Adaptive AR/MR Output Security on a Physical Device using Reinforcement Learning

Joseph DeChicchis
Research Independent Study Report
Duke University, April 29, 2019

I. Introduction

Augmented Reality (AR) and Mixed Reality (MR) are becoming increasingly ubiquitous. Research has shown that AR/MR will be a $100 industry by 2020 [7] and companies are chasing this market by making their own solution to AR/MR.

Some solutions, such as Apple's ARKit [18] and Google's ARCore [19], use smartphones to create an AR/MR experience through the smartphone camera and display. However, these experiences are not fully immersive since users must hold up their smartphone in front of them. Microsoft's HoloLense [17] and Magic Leap's Magic Leap One [16] Head Mounted Display (HMD) AR/MR devices enable a much more immersive AR/MR experience since users do not have to hold a device in front of them and objects (i.e. "holograms") are project right in front of them to augment their field of view. Although these devices are limited in their current field of view, their immersiveness is promising for the future of AR/MR.

Although current immersive AR/MR technologies are focused on the use of proprietary devices, HoloKit [20] enables immersive AR/MR experiences with the use of a smartphone and simple cardboard mounting device (similar to Google Cardboard [21] which is for Virtual Reality experiences). In addition, researchers have suggested that AR car windshields may be a good use case for AR/MR technologies [23] through applications such as left-turn driving aids [24].

While the increasing proliferation of AR/MR devices will undoubtedly enable many new applications, issues of privacy and security cannot be ignored. Much of the previous work on AR/MR privacy has focused on the inputs to AR/MR devices (i.e. input security) [14,15]. Noting this gap in AR/MR security research, Lebek et al. suggested in a position paper that there should also be a focus on output security to secure the output of AR/MR devices [4].

Visual AR/MR output security is concerned with two issues pertaining to the user's visual field:

- Regulating visual content displayed to the user to reduce distraction and obstruction of the real-world context.
- Preventing holograms from obscuring other holograms with a higher priority.

To understand why these two issues are a concern for AR/MR security, take the case of an AR windshield in a car. Suppose the AR display in the car had a hologram to display the current speed and a hologram with the name of the song which is currently playing. It would be dangerous if either of the holograms obstructed an important real-world object such as a stop sign (the first security concern). In addition, it would also be dangerous if the name of the song hologram obstructed the current speed hologram (the second security concern).

While one could leave these output security concerns to the application developer, it is much safer to have the OS guarantee the security of AR/MR device outputs. In this vein, previous work has investigated what an OS level AR/MR device output security module may look like by allowing developers to write policies [3]. However, as noted by Surin et al., these hand-coded policies are cumbersome and impractical for real-world use [1]. For example, specifying a policy to move holograms that are obstructing an important real-world object while not moving too far from its original location and not obscuring other important holograms at the same time is a very difficult task. Surin et al. proposed the use of reinforcement learning (RL) to solve this problem to automatically generate policies and demonstrated its effectiveness [1].

Although previous work has demonstrated the importance of output security and demonstrated its feasibility in simulation [1,3], they have not deployed output security policies on a physical AR/MR device to ascertain its viability in the real-world. In particular, whether a RL based output security model can be deployed without degrading performance was left an open question [1]. To fill this gap, this work investigates whether RL models can be deployed on a physical AR/MR device without a noticeable degradation in performance and test the deployment of an output security policy trained using RL.

It is important to note that while this work focuses on visual output security, there are concerns that other AR/MR output, such as audio and haptic output, may need to be regulated as well [1,3]. In addition, one may also want to limit distracting and uncomfortable AR/MR output such as blinking holograms much like web browsers have evolved to block popups and the blink tag [3].

II. RL on a Physical Device

The Magic Leap One [16] was used to ascertain whether a RL model can be deployed on a physical device without degrading the user experience. Since responsiveness is essential to a good AR/MR experience, computation which may decrease the frame rate of the AR/MR device, such as running a RL model, may be detrimental to the user experience.

