# Design and Development of a Remote User Interface for Multi-Robot On-site Component Inspection

**Juan Carlos Cruz Rivera[1], Mohsen Navazani[2], Ibukun Awolusi, Ph.D.[3], Ao Du, Ph.D.[4], and Jiannan Cai, Ph.D.(corresponding author)[5]**

[1]Department of Mechanical Engineering, The University of Texas at San Antonio, United States
[2]School of Civil Environmental Engineering, and Construction Management, The University of Texas at San Antonio, United States
[3]School of Civil Environmental Engineering, and Construction Management, The University of Texas at San Antonio, United States
[4]School of Civil Environmental Engineering, and Construction Management, The University of Texas at San Antonio, United States
[5]School of Civil Environmental Engineering, and Construction Management, The University of Texas at San Antonio, United States

juan.cruzrivera@utsa.edu, mohsen.navazani@utsa.edu, ibukun.awolusi.utsa.edu, ao.du@utsa.edu, jiannan.cai@utsa.edu

**Abstract -**

**This paper presents the design and development of an open-source browser-based interface for coordinating a multi-robot system for on-site construction inspections. The developed system enables remote human operators to effectively oversee and direct automated inspection tasks using both aerial and ground-based robotic platforms. The interface incorporates gaze-tracking technology for intuitive target selection and implements a containerized microservice architecture to ensure reliable deployment across different operating environments. The complete system architecture, implementation methodology, and a real-world inspection scenario are presented. The application has been released as open-source software to encourage adoption and further development by the construction automation community.**

**Keywords -**

**Human-machine interface; Construction inspection; Teleoperation; Human-robot interaction**

## 1 Introduction

Recent trends in infrastructure construction show a rising usage of off-site construction of components [1]. While off-site manufacturing facilities maintain rigorous quality control, the components can still suffer damage during transportation and on-site storage, requiring additional on-site inspection to ensure structural integrity before final assembly. On-site inspections, however, can be time-consuming, labor-intensive, and may not be as accurate as desired due to human error and limitations in inspection equipment [2].

Automated inspection systems have the potential to address these challenges by providing consistent, accurate, and efficient inspection capabilities. Robotic systems can be used to conduct the inspections in an efficient and automated manner [3]. Moreover, Artificial Intelligence (AI) enabled systems can leverage machine learning algorithms to better identify defects and anomalies [4], as well as navigate the challenging environments of construction sites [5].

In this paper, we present a human-machine interface for remote multi-robot on-site component inspection which capitalizes on the ongoing advancements in robotics and AI. Our application allows for the integration of rapidly acquired contextual data by one robot (e.g., drone) into an inspection mission for another robot (e.g., ground robot), all while allowing a human operator to monitor and direct the inspection process and provide additional context as needed.

We have also designed the application to allow for gaze-enabled inputs, providing an intuitive method for an operator to communicate intention and interest in a particular object.

In the following sections, we will provide further context on the use of robots in construction and infrastructure inspection, the need for human operators in automated systems, the use of gaze tracking for human input, and how the developed application fits into this context. We will then describe the design and development of the application, as well as an example use case. Performance evaluations and real-world case studies are planned for later work.

The interface's software has been released as open-source software under the MIT license on GitHub: `https://github.com/AIConLab/gaze_based_remote_user_interface`.

Our contributions to this paper are as follows:

- Development of a novel containerized architecture specifically designed for multi-robot construction inspection that ensures flexibility and portability.

- Integration of gaze-tracking as an intuitive human input method for remote inspections, which addresses challenges in traditional input methods on construction sites.
- Creation of a platform-agnostic interface human interface for coordinating aerial and ground robot inspection.
- Release of a complete open-source implementation that can serve as a foundation for further research and development in construction automation.

## 2 Background

GPS-based navigation systems have been a staple in outdoor robotic applications, allowing for precise navigation based upon a common coordinate system. Importantly, the common coordinate system allows the absolute positioning of a robot or multiple robots [6].

