TRANSPORTATION POOLED FUND PROGRAM
QUARTERLY PROGRESS REPORT

Lead Agency (FHWA or State DOT): Kansas DOT

INSTRUCTIONS:
Lead Agency contacts should complete a quarterly progress report for each calendar quarter during which the projects are active. Please provide a project schedule status of the research activities tied to each task that is defined in the proposal; a percentage completion of each task; a concise discussion (2 or 3 sentences) of the current status, including accomplishments and problems encountered, if any. List all tasks, even if no work was done during this period.

Transportation Pooled Fund Program Project # *(i.e, SPR-2(XXX), SPR-3(XXX) or TPF-5(XXX)* **TPF-5(535)**	**Transportation Pooled Fund Program - Report Period:** ☐Quarter 1 (January 1 – March 31) ☐Quarter 2 (April 1 – June 30) ☐Quarter 3 (July 1 – September 30) ☒Quarter 4 (October 1 – December 31)

TPF Study Number and Title:
TPF-5(535): Human-centered Steel Bridge Inspection enabled by Augmented Reality and Artificial Intelligence

Lead Agency Contact: **David Behzadpour**	**Lead Agency Phone Number:** **785-291-3847**	**Lead Agency E-Mail** David.Behzadpour@ks.gov
Lead Agency Project ID: Click or tap here to enter text.	**Other Project ID (i.e., contract #):** Click or tap here to enter text.	**Project Start Date:** 10/1/2024
Original Project Start Date: **10/1/2024**	**Original Project End Date:** **9/30/2027**	**If Extension has been requested, updated project End Date:** Click or tap to enter a date.

Project schedule status:

☒On schedule	☐On revised schedule	☐Ahead of schedule	☐ Behind schedule

Overall Project Statistics:

Total Project Budget	Total Funds Expended This Quarter	Percentage of Work Completed to Date
$600,000	$58,048	32%

Project Description:

The main objective of this proposed research is to provide state DOTs practical tools for supporting human-centered steel bridge inspection with real-time defect (e.g., fatigue cracks and corrosion) detection, documentation, tracking, and decision making. The proposed research will not only bridge the gaps identified in the IDEA project, but also expand the existing capability by developing AI algorithms for crack and corrosion detection. In addition to AR headsets, the project will also develop AR-based inspection capability using tablet devices. The tablet device can be used to perform AR-based inspection directly in a similar way to the AR headset. It can also leverage Unmanned Aerial Vehicles (UAV) for remote image and video acquisition during inspections, enabling bridge inspections from a distance in a human-centered manner.

Progress this Quarter
(includes meetings, work plan status, contract status, significant progress, etc.):

1. **Task 1: CV and AI algorithms for crack and corrosion inspection**
   This quarter, progress was made in the development and refinement of Meta Ensemble Learning techniques specifically designed for the automated inspection of structural defects like corrosion and cracking. The research focused on moving beyond traditional single-model architectures to leverage ensemble methods (specifically Stacking) to improve detection accuracy and robustness. Stacking involves training different models in parallel and using their collective predictions as inputs for a final meta-learner model, which learns the optimal strategy for combining these diverse outputs. This approach is particularly effective for complex infrastructure inspection because it can mitigate the individual biases and errors of standalone models.

2. **Task 2: AR-based software for human-centered bridge inspection**
   AR infrastructure inspection tool development has continued with significant progress being made. Persistent anchorage has now been successfully implemented, with anchors demonstrating reliable long-term stability across sessions and inspection spaces. With persistent anchorage solved, focus shifted onto database construction. The results database has begun development with the majority of core database functionality completed. All inspection results are now automatically stored within the database as a Binary Large OBject (BLOB) file, along with their creation dates, and are organized by inspection site. The database can be easily extracted from the Magic Leap headset, enabling inspectors to review and analyze inspection results more easily.

Anticipated work next quarter:

1. **Task 1: CV and AI algorithms for crack and corrosion inspection**
   For the upcoming quarter, the work will focus on refining the MetaCNN architecture to enhance its ability to handle complex structural patterns through the integration of mechanisms such as attention and residual blocks. These architectural improvements aim to prioritize critical defect regions and facilitate the training of deeper, more stable networks, specifically to address the lower mean IoU currently observed in crack detection. Parallel to these structural changes, research will be conducted into the implementation of a hybrid loss function that combines Binary Cross-Entropy, Dice Loss, and Focal Loss. This composite objective function is designed to improve pixel-level classification, maximize segmentation overlap, and mitigate the severe class imbalance inherent in infrastructure images where defect pixels are significantly outnumbered by background elements.
2. **Task 2: AR-based software for human-centered bridge inspection**
   During the upcoming reporting period, for AR headset, development will focus on the completion of database implementation and expanding its functionality. We plan to add the “time travel” feature that will allow inspectors to view inference results from specific inspection dates or quickly access the most recent inspection data for a given site. We will also spend time polishing the overall user experience to improve usability in the field.

