. This limitation with the file handler was explicitly done to minimize its computational load on the hardware.

The virtual agents were programmed by employing best practices with Unity 3D; however, no other optimization was done. This programming style imitated a typical game programmer and not necessarily a researcher in computer science. The reason for imitation is to ensure the test script represents a typical program written for this platform.

When a virtual agent was created, it was given a team: red or blue. The agent invoked a control function called handle update, which provided the control algorithm.

Each agent ran the same function to control themselves. However, the frequency this function invoked varied throughout the experiment, and the resulting framerates were compared. Five individual tests were administered for each following frequency:

- Update (once per frame)
- Fixed Update (20 Hz)
- 20 Hz coroutine
- 10 Hz coroutine
- 5 Hz coroutine
- 2 Hz coroutine

The control function performed the following tasks:

1. Control the nav-mesh agent.
2. Fire projectiles at the enemy team.
3. Orient and update the score panel.

The test had a large arena where there were sixteen pre-determined points the virtual agents could move. If the virtual agents were within 6 centimeters of the goal, it would then randomly choose a new goal and start navigating toward it.

The nav-mesh system in Unity was used as it is a common and popular tool amongst developers. This nav-mesh path-finding system is well optimized and would likely be chosen over building a path-finding algorithm from scratch. Please note, even though the logic was updated at different frequencies, the agents still moved continuously due to the nav-mesh.

Initially, the nav-mesh agent would be the only logic the virtual agents performed. However, it is unlikely that a typical application would only have navigation for a virtual agent being the only overhead. Therefore, logic was added to determine if there was a virtual agent on the opposite team within 50 centimeters in front of it. If there were, the agent would then fire a projectile in the same direction it was facing. If the projectile struck the other virtual agent, then the score of the first would increase by one. These projectiles also had a timer on them so that they would be destroyed after one second. The rate of fire was controlled by an additional co-routine that would wait for .3 seconds before allowing the virtual agent to fire again.

Each virtual agent had a small canvas above itself in world space. This canvas displayed the current score for each virtual agent and was used to simulate a visualization load that may be required for an AR application. The control function would rotate the canvas to make the visualizations more user-friendly to ensure it was facing the camera regardless of what direction the virtual agent was moving. Typically, this would be done in the update function, but it was added to the control function to keep all tests consistent.

4.2 Testing Procedures and Data Cleanup

The testing procedure took five individual tests of the frequencies mentioned above. The user would start each test and disable the default profiler, to keep things consistent, then move outside the arena and sit down. The user's action would be constrained to look around the arena as the different virtual elements were spawned and performed their control logic. This reduction in physical movement is essential as fast or erratic movements by the user will cause the system to do extra computation to keep the virtual environment aligned with the real environment. This experiment did not intend to put the AR application under stress from user movement. After the number of agents reached 100, the test was stopped, and the framerate was collected in a comma-delimited file. The raw data was very noisy, as shown in Figure 1.

Five tests were run for each frequency and then averaged together to reduce the noise. The results were still noisy; therefore, a moving average with a sliding window of size ten was used to improve the results further (Equation 2).

$$AverageRecord_r = \frac{\sum_{Test=1}^{5} Value_{r,Test}}{5}$$

$$\forall ma \text{ where } ma = \{10 \ldots Number\ of Records\}.$$

$$MovingAverage_{ma} = \frac{\sum_{n=ma-10}^{ma} AverageRecord_n}{10}$$

Equation 2: Calculation for Moving Average

The improvement can be seen by comparing the above graph with Figure 2. Notice the noise is significantly reduced.

5 Problems Encountered

The most significant problem encountered was getting the program to deploy to the HoloLens 2 correctly. Much time was spent configuring the project and libraries to ensure the program ran successfully on the HoloLens 2. The resolution to this problem was using the older Windows Mixed Reality plugin instead of, the newer recommended OpenXR plugin. More time is needed to determine why the newer plugin did not work correctly. Once the configuration issues had been resolved, no real problems were encountered. Due to the cross-platform capabilities of Unity and MRTK, deploying on a HoloLens 1 and Android were almost seamless.