As reported in [7], ground robots have been more popular in the past for general civil infrastructure tasks, being able to do both contact and non-contact tasks, while aerial robots have been more popular for contactless tasks, such as visual inspection tasks of bridges, buildings, and other structures. Aerial robots, however, face fundamental challenges when it comes to extended usage and large payloads, limiting the amount of sensing and inspection equipment that can be carried and the time that the drone can be in the air [8].

Ground-based mobile robots, in contrast, can carry larger sensor payloads and operate for extended periods of time [9], but lack the mobility of aerial robots.

Combined approaches using both aerial and ground robots provide a more promising solution for infrastructure inspection tasks as these approaches can take advantage of the strengths of both systems as shown in [10].

In both fully autonomous and human-in-the-loop systems, human decision-making is still required at some stage of the system process, such as for providing the robots with an initial plan or additional context at the time of inspection.

Even in AI-enabled systems there exist limitations in their ability to generalize to new and unseen scenarios. In such cases, providing large amounts of training data could improve the performance of a model for a specific scenario, but this is not always possible in real-world applications as high-quality data can be difficult to collect [11]. Thus, human operators are often needed to provide additional context and guidance to the system to ensure that the inspection is conducted correctly and efficiently.

To provide human input into the system, gaze tracking has been shown to be one promising solution [12]. The incorporation of gaze tracking has the benefit of allowing the operator to provide input without the need for physical interaction, which can be important in situations where the operator is wearing gloves or is in a hazardous environment.

A user interface or visualization tool then allows the human user to manage how their input is processed and how the robot(s) respond to the input. One such visualization tool is Rviz [13], an open-source software package that allows the user to see navigation plans, robot transform frames, and more. Mission planning plugins for Rviz such as [14], create more sophisticated planning interfaces within Rviz allowing the human operator to view and plan for multi-robot systems.

Notably, these tools are designed to be used within the Robot Operating System (ROS) ecosystem, which is a popular framework for developing robotic applications. Our application differs in that it is designed to provide a standalone, browser-based interface that has the ability to communicate with ROS-based systems - but importantly does not require them to run.

An ongoing challenge in modern robotics research has been software replicability [15]. Our application provides a working example of one approach to providing software replicability through the use of Docker containers.

Docker is a platform for creating containerized applications. Containers contain everything needed to run a specific application, are lightweight, and are loosely isolated from the host machine allowing a developer to access host files, hardware, and resources when needed. A container will run an image instance, which is a snapshot of a certain environment, for example a specific version of Ubuntu with a specific version of Python. Docker Compose provides a tool for orchestrating multiple containers. A Docker Compose file and Docker image are extremely portable text-based files that allow an application to be built and run on almost any host machine.

## 3 Methodology

This section provides a high-level overview of the application's architecture, data handling, and software module descriptions. In the Implementation section, we will provide an example implementation of the application based on trials done with real-world robots. The application, however, is designed to be platform agnostic and can be used with any robot platform that provides the required input data and can receive the output data. These inputs and outputs are further described in the following section.

### 3.1 System Overview

The developed application allows for the input of GPS-based waypoint coordinates of structural inspection targets, along with the reference images of those targets. Video streams from a variety of sources can be passed

to the application for viewing by the user. Gaze and fixation data is also streamed to the application to allow for the gaze-based segmentation mask generation output when remotely inspecting a structure. The application outputs are directed at the ground robot which will be performing the remote inspection. The outputs include formatted Robot Operating System (ROS) 1 messages including mission commands, target reference images, the GPS waypoints, and the structure segmentation mask. Handling and formatting of the input and output data will be further discussed in the following sections. Figure 1 provides a visualization of the system's inputs and outputs, where the inputs are the incoming arrows and the outputs are the outgoing ones.

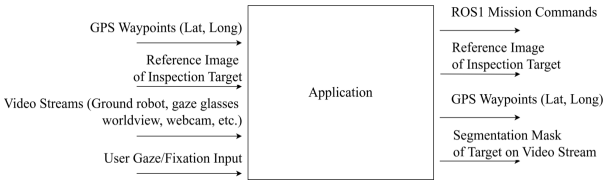


Figure 1. System inputs and outputs.