Additionally, we will investigate alternative methods of saving and displaying inspection results for better long-term accuracy and reliability of stored data.
For the AR software environment on tablet devices and UAVs, we will downsample the visual mesh to improve its visualization when overlaid on RGB imagery. We will also optimize the computational efficiency of mesh construction support real-time streaming from the UAV to the tablet device.

Significant Results:

1. **Task 1: CV and AI algorithms for crack and corrosion inspection**

   The development of the MetaCNN architecture represents a significant shift from single-model frameworks to a robust Stacking ensemble approach for automated infrastructure inspection. As illustrated in Figure 1, this methodology utilizes three parallel variants of the YOLOv8 model (small (s), medium (m), and large (x)) which are trained on an 80-10-10% data split. By using models with different parameter counts, the system captures a diverse range of features. These base model predictions are then fed into a final meta-learner, which is specifically trained to combine these inputs into a more accurate final prediction for structural defects like corrosion and cracking.

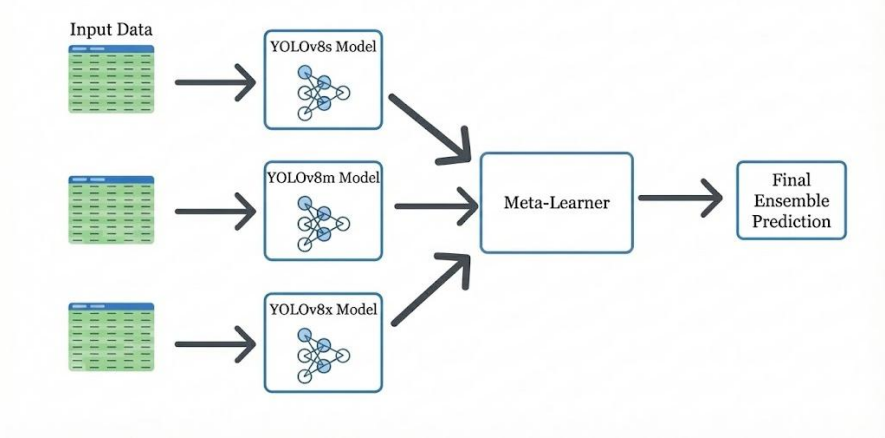


Figure 1: Stacking Ensemble Learning Framework

As shown in Figure 2, the meta-learner's internal structure is designed to refine these predictions through a sequence of specialized layers. It employs multiple convolutional stages, beginning with two layers of 128 channels followed by a 256-channel layer, all using 3x3 filters to extract high-level features. Each stage includes Batch Normalization to stabilize the feature maps and ReLU activation to introduce the non-linearity required for learning complex patterns. The network concludes with a 1x1 convolution without activation, providing the flexibility needed to output a continuous range of values for various regression tasks. To further enhance training, future iterations will implement a hybrid loss function combining Binary Cross-Entropy, Dice, and Focal Loss to better address class imbalances.

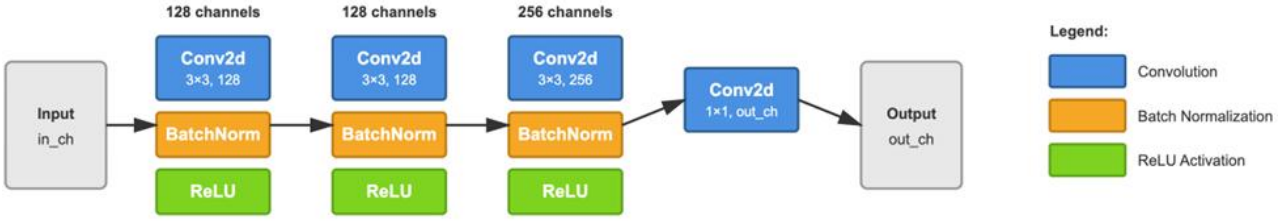


Figure 2: Architecture of the MetaCNN

The implementation of this Meta Ensemble approach yielded substantial improvements in defect detection accuracy. In corrosion detection tasks, the MetaCNN achieved a mean IoU of 69.8%, a significant increase

from the previous stacking benchmark of 55%. The model also reached a mean Dice score of 73.4% and an mAP50 of 67.5%. These results are supported by an expanded dataset of 1,079 corrosion images, which includes images from TxDOT and KDOT.