6 Results

The six frequencies were tested over Android, HoloLens 1, and HoloLens 2. Each platform mapped the average framerate per number of agents onto a graph to compare the best results. From that information, two additional tables were created. One of the tables was the device's highest number of agents at that specific frequency while remaining above 50 fps. The other table provided the last framerate recorded in the tests.

6.1 Android

The Android platform took a different philosophy than the HoloLens 1 or 2. The performance philosophy for Android was consistency. All frequencies hovered around 30 fps regardless of the number of agents. This artificial throttling of the framerate meant that up to 100 agents, the control frequency had almost no measurable impact on the device's performance. The

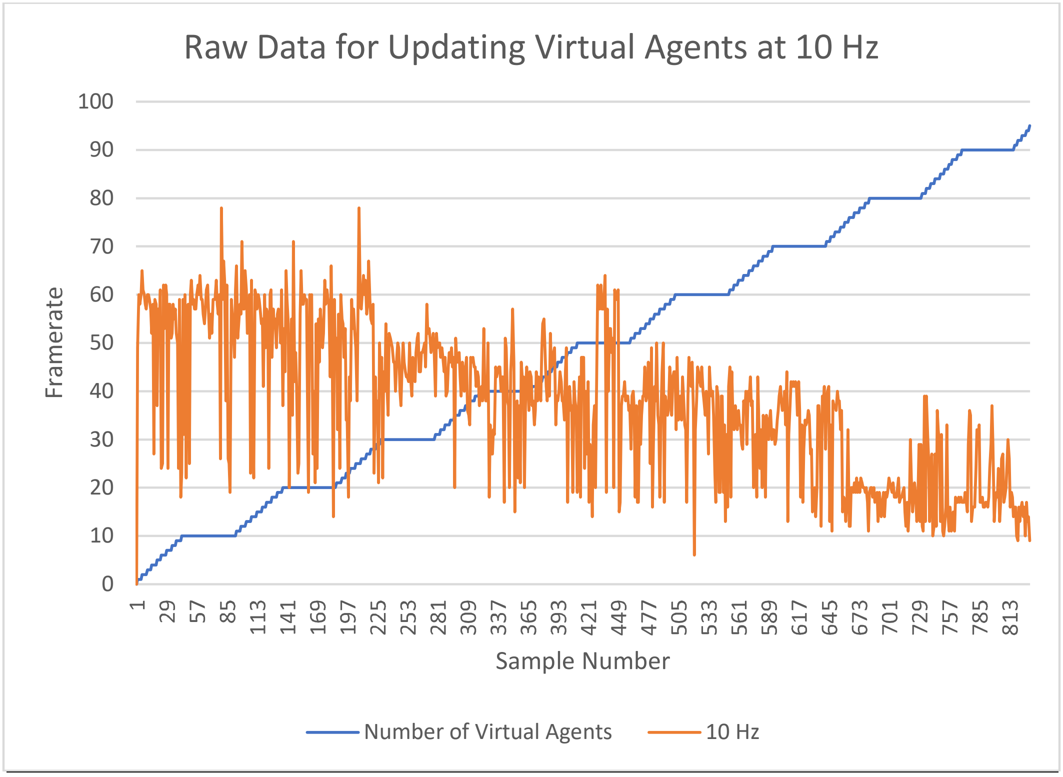

Figure 1: Raw data from 10 Hz virtual agent update experiment

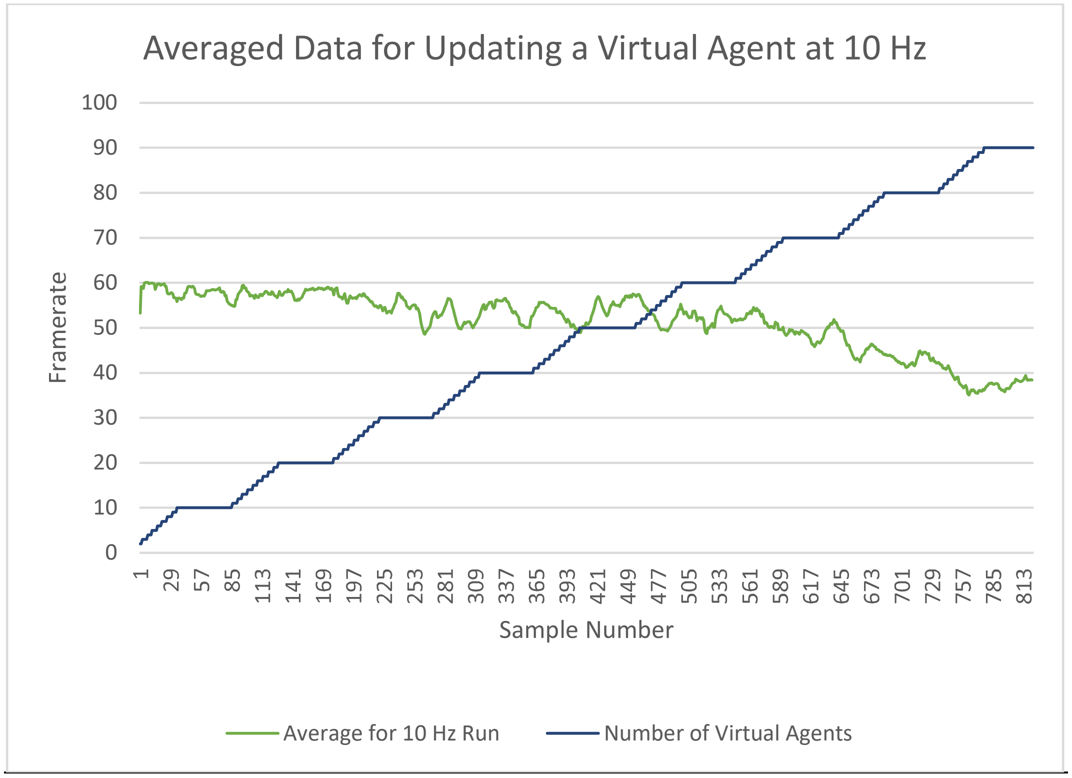

Figure 2: Averaged data for updating a virtual agent at 10 Hz

frame rate is being throttled specifically to 30 fps as the performance was the same for ten agents and 98 agents. This fact reveals two critical design considerations when building for Android. First, the platform can handle a higher number of agents before suffering from frame loss. Secondly, a developer should remember they are never going to achieve higher than 30 fps with the platform (Figure 3).

Since the frame rate never exceeded 30 fps, it was impossible to create the table reflecting the most significant number of virtual agents before falling below 50 fps. It was possible to gather the last recorded frame rate for the android. Since these values reflect a running average over multiple runs, it does indicate which frequency would be able to maintain 30 fps for the longest. Fixed update and 20Hz both had 34 fps. This