To allow for wide compatibility and guarantee software functionality across different systems, we containerized the application with Docker and separate module concerns to different Docker compose services. By separating services and images used, we have the flexibility to configure the software environment as needed to run specific packages. Figure 2 visualizes the described architecture, where the *App* contains the services defined in our Docker Compose file - namely *Mission Manager*, *Web Host*, *Segmentation*, *Module Processor*, and *Pupil Dev* services. Each service then contains modules that run certain functionality, for example, the *Frontend* and *Backend* modules within the *Web Host* service provide the live updating and rendering of the browser-based user interface. Arrows between each service represent data that is transferred asynchronously between containers. The modules within the application are further explained in the following sections.

## 3.2 Data Handling

Our architecture for the asynchronous messaging is inspired by that of ROS 1 [16]. Inter-container and inter-module message passing is achieved through an asynchronous ZeroMQ (ZMQ) publish-subscribe system using the ZMQ proxy pattern, wherein a central message broker facilitates the distribution of published messages to subscribers. The message broker acts as an intermediary that decouples publishers from subscribers, allowing for flexible and scalable communication between modules regardless of whether they are in the same container or not. For serialization and deserialization of messages, we use

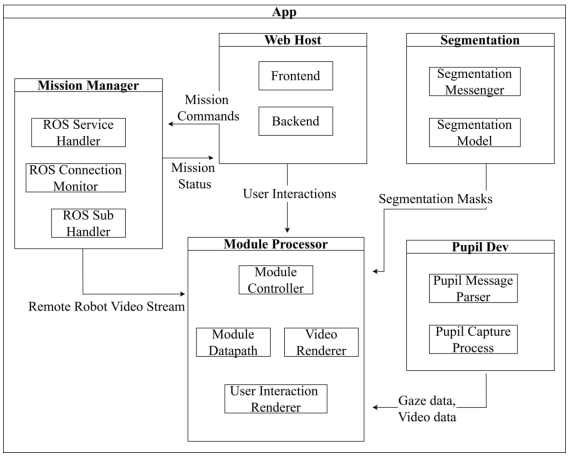


Figure 2. Docker compose services and module data flow within the application. The containers represent the services, and the modules are represented by the contained rectangles.

the MessagePack protocol.

## 3.3 Module Descriptions

### 3.3.1 Mission Manager

The GPS locations of targets to be inspected, along with their respective reference images, are inputted into the application through its local file system. Once loaded, the files are processed within the Mission Manager service which provides modules for interacting with the remote ROS robot. The Mission Manager service also contains modules for connection to a remote ROS Master URI, subscribing to published ROS data from the robot, and publishing data to the robot. The receiving robot and Mission Manager service require ROS service files for structuring the mission command formatting, remote file transfer, and mission state updates. These are included within the applications repository.

### 3.3.2 Module Processor

Most other data is routed through the Module Processor service which centralizes multi-container data flow. In particular, the Module Datapath serves as the central subscriber for video stream inputs and gaze data. Once data is received, it publishes the data to any subscribed modules.

Video topics from a remote robot or from a local camera would be published to the Video Render module for frame-by-frame processing depending on the application's processing state and the current video topic selection. The application processing state is managed by the Module Controller module, which receives commands generated by user interactions (button clicks, video topic selection,

etc.) from the Frontend. Based on the application's processing state, the Frontend will either render a live stream of a selected video topic via the Video Render module or provide a still image of the latest frame and rendered fixation point via the User Interaction Render module.

#### 3.3.3 Web Host

The Web Host service uses quart – a python based asynchronous web microservice based on Flask. The Frontend module handles the User Interface (UI) layout and user input capture from the web app, while the Backend module handles the data publishing and subscribing between different services.

#### 3.3.4 Pupil Dev

Gaze data refers to normal gaze detection from eye activity, while fixations refer to gaze in a localized region for a certain period of time. We use the fixation data as a point trigger for identifying an operator's point of interest from a screen. Gaze data and gaze-related video stream are handled in the Pupil Development service, in this case, the service is named specifically for the hardware used to capture the gaze data. This is discussed further in the Implementation section.