Table 1: Corrosion segmentation accuracy based on various metrics

Metric	Value
IoU_mean	69.8%
Dice_mean	73.4%
mAP50_mean	67.5%

The mean recall across the three classes for the test image set is 0.662, indicating that on average the MetaCNN model correctly detects about 66% of corrosion instances across all classes. Figure 3 shows two sample segmentation results, in which the model demonstrated high accuracies with an IoU as high as 91% and Recall 100% for the first image and an IoU of 69% and perfect 10083% recall rate for the second image.

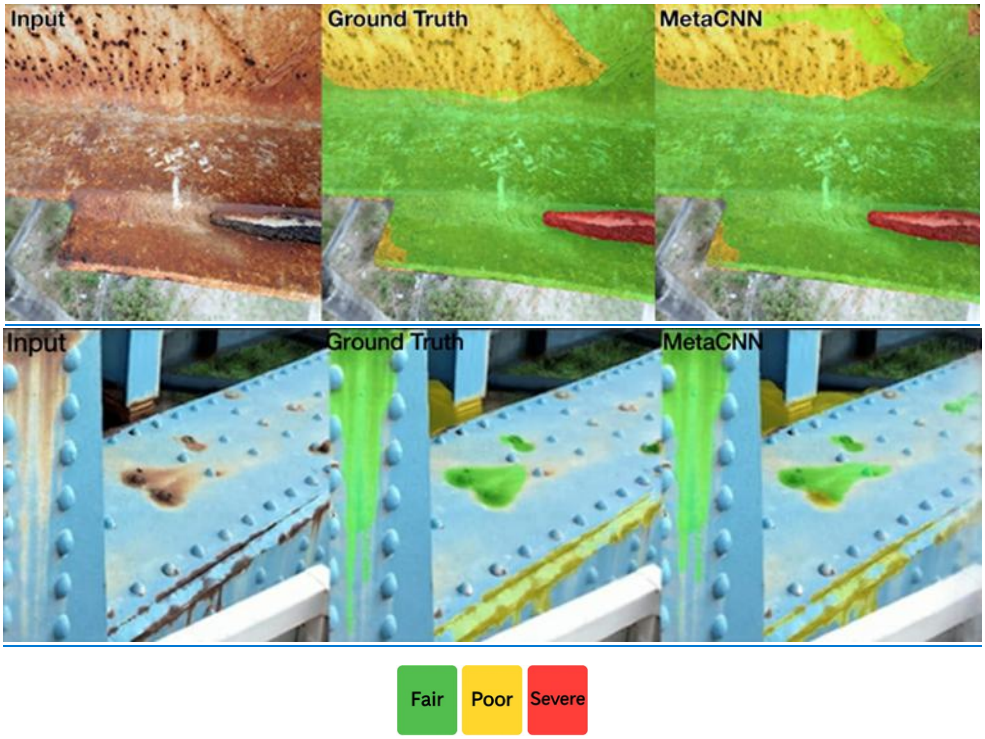


Figure 3. Sample corrosion segmentation results (left: Input, Middle: Ground Truth, Right: MetaCNN)

For crack detection, based on a dataset of 680 images, the ensemble learning approach still demonstrated measurable improvement. The architecture achieved a mean IoU of 44.1%, surpassing the previous benchmark of 37%. Other key metrics for crack detection included a mean Dice score of 49% and an mAP50 of 35.3%. While fine cracks present a higher level of difficulty, test scenarios presented in Figure 4 showed the MetaCNN architecture reaching an IoU of 75% with a 100% recall rate for the first image and an IoU of 53% with a 100% recall rate for the second image. Future work will focus on integrating other mechanisms such as attention and residual blocks to further improve these metrics. The mean recall for the test image set is 0.42 for crack detection, indicating that on average the MetaCNN model detects about 42% of crack instances.