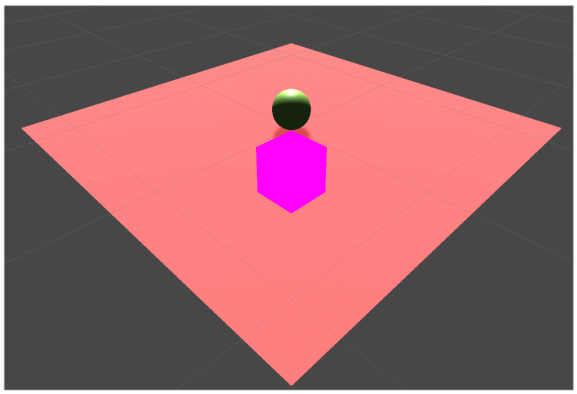

Fig. 1: A simple game which a RL agent was trained to play.

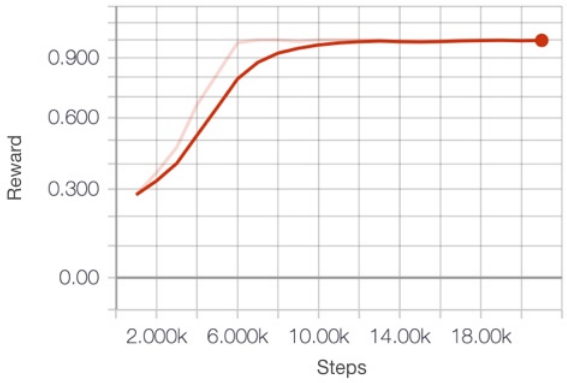

Fig. 2: Training of the agent to play the simple game.

A simple game was developed using Unity [6] which a RL agent was trained to play. The objective of the game is to move the sphere to the target (cube) without falling off the platform (see figure 1). A RL agent was trained using the ml-agents framework [5] to play the game.

Reward Function The following reward function was used to train the agent:

- $+1$ reward if the sphere came in contact with the target.
- 0 reward if the sphere fell off the platform.

Although a reward function where the agent was punished for falling off the platform (-1 reward) was tested, this made the agent more "timid" which made it less likely to explore its environment and learn the optimal policy to find the target.

Training Training converged quickly (around 10 minutes) on a 2017 15in MacBook Pro with a Radeon Pro 560 graphics card (see figure 2). The hyperparameters used to train the agent are summarized in table I. The PPO algorithm [8] was used to train the agent.

Deployment Although there were some challenges with ml-agent and Magic Leap One SDK configurations to deploy the RL model on the Magic Leap One to play the simple game, once deployed the RL agent successfully played the game without any noticeable performance degradation.

This experiment, training a RL agent to play a simple game and deploying it on the Magic Leap One, illustrated that a RL model could be deployed on a physical device without a noticeable performance impact, laying the foundation to train and deploy a RL output security model on a physical device.

Parameter	Value
Num Layers	2
Hidden Units	128
Batch Size	10
Beta	5.0×10^{-3}
Buffer Size	100
Epsilon	0.2
Gamma	0.99
Lambda	0.95
Learning Rate	3.0×10^{-4}
Num Epochs	3
Time Horizon	64
Normalize	False
Use Recurrent	False
Use Curiosity	False

TABLE I: Hyperparameters used to train a RL agent to play the simple game.

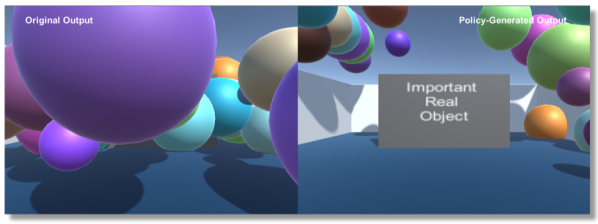

Fig. 3: Example of visual outputs before and after RL-generated policies are applied (from Surin et al. [1]).

III. RL Output Security Model

The output security problem for this project was defined as having one important real-world object and one or more holograms. The position of the important real-world object and holograms were randomly assigned and the important real-world object was always placed behind holograms, although the holograms did not necessarily obscure the important real-world object. The width and height of the holograms were also randomly assigned in some cases.

Given these conditions, the goal is to train an agent to move the holograms so that they do not obscure the important real-world object while keeping the holograms as to their original positions as possible. Note that not obscuring other holograms was not taken into account. Fig 3 is an example of a simulated visual output for before and after RL-generated policies are applied [1].