#### 3.3.5 Segmentation

The Segmentation services host the segmentation model and messaging modules. It is used to generate segmentation masks over images when a fixation point prompt is passed. The generated mask can then be used by the ground robot for target localization and specific area structural inspections.

### 3.4 User Interface

The User Interface (UI) is the primary means of interaction with the application. Figure 3 provides a skeleton of the interface design and the terms used to describe the regions in the UI. These regions include the *Mission Controls*, *Workspace*, and *Banner*.

The Workspace comprises the majority of the application display space, as it handles visualization of rendered video streams and display of fixation and segmentation outputs.

The Mission Controls section allows for the display of any mission state information, remote robot network status, and mission related utility functions.

The Banner offers state-driven button rendering based on the application's state as well as a persistent video topic selection drop down and a persistent button to toggle the gaze related functionality. The three states considered in this design are shown in Figure 4, where *Normal* is the normal operating state, *Fixation Captured* is the state when the Pupil Capture application detects a fixation on the screen, and the *Segmentation Results* state which activates once the segmentation masks have been generated, allowing a user to select which mask to send to robot during inspection time.

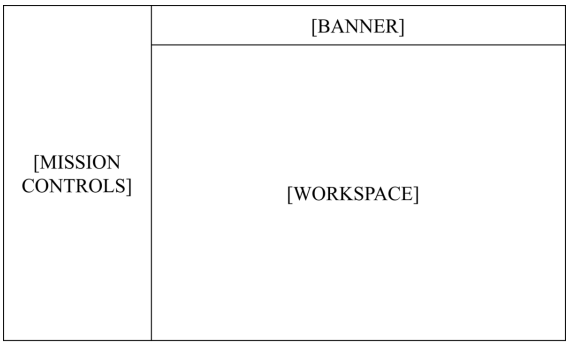


Figure 3. Skeleton of the UI layout.

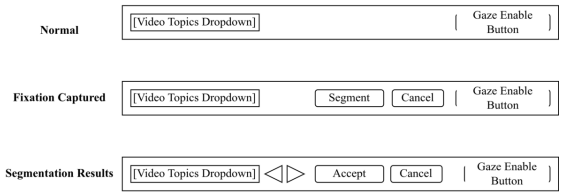


Figure 4. UI components display.

## 4 Implementation

As of the writing of this paper, the latest release of the application is V1.0.1. In the release, we have designed the application's implementation to function for the system shown in Figure 5. The arrows in Figure 5 describe the data flow between hardware as mentioned in Figure 1. The Ground Robot, Drone, and Gaze tracking hardware used in this test are as follows:

- Drone: DJI Matrice 30T
- Ground Robot: Clearpath Husky A200
- Gaze Tracking Hardware: Pupil Labs Core eye-tracking glasses

The User Computer block in Figure 5 defines the computer hardware used by the Remote Operator during testing. In this test, a Lenovo ThinkPad T490s with an Intel Core i7-8565U quad-core processor and 8 GiB of RAM is used. The laptop is running Ubuntu 24.04.2 LTS as the Operating System.

Further details of the implementation scenario and module implementation are described in the following sections.

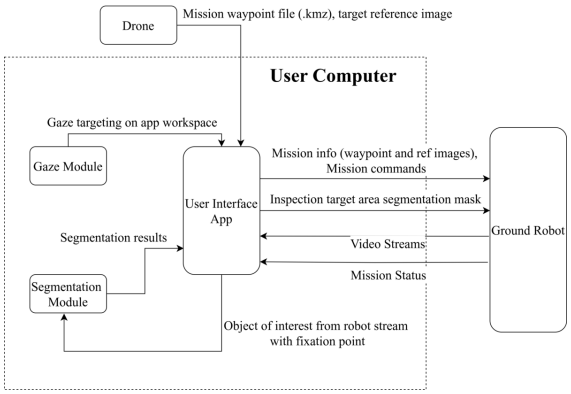


Figure 5. Overview of the system used in the implementation.

### 4.1 Task Scenario

In this inspection scenario, a remote operator is tasked with inspecting a mock bridge structure using a drone and a ground robot. We set up the inspection environment at the University of Texas at San Antonio's (UTSA) drone enclosure area, a 15000 square-foot and 60-foot-tall netted facility for drone testing. The wooden mock bridge was placed inside of the drone cage to act as our inspection object along with construction cones to add noise to the environment.