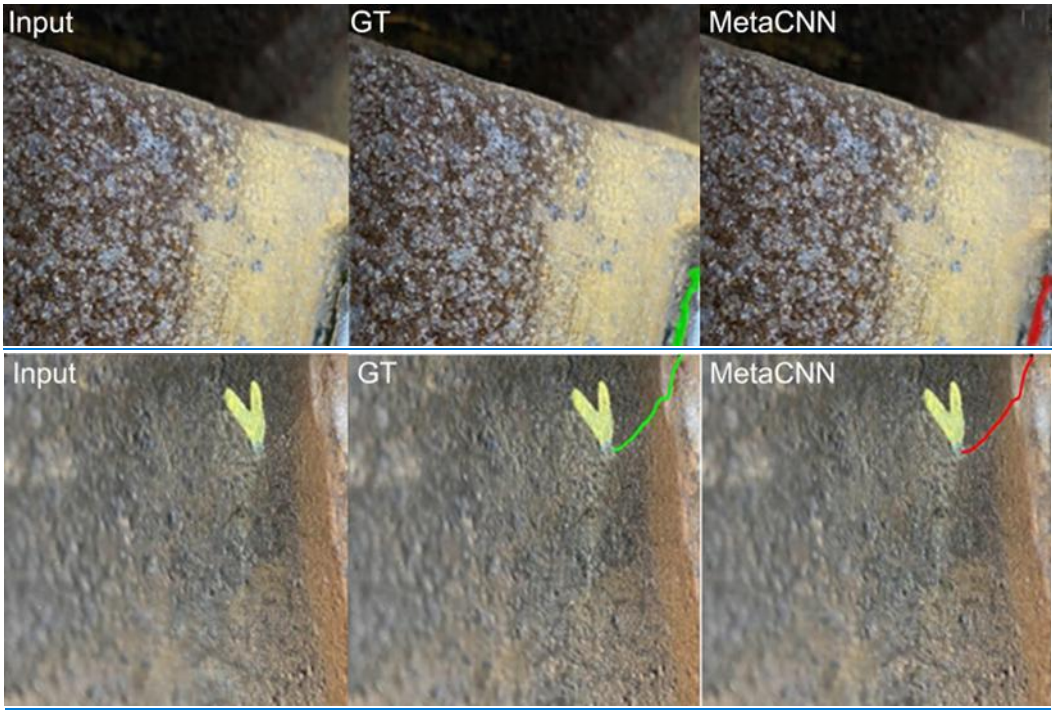


Figure 4. Sample crack segmentation results (left: Input, Middle: Ground Truth, Right: MetaCNN)

2. Task 2: AR-based software for human-centered bridge inspection

Subtask 2.3: AR software environment for AR headset

Significant progress has been made in advancing the AR inspection app on the Magic Leap 2 platform. This quarter a major milestone was achieved with the successful integration of Magic Leap's spatial anchors. Stable anchors are now created with every inspection result; they are stored long term and reloaded across inspection sessions, enabling consistent alignment of digital inference results.

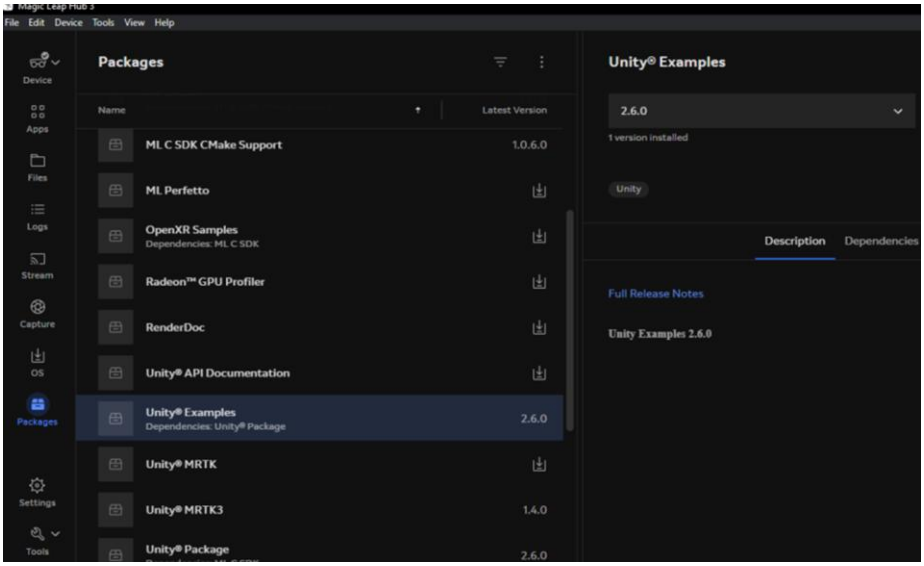


Figure 5. Magic Leap Hub Example Files

Previously, we faced challenges with integrating Magic Leaps spatial anchor storage system with our unity project due to a version mismatch in Unity. After consulting with the Magic Leap dev forum, we were informed that there are example files for spatial anchor storage inside newer Unity environments in the magic leap hub's example files, as shown in Figure 5.