result was surprising as conventional wisdom with the Unity Game Engine would suggest fixed updates would have the worst performance. Once again, this is probably due to the philosophical design approach to limit everything to 30 fps. Standard frame update and 2 Hz performed the worst (Figure 4).

6.2 HoloLens 1

The results for the HoloLens 1 revealed some noteworthy differences between itself and its subsequent version, the HoloLens 2. The HoloLens 1 held 60 fps until about 40 agents. Even more interesting is that there was very little noise (Figure 5). The HoloLens 1 held 60 fps as tightly as the Android held the 30-fps rate. The HoloLens 2 contained much more noise, and only some of the frequencies held 60 fps for controlling the first 40 virtual agents (Figure 8). However, after 40 virtual agents, the HoloLens 1 showed significant performance failure for all frequencies.

The HoloLens 1 performed the best with 10Hz (Figure 6). This result was unexpected as 2Hz would intuitively cost less CPU. As explained later, these phenomena would also extend to the HoloLens 2. In the following table (Figure 6), the 10 Hz performed the best, reaching 53 agents before falling below 50 fps. Apart from this anomaly, the results were as expected. The higher the frequency, the lower the number of virtual agents. Fixed update performing the worst due to its real-time constraint.

The table containing the final framerate recorded at each frequency is shown below (Figure 7). Once again, 10 Hz outperformed the other frequencies at a smaller margin.

Another note-worthy observation was that Unity's normal update function outperformed 20 and 2 Hz. Further analysis determined that this was due to an unintended feedback loop. As the frames-per-second drop, the number of times an update is called per second drops, thus reducing the total load.