The remote operator is stationed outside of the cage with a laptop and the Pupil Labs Core eye-tracking glasses. An EAP225-Outdoor access point was set up to create a local wireless network between the ground robot and the operator's laptop. Images of this setup are shown in Figure 6. The entire inspection sequence is described in the following section.

(a) (b) (c)

Figure 6. (a) Outdoor testing environment showing the bridge structure and cones used for the scenario. (b) An image captured by the drone during the inspection mission. (c) The UGV and drone's starting position within the caged testing area.

### 4.2 Application Setup

Prior to starting the mission, the user must calibrate the Pupil Labs Core eye-tracking glasses and create a Surface in the Pupil Capture software [17]. The Surface is used to define the boundary of the gaze input area restricting it to a certain area which allows us to then extract exact pixel coordinates corresponding to that area. A Surface is defined by four Apriltags [18] which are rendered in the application's Workspace area and captured by the Pupil Core glasses' front-facing camera. These Apriltags are visualized in Figure 7(a)


Figure 7. (a) Apriltag rendered in the corner of the Workspace area of the application. (b) Teleoperation interface for manual UGV operation. (c) Mission control buttons and file viewer. (d) Mission and connection state.

In addition, the user must place a .env file in the .docker directory of the application. This .env file is used to define the IP address and hostname of the remote robot to connect to the ROS Master URI.

### 4.3 Mission Inspection Sequence

The operator must create a GPS-based waypoint of the bridge structure and capture a reference image using the drone. In our implementation, we use the DJI pilot app in order to save the waypoints as a KMZ file. The KMZ file contains a KML file from which we extract waypoint IDs and their respective latitude and longitude.

The reference images are stored under a directory named for their respective waypoint ID. The gathered drone data can then be placed in a directory within the application for formatting and processing.

When the operator selects "Process Mission Files" in the application, the Mission Manager service processes the GPS coordinates and reference images to prepare them for transmission to the UGV. The files are then wirelessly transferred over the local network when the "Mission Start" button is pressed. These buttons are shown in Figure 7 (c).

Upon receiving the files, the UGV converts the GPS coordinates into a local map point and creates a global and local navigation plan to autonomously navigate to the area. The operator can monitor the UGV's state (Figure 7 (d)) and video feed for any issues or anomalies while the robot navigates to the target location. If manual control of the robot is required at any point, the remote operator can teleoperate the ground robot's navigation base through the interface. The controls for this are shown in Figure 7 (b).

When the inspection target is reached, the operator directs the UGV to locate the inspection object of interest using the gaze-interface in the workspace. The UGV will wait until an inspection target area is specified by the remote operator.

The inspection target area is generated by the gaze-enabled input feature of the application. When the gaze is enabled, Apriltags are rendered in the workspace area of the application and overlaid on the video stream. The Apriltags create a bounded area to filter gaze/fixation points so that they correspond directly to pixel coordinates in the video stream frame. An example is shown in Figure 7 (a).

When a fixation is detected, the application stops rendering new video frames and displays the latest frame and respective fixation. The user can then accept the fixation as an input to the segmentation module or attempt the fixation process again.

An image and fixation point sent to the segmentation module are passed to the Segment Anything 2 model (SAM2) [19] which outputs three segmentation mask predictions. These are then displayed to the user in the application's Workspace area, where they can select a mask or reattempt the fixation process. An example of a segmentation mask is displayed in Figure 8. Here the mask is shown as an orange overlay over a pillar of the mock bridge the remote operator wishes to inspect. Within the mask, a red dot shows the specific point the user's fixation was captured at.