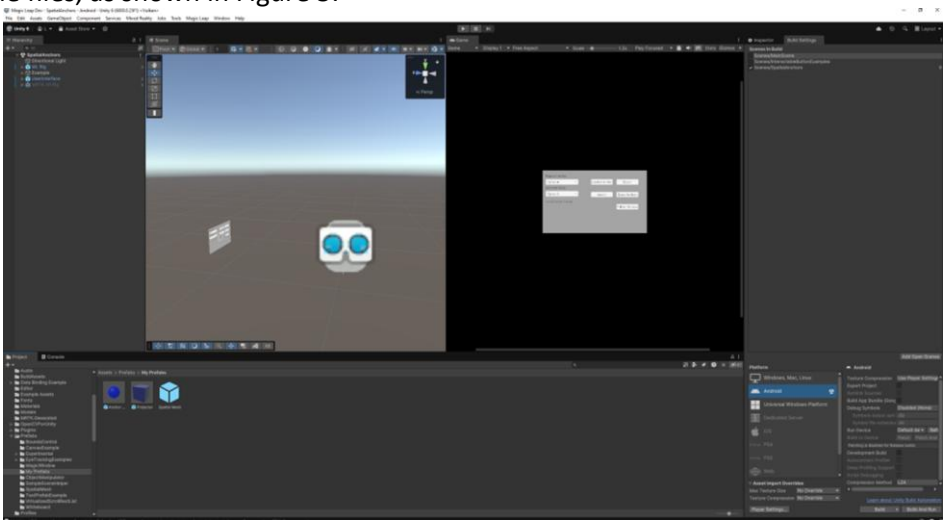


Figure 6. Example Scene

Inside the example files was an example unity scene, as seen in Figure 6. The scene contained the scripts and plugins as well as the scene contents/structure that was necessary for the spatial anchor storage system to work. We built this scene and deployed it on the headset for testing. Once it was seen that the storage was correct and reliable, we shifted focus to moving implementing the necessary changes from this scene into the inspection project to achieve the same functionality.

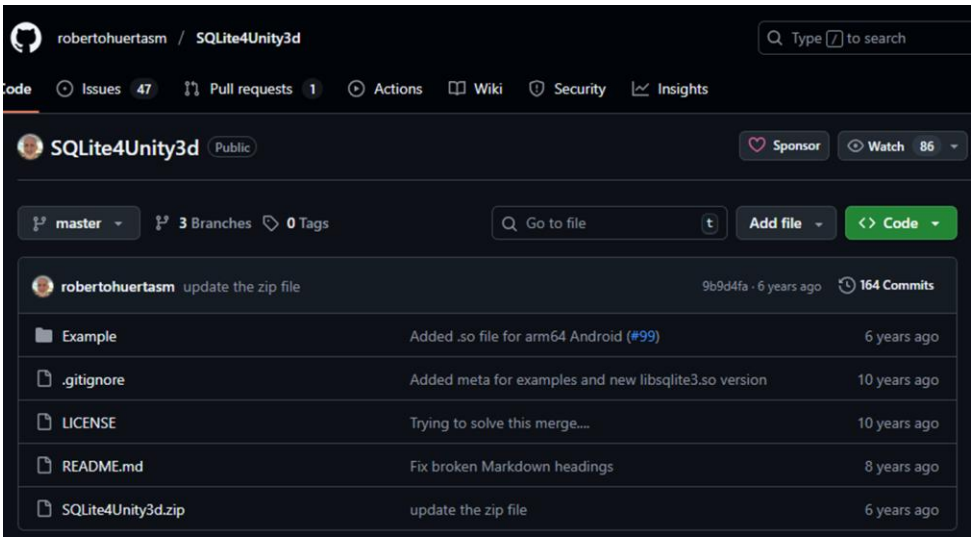


Figure 7. SQLite4Unity3d Github Page

With persistent anchorage solved, development continued to creation and management of the database for storing inspection results long term. For this purpose, we adopted a branch of SQL called SQLite for the purpose of running and managing small databases locally on a device. This decision was also driven by the fact that SQLite has been used previously to integrate with Unity, and there are various methods of doing so. We found that the most straight forward method of doing so was through a 3rd SQLite integration plugin available on GitHub. The GitHub repository featured in Figure 7 contained the necessary files to access read from and write to an SQLite database inside of unity, as well as examples on how to do so.