6.2 HoloLens 2

The results for the HoloLens 2 demonstrated an iterative improvement from its predecessor. However, there were some unexpected results. The performance was stronger than the HoloLens 1 but much noisier. This noise in the performance was unexpected as the hardware is significantly more powerful. The only element that could have influenced this was the fact

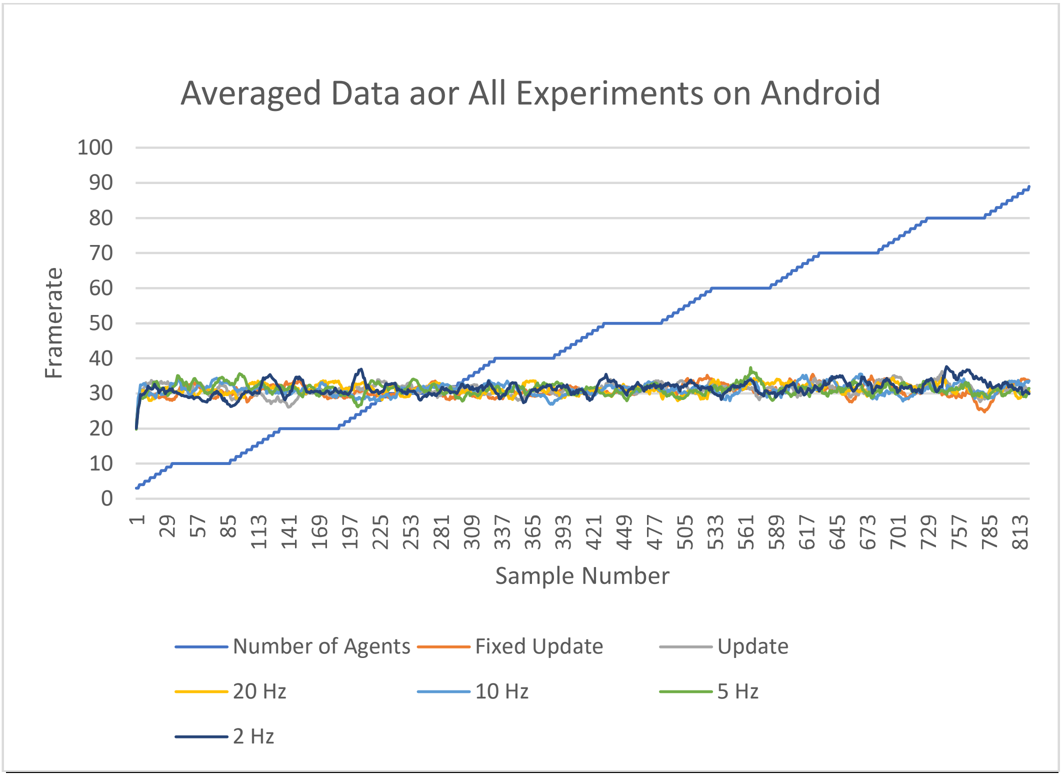

Figure 3: Averaged Data for All Experiments on Android

Control Frequency (From Best to Worst Performance)	Last Framerate for Android
Fixed Update	34
20 Hz	34
10 Hz	33
5 Hz	31
Update	30
2 Hz	30

Figure 4: Last Frame Rate for Android Platform per Control Frequency

that the HoloLens 2 uses an ARM processor, whereas the HoloLens 1 used a standard PC architecture processor (Figure 8).

Comparing the highest number of virtual agents while maintaining 50 FPS, 10 Hz significantly outperformed other frequencies. This result was an interesting trend where 10 Hz outperformed 5 and 2 Hz. The performance spread was much larger than the HoloLens 1. 2 Hz, and the Fixed update could only maintain 50 virtual agents at 50 fps. From this test, it was concluded that 10 Hz was optimal for programming control agents for the HoloLens family (Figure 9).

The last framerate recorded showed all the frequencies in the same order, except for the normal update, which surpassed 20 Hz. Again, this is most likely due to the feedback loop associated specifically with the update function (Figure 10).