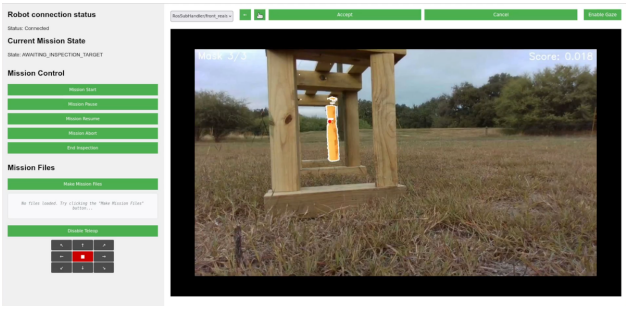


Figure 8. An example of a segmentation mask generated after fixation. The orange area depicts the target area, while the red dot depicts the fixation point.

With a target surface area mask received, the UGV could begin the required inspection and data-capturing process. When the inspection of the previous mask is complete the operator could provide another inspection target area or terminate the inspection and move on to the next waypoint. If no more waypoints were available, then the mission would be completed.

The entire mission sequence is visualized in Figure 9, showing a sequence chart with the Drone, Operator, and the UGV as actors. In the chart, the 3 phases of the implemented inspection process are the *Data Gathering Phase*, *Mission Execution Phase*, and *Inspection Phase*. The arrows show actions each actor takes from start to finish. Loops are shown as dashed rectangles that continue until an exit condition.

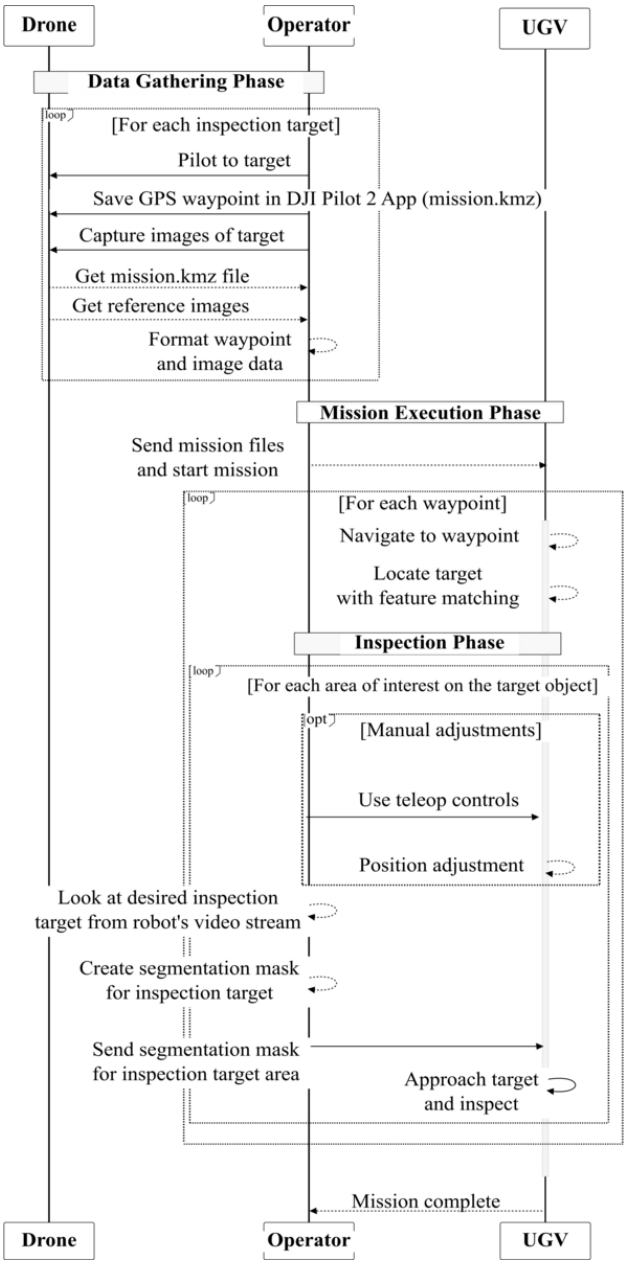


Figure 9. Mission Sequence

## 5 Limitations

The current approach assumes the drone can safely gather images and a GPS coordinate for the inspection

targets and that the ground robot can navigate to those targets. A mission may still be possible if image data can be captured of the objects, and the GPS coordinate for waypoints can be formatted to a KMZ file, but this approach requires additional effort by the operator and is not the intended approach for our implementation.