State Space A model where the agent has complete knowledge of the states was used in this project. Two state spaces were used:

- S_1 in which each $s \in S_1$ consisted of the location (x and y coordinate), width, and height of the holograms as well as the important real-world object. That is, S_1 has $4(N+1)$ observations at each time step where N is the number of holograms.

- S_2 in which each $s \in S_2$ consisted of the location (x and y coordinate) of the important real-world object and the location (x, y coordinate), x-velocity, and y-velocity of the holograms. That is, S_2 has $4N + 2$ observations at each time step where N is the number of holograms.

Action Space The RL agent was tasked with moving hologram(s) away from real-world objects. Two action spaces were used:

- A_1 where the agent outputs the x and y coordinate for each hologram. That is, A_1 has 2 actions at each time step where N is the number of holograms.
- A_2 where the agent outputs the x and y force for each hologram. That is, A_1 has 2 actions at each time step where N is the number of holograms.

Reward Function Initially, the following reward function (adapted from Surin et al. [1]) with various values of α and β, where r is the position of the important real-world object, O is the set of original hologram positions, and O' is the set of new hologram positions was chosen, was used:

$$R_1 = \alpha \sum_{o' \in O'} (Dist_{x,y}(o', r)) - \beta \sum_{o \in O} (Dist_{x,y}(o, r)).$$

This reward function is meant to move holograms away from the important real-world object while keeping holograms as close to their original position as possible. Although the suggested values of $\alpha = 2.0$ and $\beta = 1.5$ [1] were tried, the reinforcement learning model did not converge in some cases, most likely because of hyperparameter issues. The following four states were defined to make a simpler reward function which worked well:

- Success: The hologram does not obscure the important real-world object and has not moved outside of the user's field of view.
- Failed: The hologram has moved outside of the user's field of view.
- Incomplete: The hologram obscures the important real-world object and has not moved outside of the user's field of view.
- Done: The training session ended without reaching a Success or Failed state.

Two reward functions were defined using these three states.

Reward Function R_2

$$R_2 = \begin{cases} 1, & \text{State = Success} \\ 0, & \text{State = Failed} \\ \text{continue session}, & \text{State = Incomplete} \\ 0, & \text{State = Done} \end{cases}$$

R_2 rewards the agent for moving a hologram away from a real-world object while ensuring its visible to the user. In the case that there is more than one hologram, the reward was accumulated for each hologram and divided by the number of holograms to normalize the reward.

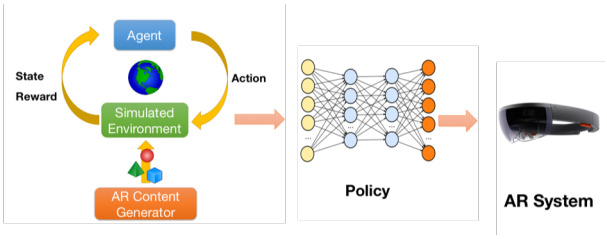

Fig. 4: General overview of the model training and deployment process (from Surin et al. [1]).

Reward Function R_3

$$R_3 = \begin{cases} \frac{M-S}{M}, & \text{State = Success} \\ 0, & \text{State = Failed} \\ \text{continue session}, & \text{State = Incomplete} \\ 0, & \text{State = Done} \end{cases}$$

where M is the max steps and S is the current step. R_3 rewards the agent for moving a hologram away from a real-world object while ensuring its visible to the user in a timely manner by giving less reward the more steps it takes. In the case that there is more then one hologram, the reward was accumulated for each hologram and divided by the number of holograms to normalize the reward. Furthermore, the Success state, in this case requires, that the hologram is in the same relative location to the important real-world object it was originally in. For example, if the hologram was originally at the top left of the important real-world object, it would have to move away from the important real-world object while remaining in the top left region of the important real-world object.

IV. RL Output Security Model Evaluation

Fig 4 provides an overview of the output security model training and deployment process [1]. The model is trained using reinforcement learning in a simulated environment and deployed to the physical device (Magic Leap One) once the training converges.

In the simulated environment, a green cube, which had a consistent scale, represented the important real-world object. It was placed randomly within the user's field of view at the beginning of each training cycle. Spherical holograms represented the objects which needed to be moved away from the important real-world object.

Two types of RL output security models were trained. The first policy calculated where to place the holograms while the second calculated the direction in which to move the holograms.