7 Analysis

To conclude this research, the optimal frequencies were compared and graphed. The results below reflect both the throttled philosophy of the Android platform and the increased performance of the HoloLens 2. It is worth noting that the HoloLens 1 did maintain 60 frames-per-second longer than the HoloLens 2. However, its performance decay was quicker than the HoloLens 2 (Figure 11).

Since the Android platform did not have a higher framerate than 30 fps, the last virtual count before falling below 50 fps was compared between the HoloLens 1 and HoloLens 2. Here both tests show that 10 Hz performed the best. However, the HoloLens 1 has the smaller frequencies performing better after this, whereas the HoloLens 2, 2 Hz, is one of the lowest performing frequencies. This discrepancy in performance indicates the hardware change between the two devices (Figure 12).

The final analysis compared the final framerate for all frequencies across all platforms. On the HoloLens 1 and 2, the 10 Hz performed the best. The android fixed update and 20 Hz tied for the highest framerate. It is interesting to note that despite the hardware difference, the Android's performance at the end of the test is comparable with both HoloLens 1 and HoloLens 2.

‘

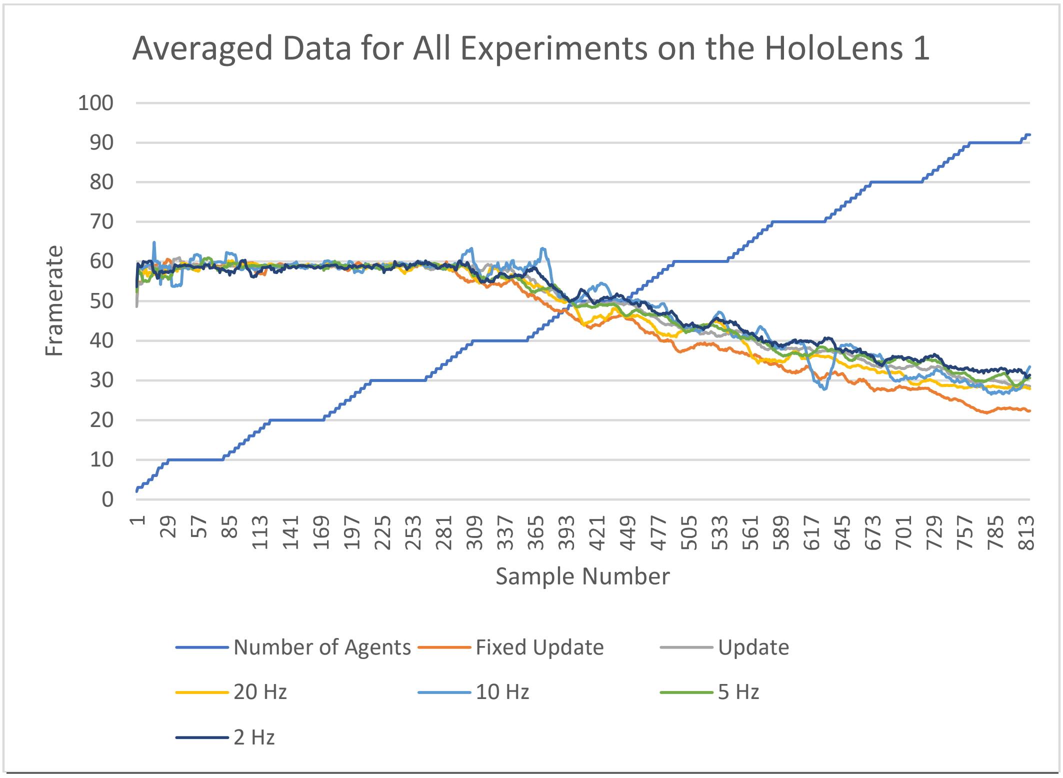

Figure 5: Averaged data for all experiments on the HoloLens 1

Frequency (From Best to Worst Performance)	Highest Number of Virtual Agents before Dropping Below 50 fps. (HoloLens 1)
10 Hz	53
2 Hz	51
5 Hz	49
20 Hz	49
Update	49
Fixed Updated	43

Figure 6: Highest number of virtual agents before falling below 50 fps

Frequency (From Best to Worst Performance)	**Last Framerate for the HoloLens 1**
10 Hz	38
5 Hz	36
Update	34
20 Hz	33
2 Hz	30
Fixed Update	26