Image data captured by the first robot (the drone in our example) may lose relevancy over time if, for example, the images were captured at a different time of day or there was a significant scene change from when the initial images were captured.

## 6 Conclusions

On-site construction inspection is critical to ensure the quality and durability of infrastructure structures. On-site inspections, however, can be time-consuming, labor-intensive, and may not be as accurate as desired due to human error and limitations in inspection equipment. Automated inspection systems have the potential to address these challenges by providing consistent, accurate, and efficient inspection capabilities. Robotic systems can be used to conduct the inspections in an efficient and automated manner. However, due to the complexity of construction sites, human operators are often needed to provide additional context and guidance to the system to ensure that the inspection is conducted correctly and efficiently.

In this work, we provided an architecture overview and dataflow description of the developed human-machine interface application for multi-robot on-site component inspection. Our application allows for the integration of rapidly acquired contextual data by one robot (e.g., drone) into an inspection mission for another robot (e.g., ground robot), all while allowing a human operator to monitor and direct the inspection process and provide additional context as needed. An implementation of the application was demonstrated in a real-world scenario, showing the application's functionality and user interface design.

In future studies, we plan to run more inspection case studies to evaluate the application's usability and impact on productivity in more realistic inspection scenarios.

The current implementation serves as a functional base for use in novel inspection applications where the software can be easily modified to extend functionality and hardware compatibility. We encourage any interested parties to incorporate the applications into their projects.

## 7 Acknowledgement

This research was funded by the Transportation Infrastructure Precast Innovation Center (TRANS-IPIC), Tier 1 University Transportation Center (UTC) via Project UT-23- RP-02, and the U.S. National Science Foundation (NSF) via Grants 2138514 and 2222670. The authors gratefully acknowledge the support. Any opinions, findings, recommendations, and conclusions in this paper are those of the authors and do not necessarily reflect the views of TRANS-IPIC, NSF, and The University of Texas at San Antonio.

## References

[1] R. H. Assaad, I. H. El-adaway, M. Hastak, and K. LaScola Needy. Quantification of the state of practice of offsite construction and related technologies: Current trends and future prospects. *Journal of Construction Engineering and Management*, 148 (7):04022055, 2022. doi:10.1061/(ASCE)CO.1943-7862.0002302.

[2] B. M. Phares, G. A. Washer, D. D. Rolander, B. A. Graybeal, and M. Moore. Routine highway bridge inspection condition documentation accuracy and reliability. *Journal of Bridge Engineering*, 9(4):403–413, 2004. doi:10.1061/(ASCE)1084-0702(2004)9:4(403). URL https://ascelibrary.org/doi/abs/10.1061/%28ASCE%291084-0702%282004%299%3A4%28403%29.

[3] J. Seo, L. Duque, and J. P. Wacker. Field application of uas-based bridge inspection. *Transportation Research Record*, 2672(12):72–81, 2018. doi:10.1177/0361198118780825.

[4] S. Lee, S. Kwon, M. Jeong, S. Hasan, and A. Kim. Automated on-site quality inspection and reporting technology for off-site construction(osc)-based precast concrete members. In *Proceedings of the 37th International Symposium on Automation and Robotics in Construction (ISARC)*, pages 1152–1159, Kitakyushu, Japan, October 2020. International Association for Automation and Robotics in Construction (IAARC). ISBN 978-952-94-3634-7. doi:10.22260/ISARC2020/0158.

[5] D. Shah, A. Sridhar, N. Dashora, K. Stachowicz, K. Black, N. Hirose, and S. Levine. Vint: A foundation model for visual navigation, 2023. URL https://arxiv.org/abs/2306.14846.

[6] X. Gao, J. Li, L. Fan, Q. Zhou, K. Yin, J. Wang, C. Song, L. Huang, and Z. Wang. Review of wheeled mobile robots' navigation problems and application prospects in agriculture. *IEEE Access*, 6:49248–49268, 2018. doi:10.1109/ACCESS.2018.2868848.