Position Policy The policy to calculate the position to move the holograms was trained using reward function R_1, with various values for α and β, along with state space S_1 and action space A_1. The hyperparameters in table II were used along with the PPO algorithm.

Direction Policy The policy to calculate the direction to move the holograms was trained using reward function R_2

Parameter	Value
Num Layers	3
Hidden Units	128
Batch Size	64
Beta	5.0×10^{-3}
Buffer Size	2048
Epsilon	0.2
Gamma	0.99
Lambda	0.95
Learning Rate	3.0×10^{-4}
Num Epochs	5
Time Horizon	2048
Normalize	False
Use Recurrent	False
Use Curiosity	False

TABLE II: Hyperparameters used to train a RL agent to move holograms to a specific position away from an important real-world object.

Parameter	Value
Num Layers	2
Hidden Units	64
Batch Size	10
Beta	1.0×10^{-2}
Buffer Size	512
Epsilon	0.2
Gamma	0.99
Lambda	0.95
Learning Rate	1.0×10^{-3}
Num Epochs	5
Time Horizon	64
Normalize	False
Use Recurrent	False
Use Curiosity	False
Max steps per training cycle	100 to 500

TABLE III: Hyperparameters used to train a RL agent to move holograms away from an important real-world object.

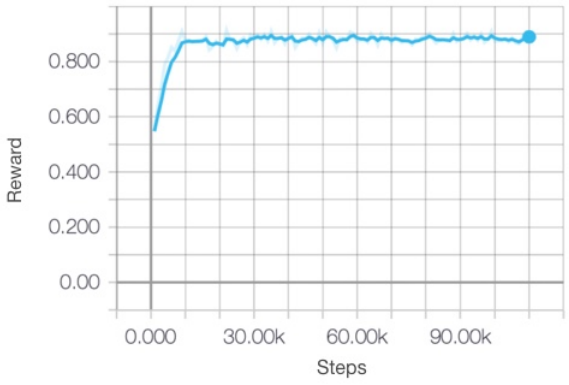

Fig. 5: Training for the position policy with one hologram with no penalty for moving the hologram away from its original position.

with state space S_2 and action space A_2. The hyperparameters in table III were used along with the PPO algorithm.

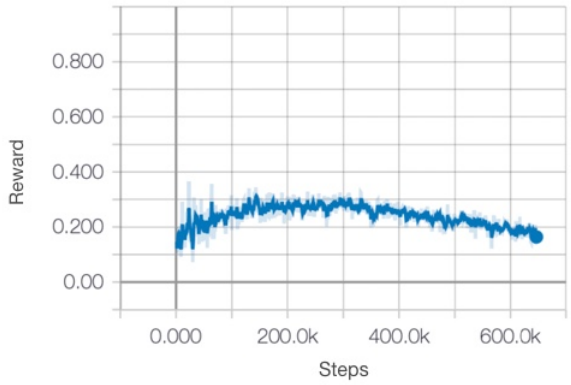

Fig. 6: Training for the position policy with one hologram with a penalty for moving the hologram away from its original position.

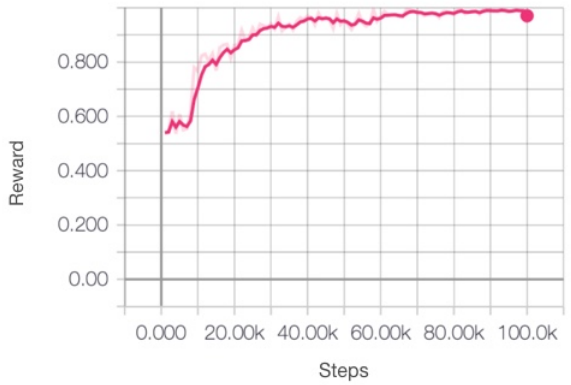

Fig. 7: Training for the direction policy with one hologram.

V. Results

Position Policy As illustrated in figure 5, training a RL model for the position policy with one hologram converged when not penalizing the agent for moving the hologram away from its original location (i.e. setting β to 0 in R_1). However, the agent simply learned to move the hologram to the top right of the user's field of view in this case.

To overcome this problem, a RL agent for the position policy was also trained by penalizing the agent for moving the hologram away from its original location (i.e. setting β to some positive number in R_1). Although various values of α and β were tried, the model did not converge. An example training for is shown in figure 6.