SQLite.db		
Bridge1		
Image Name (Anchor ID	Image Capture Date	Image BLOB
Image 1	11/20/2025	101010
Image 2	11/21/2025	101010
Image 3	11/22/2025	101010
Image 4	11/23/2025	101010
Image 5	11/24/2025	101010
Bridge2		
Image Name (Anchor ID	Image Capture Date	Image BLOB
Image 1	12/20/2025	101010
Image 2	12/21/2025	101010
Image 3	12/22/2025	101010
Image 4	12/23/2025	101010
Image 5	12/24/2025	101010
Bridge3		
Image Name (Anchor ID	Image Capture Date	Image BLOB
Image 1	1/20/2026	101010
Image 2	1/21/2026	101010

Fig 8. SQLite Database Example

With this plugin installed, we began construction of the database. We drafted a model for what the database would look like, as shown in Figure 8. The database would include multiple tables, one for each inspection site, inside each table the inferences are stored as BLOB files, that can be read as images. The inferences would be stored with its date of creation, as well as an anchor ID that corresponds to their spatial anchor, so that when the app loads a persistent spatial anchor (from a previous inspection) it can search for the anchors corresponding inspection result, and use it for projection.

Subsequently, the scripts necessary to query and insert into the SQLite database were created. The inference pipeline was altered to incorporate the new permanent solution for saving inference results, and the spatial anchors were also updated to draw their images from the SQL database.

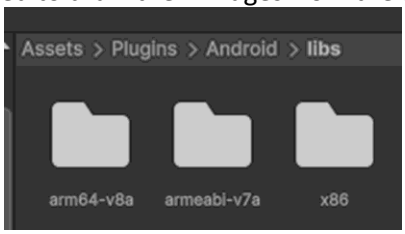


Figure 9. Missing x86-64 plugins for SQLite

After implementing all these changes, the updated pipeline was tested on the Magic Leap headset; however, an issue was encountered. The Magic Leap 2 runs on an x86-64 CPU. This means that in order to use SQLite4unity3d, we would have to find the appropriate ".so" plugin file. However, because this is a relatively rare type of CPU to be used on an android device, it was difficult to find a precompiled .so file online, and the GitHub page for SQLite4Unity3d only included the android .so files seen in Figure 9. To ensure the implementation of SQLite would work on the Magic Leap 2, we compiled a .so file for the x86_64 CPU, and we were able to test the program on the headset. The changes were successful, and the program is now functioning properly with the SQLite database integration.

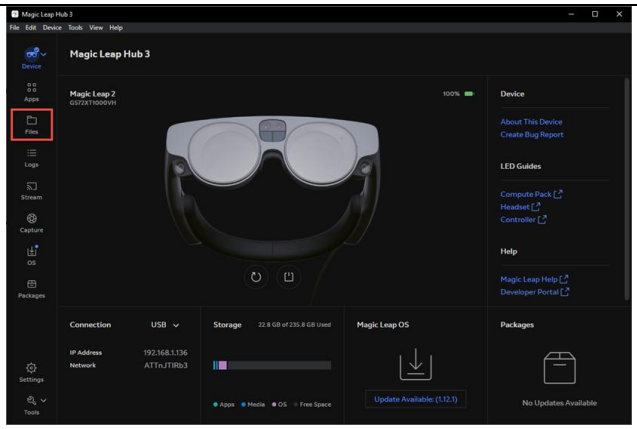


Figure 10. Database retrieval

Another important requirement for our database integration was that the inspector would be able to easily take the database from the headset to view the inspection results and allow for easy data transfer. The database retrieval process is quite straight forward, as seen in Figure 10, starting from the magic leap hub, the user can click the files icon, from there, they can navigate the headsets file structure. The file path to the database is as follows:" Android > data > org.inspectiontool > files > Database SQLite.db" (the name "org.inspectiontool" is subject to change and will likely not be the final package name for our app). From here the user can download the database to their device, and view the SQL database in a viewing app. Figure 11 shows sample inspection result viewed on a PC. The BLOB file can be downloaded as a .png file, and viewed as an image.