[7] X. Hu and R. H. Assaad. The use of unmanned ground vehicles (mobile robots) and unmanned

aerial vehicles (drones) in the civil infrastructure asset management sector: Applications, robotic platforms, sensors, and algorithms. *Expert Systems with Applications*, 232:120897, 2023. ISSN 0957-4174. doi:https://doi.org/10.1016/j.eswa.2023.120897. URL https://www.sciencedirect.com/science/article/pii/S0957417423013994.

[8] R. Hadidi, B. Asgari, S. Jijina, A. Amyette, N. Shoghi, and H. Kim. Quantifying the design-space tradeoffs in autonomous drones. In *Proceedings of the 26th ACM International Conference on Architectural Support for Programming Languages and Operating Systems*, ASPLOS '21, pages 661–673, New York, NY, USA, 2021. Association for Computing Machinery. ISBN 9781450383172. doi:10.1145/3445814.3446721. URL https://doi.org/10.1145/3445814.3446721.

[9] Q. Liu, Z. Li, S. Yuan, Y. Zhu, and X. Li. Review on vehicle detection technology for unmanned ground vehicles. *Sensors*, 21(4), 2021. ISSN 1424-8220. doi:10.3390/s21041354. URL https://www.mdpi.com/1424-8220/21/4/1354.

[10] S. Halder and K. Afsari. Robots in inspection and monitoring of buildings and infrastructure: A systematic review. *Applied Sciences*, 13(4), 2023. ISSN 2076-3417. doi:10.3390/app13042304. URL https://www.mdpi.com/2076-3417/13/4/2304.

[11] G. Fobiri, I. Musonda, and F. Muleya. Reality capture in construction project management: A review of opportunities and challenges. *Buildings*, 12(9), 2022. ISSN 2075-5309. doi:10.3390/buildings12091381. URL https://www.mdpi.com/2075-5309/12/9/1381.

[12] F. Nenna, D. Zanardi, and L. Gamberini. Enhanced interactivity in vr-based telerobotics: An eye-tracking investigation of human performance and workload. *International Journal of Human-Computer Studies*, 177:103079, 2023. ISSN 1071-5819. doi:https://doi.org/10.1016/j.ijhcs.2023.103079.

[13] H. R. Kam, S.-H. Lee, T. Park, and C.-H. Kim. Rviz: a toolkit for real domain data visualization. *Telecommunication Systems*, 60:337–345, 2015.

[14] R. Sakagami, S. G. Brunner, A. Dömel, A. Wedler, and F. Stulp. Rosmc: A high-level mission operation framework for heterogeneous robotic teams. pages 5473–5479, 2023.

[15] F. Bonsignorio and A. P. del Pobil. Toward replicable and measurable robotics research [from the guest editors], 2015.

[16] M. Quigley, K. Conley, B. Gerkey, J. Faust, T. Foote, J. Leibs, R. Wheeler, and A. Y. Ng. Ros: an open-source robot operating system. In *ICRA workshop on open source software*, number 3.2, page 5, 2009.

[17] M. Kassner, W. Patera, and A. Bulling. Pupil: An open source platform for pervasive eye tracking and mobile gaze-based interaction. In *Adjunct Proceedings of the 2014 ACM International Joint Conference on Pervasive and Ubiquitous Computing*, UbiComp '14 Adjunct, pages 1151–1160, New York, NY, USA, 2014. ACM. ISBN 978-1-4503-3047-3. doi:10.1145/2638728.2641695. URL http://doi.acm.org/10.1145/2638728.2641695.

[18] E. Olson. Apriltag: A robust and flexible visual fiducial system. In *2011 IEEE International Conference on Robotics and Automation*, pages 3400–3407, 2011. doi:10.1109/ICRA.2011.5979561.

[19] N. Ravi, V. Gabeur, Y.-T. Hu, R. Hu, C. Ryali, T. Ma, H. Khedr, R. Rädle, C. Rolland, L. Gustafson, E. Mintun, J. Pan, K. V. Alwala, N. Carion, C.-Y. Wu, R. Girshick, P. Dollár, and C. Feichtenhofer. Sam 2: Segment anything in images and videos, 2024. URL https://arxiv.org/abs/2408.00714.