Direction Policy As illustrated in figure 7 and figure 8, training a RL model for the direction policy with one hologram and three holograms both converged. However, the reward for the three hologram case was substantially lower. Figure 9 is an example of the direction policy being applied by an RL agent to three holograms. In addition, the holograms tended to simply move in a general direction (i.e. right) which led to the holograms moving away from its original location more

Fig. 8: Training for the direction policy with three holograms.

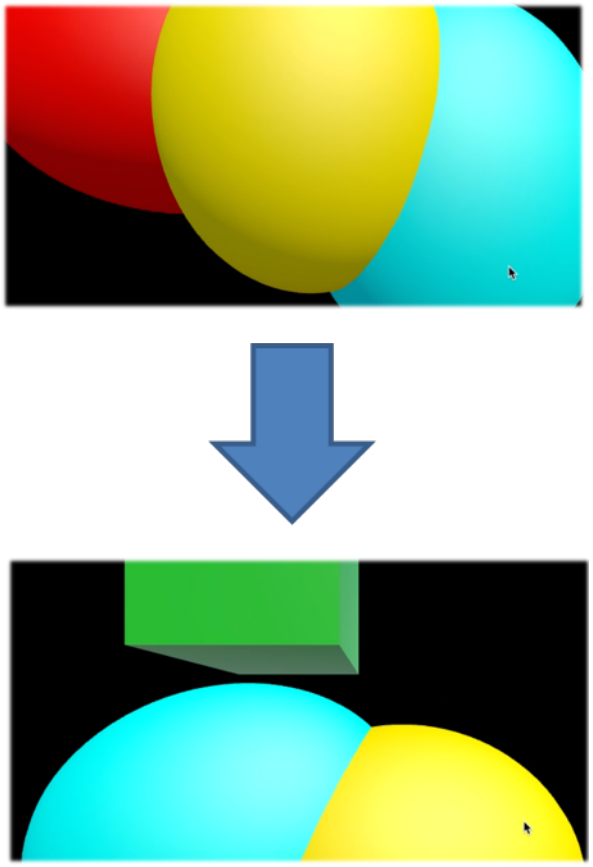

Fig. 9: Example of visual output before and after model generated direction policy is applied to three holograms. The green cube represents an important real-world object.

than they had to for them to no longer obscure the important real-world object.

VI. RL Output Security Application

A full output security application was developed to test the deployment of a RL trained output security policy. The direction policy was chosen for this application since it performed better than the position policy during initial testing. However,

Parameter	Value
Num Layers	2
Hidden Units	128
Batch Size	10
Beta	5.0×10^{-3}
Buffer Size	100
Epsilon	0.2
Gamma	0.99
Lambda	0.95
Learning Rate	3.0×10^{-4}
Num Epochs	3
Time Horizon	64
Normalize	False
Use Recurrent	False
Use Curiosity	False
Max steps per training cycle	100 to 500

TABLE IV: Hyperparameters used to train a RL agent to move holograms away from an important real-world object.

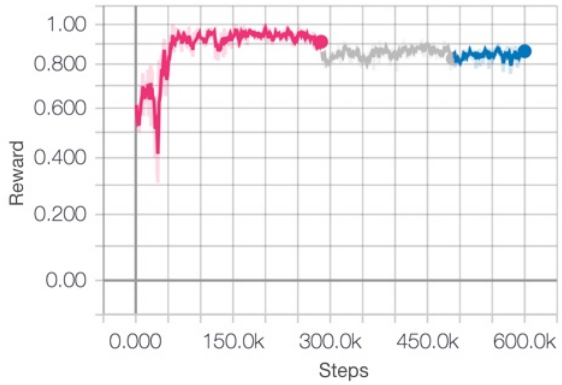

Fig. 10: Training for the direction policy with one hologram while incrementally increasing the distance the hologram should be from the important real-world object to be in the Success state. Note that the varying line colors is a result of pausing then resuming training.

instead of training a model to move n number of holograms away from an important real-world object, one model was trained using the R_3 reward function, S_2 state space and A_2 action space for a single hologram, enabling this single model to be used for an arbitrary number of holograms.

For the purposes of this application, the important real-world object and hologram had a constant size and random location.

Training The RL agent was trained using the hyperparameters in table IV. In addition, the distance from the real-world object needed for the hologram to be considered no longer obstructing it was incrementally increased to help the model converge. Figure 10 shows the training of the direction policy RL agent.

Image Tracking The image tracking library built into the Magic Leap One was used to recognize and track an important real-world object. The image in figure 11 was printed on a letter size piece of paper and used as the important real-world object. Although a simpler image was originally used, the

Fig. 11: The target which was used as the important real-world object for the output security application (from the Magic Leap documentation [16]).

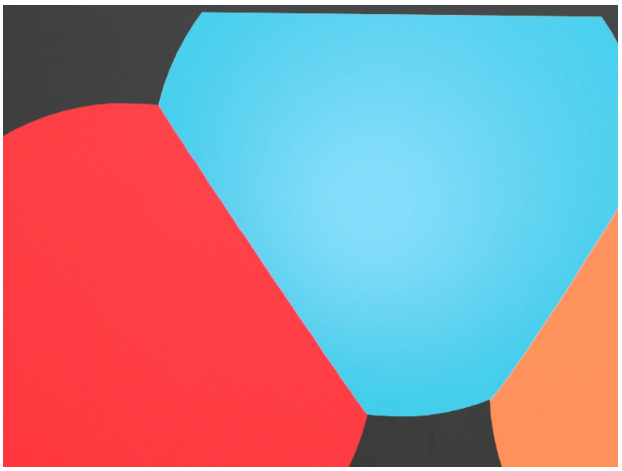

Fig. 12: Example of what is displayed on the Magic Leap One display before model generated direction policy is applied to three holograms. Note that the user's field of view is actually larger but is clipped by the Magic Leap One screenshot functionality.

Magic Leap One library performed better using this more complex image. The user interface for this output security demonstration overlaid a green square on top of the important real-world object once it was recognized and tracked. Note that in an actual deployment the important real-world object would be objects in the user's field of view which should not be blocked such as stop signs and pedestrians.

Heuristics Some heuristics were used in addition to the RL trained direction output security policy:

- Only apply the RL trained direction policy if the hologram's original position obstructs the important real-world object.
- Once the RL trained direction policy is applied, move the hologram back to its original position as soon as its original position no longer obstructs the important real-world object.

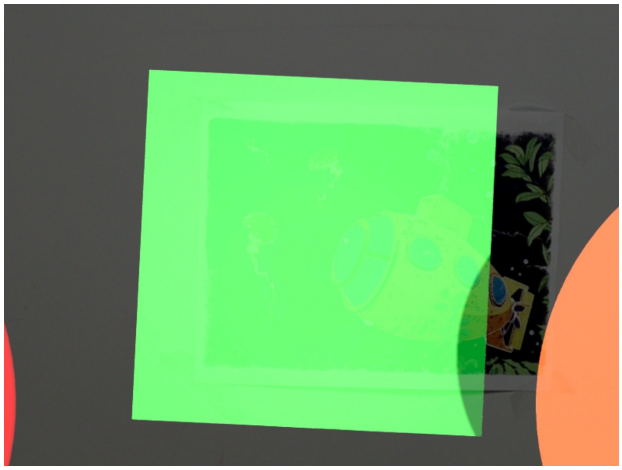

Fig. 13: Example of what is displayed on the Magic Leap One display after model generated direction policy is applied to three holograms. The green square is overlaid in front of the important real-world object, indicating its position. Note that the user's field of view is actually larger but is clipped by the Magic Leap One screenshot functionality.

Deployment The output security application was deployed on a Magic Leap One and without any noticeable performance degradation. Figure 12 shows an example of the user's field of view before the RL trained direction policy is applied and figure 13 shows an example of after the policy is applied. As these images demonstrate, the application successfully identified an important real-world object and moved obstructing holograms in real time.

VII. Discussion

As the training of the position and direction policy using the R_1 and R_2 reward function indicate, the reward function chosen has a direct effect on agent behavior. In the case of the position policy using R_1 with no penalty for moving the hologram away from its initial location, the model simply learned to always move the object to the top right corner which defeats the purpose of moving the obstructing hologram while keeping it as visible to the user as possible. Similarly, in the case of the direction policy using R_2 the agent learned to move the holograms in a general direction (i.e. right) which again moved the holograms away from its initial location. Furthermore, model convergence was an issue when training the position policy with the R_1 reward function with a penalty, most likely due to a lack of hyperparameter tuning.

While the convergence problem of the position policy may have been solved with more hyperparameter tuning, the approach was modifying the R_2 reward function to take in the hologram's initial relative location to the important real-world object was used to come to a more scalable solution by using the R_3 reward function with one hologram. With this training model, used to train the RL agent for the output security application deployment, a RL agent is trained which can be used by an arbitrary number of holograms, cutting back

on the need to train a separate model for various numbers of holograms. In addition, slowly increasing the distance the hologram has to move from the important real-world object to no longer obstruct it aided in model convergence.

Although the built-in image tracking for the Magic Leap One was limited (the user had to be relatively close to the image and not too off center), the output security application performance without any performance degradation for three holograms, indicating that an RL trained agent is a viable solution for visual output security.

VIII. Related Work

A. AR/MR Output Security

Previously, Lebeck et al. proposed the need to focus on AR/MR output security in addition to input security [4] and developed Arya, a simulated proof of concept for an OS-level visual output security module [3]. Surin et al. expanded the policy generation for visual output security using reinforcement learning [1] and more recently using imitation learning [2]. This work expands on the work of using RL to train a visual output security policy to develop a proof of concept output security application on the Magic Leap One.

B. Deep Reinforcement Learning

Reinforcement learning is a trial-and-error model training technique based on behaviorist psychology [10]. Deep reinforcement learning (DRL) is a method for reinforcement learning which uses neural networks to train agent policies and faces three main challenges[9]:

- The only signal the agent receives during training is the reward.
- An agent's observation can contain strong temporal correlations.
- Agents have to be able to overcome long-range time dependencies.

Deep reinforcement learning has been successfully used to solve complex problems such as playing classic Atari 2600 games [12], 3D bipedal and quadrupedal locomotion [11], and playing Go [13]. While various deep reinforcement learning algorithms have been proposed [9], the proximal policy optimization algorithm (PPO), which was used to train the agents in this project, has been found to be easier to implement, more general, and have better sample complexity [8].

IX. Future Work

Some future directions for AR/MR output security include:

- Investigation into what constitutes an important real-world object. This is something which may depend on the context in which the AR/MR device is being used (e.g. driving or walking).
- Better image recognition and tracking for visual AR/MR output security.
- Develop more robust models for visual output security using techniques such as curriculum learning and training agents to cooperate with each other, especially to add parameters so that holograms do not obstruct each other as well as stay as close to their original position as possible.
- Develop a proof of concept for an OS level output security framework similar ro Arya [3] which uses a RL trained agent.

In addition, future work should focus on non-visual output security concerns as well such as audio and haptic output.

Furthermore, as this project has shown because running RL models does not noticeably degrade Magic Leap One performance, additional applications which may benefit from RL or other machine learning models should be investigated. In doing so, the limits of running trained models should be investigated and computation offloaded to edge or cloud servers if necessary.

X. Conclusion

As AR/MR devices becoming prevalent there is a need to secure both the input and output of AR/MR devices. While the sensor input applications have access to present security and privacy concerns, the output of AR/MR applications also poses a security threat. For example, visual output could be distracting and obstruct important real-world objects and audio output could disorient users.

This work developed a proof of concept AR/MR visual output security application using a RL trained output security policy so show that RL trained models are a viable method for developing policies for output security. Although various training parameters where used, a direction based policy trained for individual holograms appears to be the most scalable and easiest to train RL agent for visual output security applications. RL trained models were successfully deployed on a physical AR/MR device (Magic Leap One) without any noticeable performance degradation, showing that machine learning models can be used not only for output security applications but for other real-time AR/MR applications as well.

References

[1] S. Ahn, M. Gorlatova, P. Naghizadeh, M. Chiang, and P. Mittal, "Adaptive Fog-Based Output Security for Augmented Reality," *Proceedings of the 2018 Morning Workshop on Virtual Reality and Augmented Reality Network*, 2018.

[2] S. Ahn, M. Gorlatova, P. Naghizadeh, and M. Chiang, "Personalized Augmented Reality Via Fog-based Imitation Learning," *Proceedings of Workshop on Fog Computing and the IoT*, 2019.