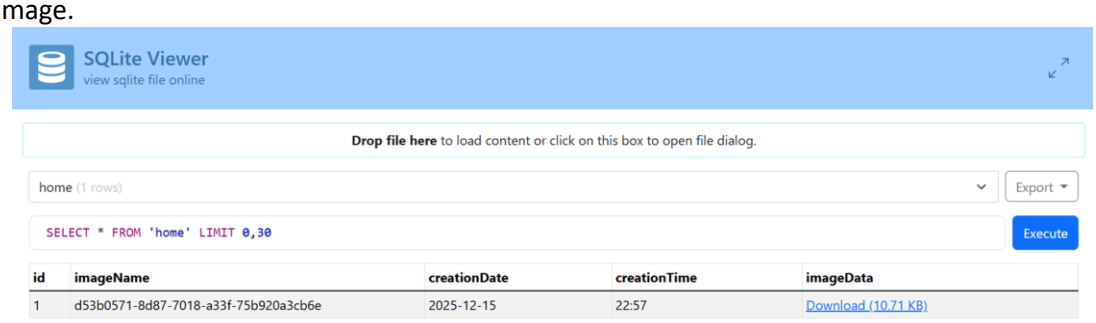


Figure 11. Database viewing

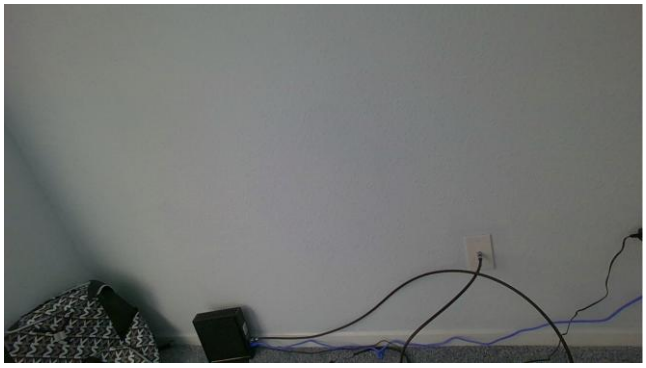
Subtask 2.4: AR software environment for tablet device and UAV

In this quarter, we continued investigating the use of a depth camera (Intel RealSense D455) for constructing 3D point clouds and mesh models of surrounding environments. By establishing correspondences between the reconstructed 3D mesh models and 2D images, the mesh can be accurately projected back onto the image plane, enabling reliable tracking of anchored objects in a 3D space.

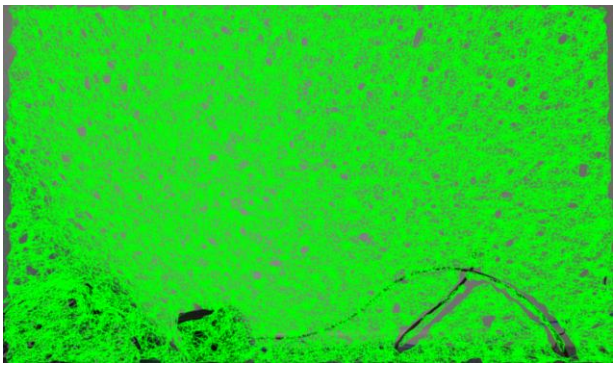
To evaluate the feasibility of mapping the constructed mesh models back to 2D images, we conducted a simple experiment in which a backpack was placed in the corner of a room and a video of the scene was recorded. From the video, RGB images and depth maps were extracted along with the camera intrinsic parameters. Using this information, 3D mesh models were reconstructed and subsequently projected back onto the corresponding 2D images. The results demonstrate that the reconstructed mesh aligns well with the image content, indicating that the mapping from 3D to 2D is feasible for object tracking.