Figure 7: Last frame rate for HoloLens 1 per control frequency

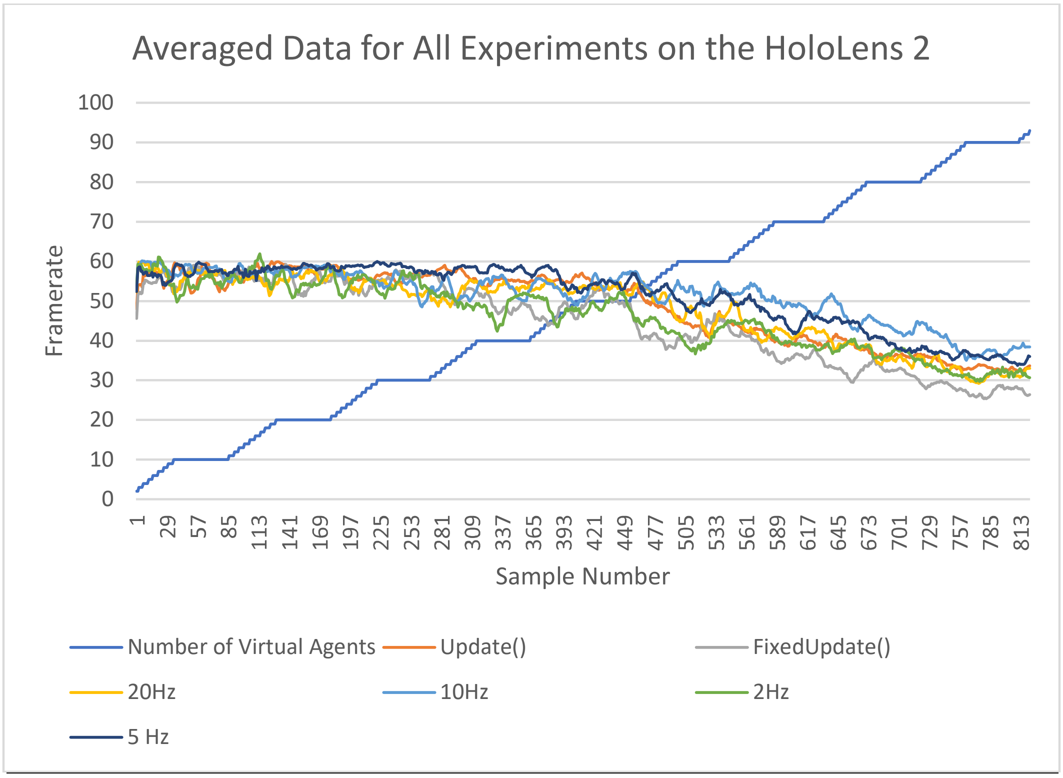

Figure 8: Averaged data for all experiments on the HoloLens 2

Frequency (From Best to Worst Performance)	**Highest Number of Virtual Agents before Dropping Below 50 FPS. (HoloLens 2)**
10 Hz	71
5 Hz	66
20 Hz	58
Update	54
2 Hz	50
Fixed Update	50

Figure 9: Highest number of virtual agents before falling below 50 fps (HoloLens 2)

Frequency (From Best to Worst Performance)	**Last Framerate for the HoloLens 2**
10 Hz	38
5 Hz	36
Update	34
20 Hz	33
2 Hz	30
Fixed Update	26

Figure 10: Last frame rate for HoloLens 1 per control frequency (HoloLens 2)

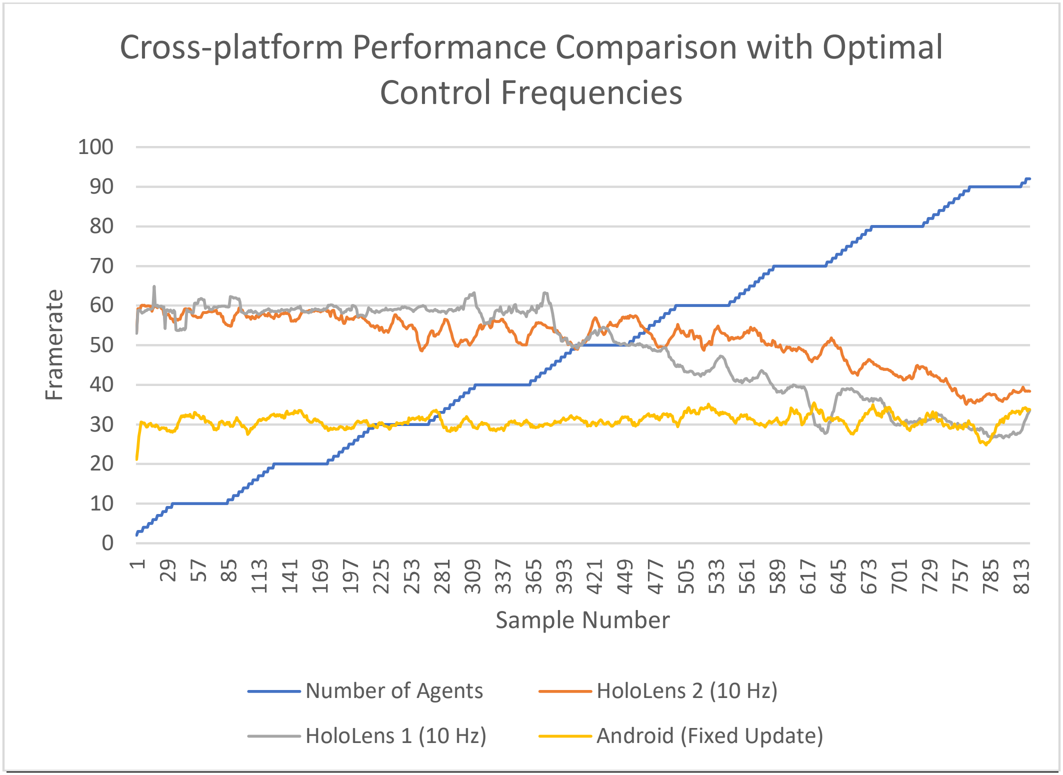

Figure 11: Comparing the optimal frequencies for all three platforms

Comparison of Highest Virtual Agent Count Before Dropping Below 50 fps		
Frequency	**HoloLens 1**	**HoloLens 2**
2 Hz	51	50
5 Hz	49	66
10 Hz	53	71
20 Hz	49	58
Update	49	54
Fixed Update	43	50

Figure 12: Highest number of agents before dropping below 50Hz

Comparison of the Platforms and the Final Framerate per Each Frequency			
Frequency	**HoloLens 2**	**HoloLens 1**	**Android**
2 Hz	30	31	30
5 Hz	36	31	31
10 Hz	38	34	33
20 Hz	33	28	34
Update	34	29	30
Fixed Update	26	22	34

Figure 13: Comparison of the platforms and the final framerate per each frequency

8 Future Work

This research created a foundation for the computational power expected from common AR devices. Several projects will benefit from this research and the knowledge of the optimal control frequencies. One research area is localizing multiple users and virtual objects without object tracking. The AR device must map the environment and combine maps from other devices to create a shared coordinate system. The computational requirement for this is immense. Another project that could benefit from this research is dynamically loading content into an AR environment. This capability would also require computational power and bandwidth to import, load, and visualize new objects that were not natively part of the application.

9 Conclusion

In conclusion, this paper presented computation stair-step tests to determine the optimal control frequency for three platforms: HoloLens 1, HoloLens 2, and an Android Phone. Surprisingly the lowest frequency did not perform the best. 10 Hz performed the best for HoloLens 1 and 2, while 20 Hz or Fixed Update performed the best on the Android. This paper also discovered a philosophical difference between the platforms. The HoloLens family would give the highest framerate possible, while the Android system kept a consistent 30 frames per second regardless of the computational load. These results will be helpful when designing AR applications in the future when considering platform constraints.

References

[1] J. Ahn, J. Williamson, M. Gartrell, R. Han, Q. Lv, and S. Mishra, "Supporting Healthy Grocery Shopping via Mobile Augmented Reality," ACM Trans Multimedia Comput. Commun. Appl., 12:16:1-16:24, 2015, doi: 10.1145/2808207.

[2] B. Garrett, J. Anthony, and C. Jackson, "Using Mobile Augmented Reality to Enhance Health Professional Practice Education," Current Issues in Emerging eLearning 4, 2018.

[3] A. Hartholt, S. Mozgai, E. Fast, M. Liewer, A. Reilly, W. Whitcup, and A. S. Rizzo, "Virtual Humans in Augmented Reality: A First Step Towards Real-World Embedded Virtual Roleplayers," Proceedings of the 7th International Conference on Human-Agent Interaction, pp. 205-207, 2019.

[4] K. Kim, L. Boelling, S. Haesler, J. Bailenson, G. Bruder, and G. F. Welch, "Does a Digital Assistant Need a Body? The Influence of Visual Embodiment and Social Behavior on the Perception of Intelligent Virtual Agents in AR," 2018 IEEE International Symposium on Mixed and Augmented Reality (ISMAR), IEEE, pp. 105-114, 2018.

[5] C. Moro, Z. Štromberga, A. Raikos, and A. Stirling, "The Effectiveness of Virtual and Augmented Reality in Health Sciences and Medical Anatomy," *Anatomical Sciences Education* 10:549-559, 2017, doi: 10.1002/ase.1696.

[6] M. Obaid, I. Damian, F. Kistler, B. Endrass, J. Wagner, and E. André, "Cultural Behaviors of Virtual Agents in an Augmented Reality Environment," International Conference on Intelligent Virtual Agents, Springer, pp 412–418, 2012.

[7] M. Obaid, R. Niewiadomski, and C. Pelachaud, "Perception of Spatial Relations and of Coexistence with Virtual Agents," International Workshop on Intelligent Virtual Agents. Springer, pp 363-369, 2011.

[8] P. Parashar, L. M. Sanneman, J. A. Shah, and H. I. Christensen, "A Taxonomy for Characterizing Modes of Interactions in Goal-driven, Human-Robot Teams," 2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), pp. 2213-2220, 2019.

[9] S. Saeedi, B. Bodin, H. Wagstaff, A. Nisbet, L. Nardi, J. Mawer, N. Melot, O. Palomar, E. Vespa, T. Spink, C. Gorgovan, A. Webb, J. Clarkson, E. Tomusk, T. Debrunner, K. Kaszyk, P. Gonzalez-De-Aledo, A. Rodchenko, G. Riley, C. Kotselidis, B. Franke, M. F. P. O'Boyle, A. J. Davison, P. H. J. Kelly, M. Luján, and S. Furber, "Navigating the Landscape for Real-Time Localization and Mapping for Robotics and Virtual and Augmented Reality," Proceedings of the IEEE 106:2020-2039, 2018, doi: 10.1109/JPROC.2018.2856739.

[10] I. C. S. da Silva, G. Klein, and D. M. Brandão, "Segmented and Detailed Visualization of Anatomical Structures based on Augmented Reality for Health Education and Knowledge Discovery," Adv. Sci. Technol. Eng. Syst J, 2:469-478, 2017, doi: 10.25046/aj020360.

[11] I. Wang, J. Smith, and J. Ruiz, "Exploring Virtual Agents for Augmented Reality," Proceedings of the 2019 CHI Conference on Human Factors in Computing Systems, ACM, Glasgow Scotland Uk, pp. 1-12, 2019.

[12] Augmented Reality - Wikipedia, https://en.wikipedia.org/wiki/Augmented_reality, Accessed 19 Nov 2022.

[13] Unity - Scripting API: Time.unscaledDeltaTime, https://docs.unity3d.com/ScriptReference/Time-unscaledDeltaTime.html, Accessed 5 Apr 2022.

Bradford A. Towle, Jr. is an Assistant Professor at Florida Polytechnic University. He has designed and coordinated the Game Design and Development concentration within the Computer Science Department since 2016. His primary research topics include augmented reality applications, autonomous robotic control architectures, and human-computer interaction. Dr. Towle researches individually and with undergraduates, hoping to foster future generations of researchers within Computer Science. He has successfully advised three graduate students helping them achieve a Master's degree in Computer Science. He is actively working to build an international reputation for his augmented reality research and has formed a student research group at Florida Polytechnic.