[3] K. Lebeck, K. Ruth, T. Kohno, and F. Roesner, "Securing Augmented Reality Output," *2017 IEEE Symposium on Security and Privacy (SP)*, 2017.

[4] K. Lebeck, T. Kohno, and F. Roesner, "How to Safely Augment Reality," *Proceedings of the 17th International Workshop on Mobile Computing Systems and Applications - HotMobile 16*, 2016.

[5] *Unity-Technologies*, "Unity-Technologies/ml-agents," GitHub. Available: https://github.com/Unity-Technologies/ml-agents.

[6] *Unity*. Available: https://unity.com.

[7] "ABI Research Shows Augmented Reality on the Rise with Total Market Worth to Reach $100 Billion by 2020," *ABI Research*. Available: https://www.abiresearch.com/press/abi-research-shows-augmented-reality-rise-total-ma/.

[8] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal Policy Optimization Algorithms," *arXiv.org*, 28-Aug-2017. Available: https://arxiv.org/abs/1707.06347.

[9] K. Arulkumaran, M. P. Deisenroth, M. Brundage, and A. A. Bharath, "A Brief Survey of Deep Reinforcement Learning," *arXiv.org*, 28-Sep-2017. Available: https://arxiv.org/abs/1708.05866.
[10] R. S. Sutton and A. G. Barto, *Reinforcement learning: An Introduction*. Cambridge (Mass.): The MIT Press., 2018.
[11] J. Schulman, P. Moritz, S. Levine, M. I. Jordan, and P. Abbeel, "High-Dimensional Continuous Control Using Generalized Advantage Estimation," arXiv.org, 20-Oct-2018. Available: https://arxiv.org/abs/1506.02438.
[12] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, S. Petersen, C. Beattie, A. Sadik, I. Antonoglou, H. King, D. Kumaran, D. Wierstra, S. Legg, and D. Hassabis, "Human-level control through deep reinforcement learning," *Nature News*, 25-Feb-2015. Available: https://www.nature.com/articles/nature14236?error=cookies_not_supported&code=7d5e93c7-d8a2-4a86-bccc-d72fdec892e3.
[13] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. van den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, S. Dieleman, D. Grewe, J. Nham, N. Kalchbrenner, I. Sutskever, T. Lillicrap, M. Leach, K. Kavukcuoglu, T. Graepel, and D. Hassabis, "Mastering the game of Go with deep neural networks and tree search," *Nature News*, 27-Jan-2016. Available: https://www.nature.com/articles/nature16961?error=cookies_not_supported&code=5cde24af-4148-4051-90a3-b1ac1fdc51af.
[14] S. Jana, D. Molnar, A. Moshchuk, A. Dunn, B. Livshits, H. J. Wang, and E. Ofek, "Enabling Fine-Grained Permissions for Augmented Reality Applications with Recognizers," *USENIX Security*, 2013.
[15] F. Roesner, T. Kohno, and D. Molnar, "Security and privacy for augmented reality systems," *Communications of the ACM*, vol. 57, no. 4, pp. 8896, 2014.
[16] *Magic Leap*. Available: https://www.magicleap.com.
[17] "HoloLens," *Microsoft*. Available: https://www.microsoft.com/en-us/hololens.
[18] ARKit,Apple Developer. Available: https://developer.apple.com/arkit/.
[19] "ARCore," *Google Developer*. Available: https://developers.google.com/ar/.
[20] *HoloKit*. Available: https://holokit.io/.
[21] "Google Cardboard," *Google*. Available: https://vr.google.com/cardboard/.
[22] *WayRay*. Available: https://wayray.com/.
[23] R. Haeuslschmid, B. Pfleging, and F. Alt, "A Design Space to Support the Development of Windshield Applications for the Car," *Proceedings of the 2016 CHI Conference on Human Factors in Computing Systems - CHI 16*, 2016.
[24] C. Tran, K. Bark, and V. Ng-Thow-Hing, "A left-turn driving aid using projected oncoming vehicle paths with augmented reality," *Proceedings of the 5th International Conference on Automotive User Interfaces and Interactive Vehicular Applications - AutomotiveUI 13*, 2013.