Figure 5 illustrates the experimental results for two RGB images with their corresponding reconstructed 3D mesh models projected and overlaid onto the images. The mesh models are composed of triangular faces, with vertices corresponding to points in the reconstructed point cloud. This local mesh representation provides a stable 3D reference for the scene, allowing anchored objects to be tracked consistently in 3D space. Unlike 2D image features, which may change due to viewpoint variations or camera motion, the 3D mesh

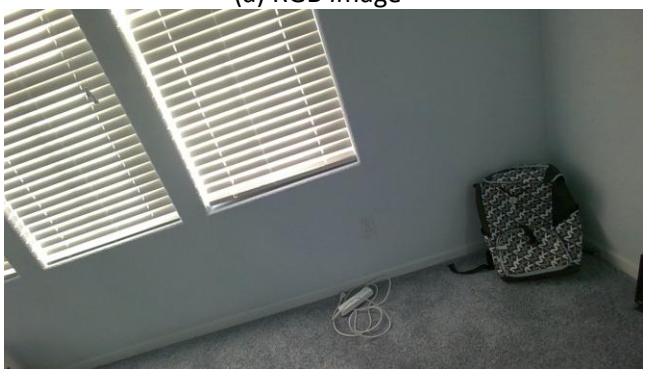
remains geometrically fixed in the environment. As a result, objects attached to or interacting with the mesh can be reliably tracked and visualized across video frames, enabling real-time interaction between the 2D video stream and the reconstructed 3D world.



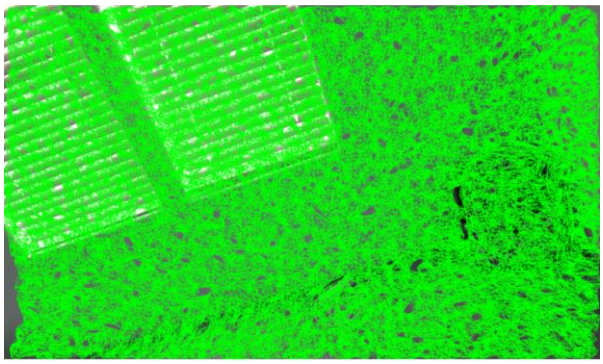
(a) RGB image



(b) 3D mesh mapped on the 2D image



(c) RGB image



(d) 3D mesh mapped on the 2D image

Figure 12. RGB images with their corresponding 3D mesh models overlaid onto the images

To develop the Unity user interface for the tablet-based AR inspection environment and enable intuitive interaction with visual inspection data, two main scripts were implemented in Unity. The first, TabletVideoManager, integrates prerecorded inspection videos into the interface, allowing users to load, play, pause, and switch between videos using simple UI controls. Video content is displayed on a tablet-style screen through Unity UI elements, ensuring smooth visualization within the AR environment. Event-driven logic manages video preparation and completion, updating button labels and screen states in real time to provide clear user feedback, which is essential for efficient and user-friendly field inspection workflows.

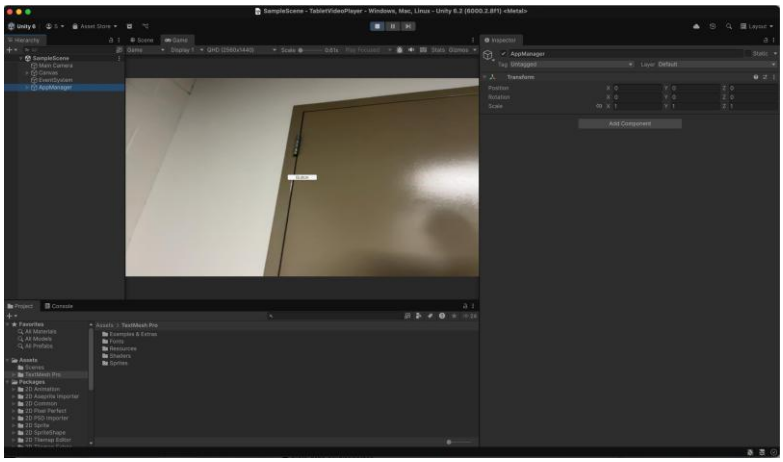


Figure 13. SimpleWebcamFeed in Unity for real-time visual input by streaming live camera.

The second script, SimpleWebcamFeed, as shown in Figure 13, provides real-time visual input by streaming live camera data directly onto the tablet interface. This enables the tablet to function as a live inspection and viewing tool, enhancing AR-based situational awareness by embedding real-world imagery into the Unity UI. The system automatically detects available camera devices, initializes the video stream, and safely releases camera resources when no longer in use. Together, these UI components form a foundational AR software environment for tablet devices, supporting both live and prerecorded visual data and facilitating effective interaction between the tablet interface and UAV-assisted inspection operations.

Circumstance affecting project or budget. (Please describe any challenges encountered or anticipated that might affect the completion of the project within the time, scope and fiscal constraints set forth in the agreement, along with recommended solutions to those problems).

N/A

Potential Implementation: