Leveraging Aruco Fiducial Marker System for Bridge Displacement Estimation Using Unmanned Aerial Vehicles

Mohamed Aly

University of Nebraska-Lincoln, mohamed97010@huskers.unl.edu

LEVERAGING ARUCO FIDUCIAL MARKER SYSTEM FOR BRIDGE DISPLACEMENT ESTIMATION USING UNMANNED AERIAL VEHICLES

by

Mohamed Aly

A THESIS

Presented to the Faculty of

The Graduate College at the University of Nebraska

In Partial Fulfilment of Requirements

For the Degree of Master of Science

Major: Computer Science

Under the Supervision of Professor Carrick Detweiler

Lincoln, Nebraska

May, 2023

LEVERAGING ARUCO FIDUCIAL MARKER SYSTEM FOR BRIDGE DISPLACEMENT ESTIMATION USING UNMANNED AERIAL VEHICLES

Mohamed Aly, M.S.

University of Nebraska, 2023

Advisor: Carrick Detweiler

The use of unmanned aerial vehicles (UAVs) in construction sites has been widely growing for surveying and inspection purposes. Their mobility and agility have enabled engineers to use UAVs in Structural Health Monitoring (SHM) applications to overcome the limitations of traditional approaches that require labor-intensive installation, extended time, and long-term maintenance. One of the critical applications of SHM is measuring bridge deflections during the bridge operation period. Due to the complex remote sites of bridges, remote sensing techniques, such as camera-equipped drones, can facilitate measuring bridge deflections. This work takes a step to build a pipeline using the state-of-the-art computer vision ArUco framework to detect and track ArUco tags placed on the area of interest. The proposed pipeline analyzes videos of tags captured by stationary cameras and camera-equipped UAVs to return the displacements of tags. This work provides experiments of the ArUco pipeline with stationary and dynamic camera platforms in controlled environments. Estimated displacements are then compared with ground truth data. Experiments show the significance of pixel resolution, platform stability, and camera resolution in achieving high accuracy estimation. Results demonstrate that the ArUco pipeline outperforms existing methods with stationary cameras, reaching an accuracy of 95.7%. Moreover, the pipeline introduces an approach to eliminating the noised cause drone's motion using a static reference tag.

This technique has yielded an accuracy of 90.1%. This work shows promise toward a completely targetless approach using computer vision and camera-equipped drones.

ACKNOWLEDGMENTS

We want to thank the University of Nebraska-Lincoln, School of Computing, Department of Civil and Environmental Engineering, and members of the Nebraska Intelligent MoBile Unmanned Systems (NIMBUS) Laboratory.

Great appreciation for all of my committee members for their time and guidance. I would like to thank Dr. Motassem Al-Tarazi for being the third committee member and providing feedback on my work. I want to thank my advisors, Dr. Detweiler and Dr. Sim, for their continuous support and patience in guiding me to become a better researcher.

Lastly, I would like to thank Ji Young Lee, NIMBUS Lab PhD student, for helping with the drone flights and consultation through this research.

GRANT INFORMATION

This research was financially supported by the NSF-BD Spokes Program (Award: 1762034), NSF-IIS (Award: 1925052), NSF (Award: $25 - 0511 - 0206 - 001$), and DoD USACE MADS-OPP (Award: $W912HZ21C0060$).

Table of Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Motivation

Deflection measurement for bridges is useful in structural health monitoring (SHM) during the bridge service life [3] [4]. Throughout the years, conventional contact sensors, such as Linear Variable Differential Transducers LVDTs [5], accelerometers [6], and global positioning system (GPS)-based systems [7], have been the main techniques for displacement measuring. Despite the success of the traditional sensor-based methods in monitoring the response of structures, they pose several practical challenges. In addition, they involve time and labor-intensive installation processes and seek substantial maintenance to achieve long-term monitoring and maintenance. Moreover, the structure of interest might not be accessible for installing instruments due to complex site conditions. These limitations affect the repeatability, practicality, and automation of such systems. Thus, the critical need for remote sensing techniques, such as camera platforms, to estimate bridge displacements has emerged. These techniques have the ability to provide non-contact inexpensive, and reliable alternatives to measure displacements.

Over the past decades, unmanned aerial vehicles (UAVs) have been used in many real-world applications; package delivery, remote sensing, and remote video surveillance are among the most popular. Additionally, Unmanned Aerial Vehicles (UAVs) have been deployed for inspecting bridges [8]. Combining the UAV technology with remote sensing techniques can significantly improve the efficiency of the entire bridge inspection process ranging from data collection and analysis to decision-making [9].

This research pursues to combine the use of computer vision techniques with camera-equipped UAVs to enhance the process of estimating bridge displacements. This improvement can occur by creating a pipeline of capturing videos of the structure of interest, analyzing it, and outputting the changes in displacements. As shown in Figure 1.1, a camera-equipped UAV is deployed to measure tag displacements moved by an arbor press using computer vision ArUco framework. The videos captured by the UAV are entered into a pipeline to return the tag displacements in inches. We expand on this setup more in Chapter 5.

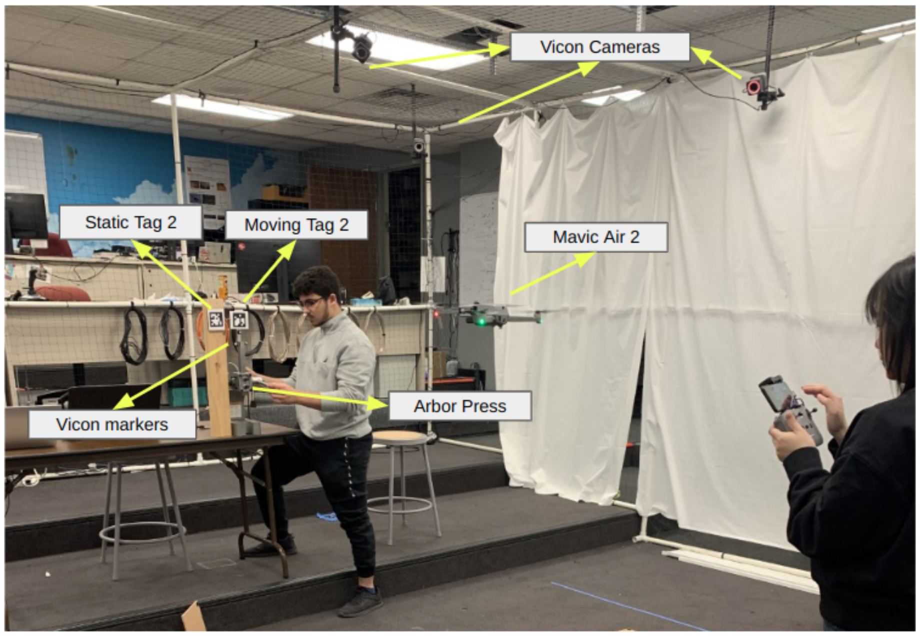


Figure 1.1: Experimentation setup for estimating tag displacement using a camera-equipped drone (Mavic Air 2)

However, several complicated challenges are associated with building such a platform. First, the drone's motion introduces noise into the estimated bridge displacement. Second, as the bridge displacements are minute, in millimeters, the system's accuracy must be within a millimeter. Third, environmental changes such as illumination changes, angle of perspective, and surface conditions can affect the estimation performance of computer-vision-based techniques. Prior work has proposed frameworks that try to overcome these challenges through various scenarios. However, none were able to overcome some of these challenges completely. We discuss these techniques and their limitations in Chapter 2.

Most existing vision-based displacement measurement methods assume that the camera is stationary [10]. Unlike prior work, this thesis aims to employ a state-of-the-art ArUco Fiducial Marker System pipeline to track tags placed on structures of interest in videos captured by camera-equipped drones. To reach this goal, several research questions have been proposed in section 1.2.

1.2 Research Questions

1. **Can we develop a proof-of-concept pipeline that utilizes state-of-the-art computer vision techniques to track bridge displacements captured from camera-equipped drones?** As an emerging technique in tracking and localization for many robotics applications, we *hypothesize* that leveraging fiducial markers can show promise in estimating displacements with a drone. This will be accomplished by employing ArUco fiducial marker system to place tags on the area of interest. In consequence, we deploy a low computational, robust to illumination changes, high detection rate fiducial system. These advantages will enable us to achieve better accuracy in estimating the

displacements of the targeted area.

In addition, we track the marker by engineering a pipeline of analyzing the video captured by the drone, tracking the ArUco tag in each frame and reporting the pixel location, getting the pixel displacements, and finally converting from pixels to inches to get the actual displacements in inches. We compare the results with the achieved ground truth and report the root mean square error (RMSE) of the recorded displacements. Such a system needs to be robust and hardened to detect minimal displacements. Chapter 3 explains the pipeline approach with specifications of the used tools. Chapter 5 answers this question in detail and show the steps taken to overcome the challenges accompanied by this problem.

2. **While fiducial marker systems have proven successful in tracking and localization of several robotics applications, how accurate are they in tracking such minimal displacements?** In order to answer this question, we take a step back from initially testing the ArUco pipeline on a drone. Instead, we perform an extensive study of experimenting with the ArUco detector on videos captured by stationary cameras. We use two different types of cameras in two different ways to track ArUco tags. The first approach uses Canon Vixia HF G50 to record videos of the marker placed on the targeted area of measuring. These videos are then entered through the ArUco detector pipeline to analyze each frame and report the pixel locations of the tag. The second approach utilizes the ArUco real-time detector by integrating a Blackfly S camera sensor with an ArUco Robot Operating System (ROS) node. This approach detects ArUco tags on demand and reports the pixel location of the tag in each frame. We then take these locations to get the pixel

displacements and convert them to inches. We deploy these approaches in laboratory studies for Stationary Experimentations. Chapter 4 expands on these studies and answers this question.

Experimenting with stationary cameras aims to report the highest obtained accuracy from this approach. Things will only get worse when operating the pipeline on videos captured by a drone. These studies give us the boundary of how well the ArUco detector can estimate such small displacements. Thus, we benefit from these studies' conclusions to deploy the ArUco pipeline on a UAV. Chapter 5 talks about the experiments performed using a UAV.

3. **What pixel-to-inch ratio is required to accurately detect tags with small displacements? Does the camera platform's resolution, zoom level, distance from the target, and motion on the platform affect the robustness of the estimation?** As these two questions are related, we hypothesize answering both by replicating the Stationary experiments mentioned in the previous answer at different distances and with various recording settings. In doing this, we acquire the pixel-to-inch ratio required to detect the displacement needed. The pixel-to-inch ratio is changeable based on the lens zoom level, the camera's resolution, and the distance from the target. Thus, we need to adjust these factors to learn the required ratio. Additionally, we do the same with drone experiments. We replicate the same test at different distances and zoom levels to understand how that affects the total robustness of the system in estimating displacements. Chapters 4 and 5 answer these questions in detail and expand on the related topics.

4. **How can the drone's motion noise be removed to reach the true estimated displacement of the tag?** Drone stability depends on several factors, such as

wind speed, the robustness of the drone's controller, calibration of the drone sensors, and the hardening of the drone's camera gimbal to compensate for the drone's movements. We show how these factors can affect the estimation of displacements by comparing the results of deploying two different drone types to detect a stationary tag in Chapter 5.

To compensate for the drone's motion, we compare the two approaches. The first approach is to track the drone's movements with Vicon motion-capturing system and clear the motion noise from the tracked tag displacements. For the second approach, we place a stationary reference tag in the drone's camera field of view. We also track the static tag in our ArUco pipeline and subtract the noise of the stationary tag from the moving tag. These two approaches are distinct as the first one uses the drone's raw motion to estimate the tag's actual displacements, while the other counts for the estimation error of the detector in detecting the markers. Chapter 5 compares the two approaches and shows the different results each one gives.

1.3 Research Contributions

This research mainly contributed the following to the area of utilizing UAVs to estimate bridge displacements:

1. **Development of a novel pipeline leveraging the state-of-the-art ArUco fiducial marker system to measure bridge displacements in videos captured by stationary camera platforms as well as camera-equipped UAVs.**
   This pipeline returns the bridge displacements in inches to compare with ground truth with accuracy of **95.7**% using a stationary camera and **90.1**% using a camera-equipped drone.

2. **Laboratory experiments and results to demonstrate the accuracy level of estimating displacements by stationary cameras and drones.**
   These studies include the effect of zoom level, camera resolution, and the distance to the target on the pipeline's accuracy. In addition, these studies deliver the pixel-to-inch ratio required to get the accuracy level needed to capture the targeted displacement.

3. **Development of an approach compensating for UAV's motion to estimate the targeted area's displacements using a static reference tag with an accuracy of 90.1%.**
   This development is a result of a comparison study conducted to compare two different techniques to clear the noise created by the UAV's motions and vibrations.

1.4 Document Overview

This thesis is organized as follows: Chapter 2 provides a literature review of published related work, Chapter 3 discusses and analyzes the different components of the proposed pipeline. Chapter 4 provides results of testing the pipeline with stationary camera platforms to answer research questions 2 and 3, Chapter 5 covers research questions 1 and 4 by showing the experiments of using camera-equipped drones, and Chapter 6 summarizes the discussion of all results and provides conclusions of the work done.

Chapter 2

Related Work

Structural Health Monitoring of bridges using traditional contact measurement sensors, such as Linear Variable Differential Transducers (LVDTs) [5] and accelerometers [11] [6] [12], has always been challenging as they require personnel to climb the bridge and install them. LVDTs require the installation of scaffolds next to the bridge [5], which may not be possible in some inaccessible locations. In some unique locations, like high-speed railway bridges, it is not allowed to install the measurement equipment during the operation period [13]. On the other hand, accelerometers are used to measure displacement through double integration. However, they are usually accompanied by numerical integration errors [14]. Thus, to avoid these limitations, bridge inspectors are interested in new non-contact reference-free methods to measure the displacements. Global positioning systems (GPS) [15] offer potential advantages due to their non-contact nature, but they are limited in application due to the high cost of these systems. Prior work offers promising cost-effective non-contact approaches to measuring vertical displacements of bridges during the bridge operation period.

Stationary cameras and Unmanned Aerial Vehicles (UAVs) can be used as remote sensing techniques to develop non-contact methods. The aid of computer vision and machine learning has facilitated full-field monitoring of structures [16]

[17] [18] [19]. However, measuring vertical displacements generally depends on the camera's resolution, the motion of the camera platform, target or targetless feature point extraction, and the choice of vision algorithms for measurement extraction [16] [17] [18] [19]. Using deep-learning-based full-field optical flow is proposed in [20] by utilizing the CNN FlowNet and FlowNet2.

UAV-based analysis monitoring has gained momentum, over stationary cameras, for bridge monitoring in recent years due to improved accessibility and cost efficiency, avoidance of traffic closure, and reduced safety hazards during the inspection process [21] [22]. However, one limitation of UAVs is the introduction of motion into measured displacements. One way to overcome this limitation is by tracking the movement of the UAV using a fixed reference and eliminating it from the displacement measurements, as done in [13] by a coplanar laser indicator. Another way is to eliminate the UAV translations and rotations through a Normalized Cross Correlation approach [14].

Techniques for monitoring bridge displacements using computer vision can be grouped into two categories: Target-based and Targetless approaches. In the following subsections (2.1, 2.3), we introduce some advancements in prior work in each category and their limitations. In subsection 2.2, we review the state-of-the-art fiducial marker systems to evaluate their potential use as trackable targets.

2.1 Target-based Vision Frameworks

Target-based frameworks mainly use traditional feature extraction algorithms to extract features and then are combined with template-matching algorithms or classifiers for target recognition [23]. They often rely on placing a trackable target on the targeted plane, such as printed markers or LED targets [18] [19].

These targets are then detected and tracked for estimating vertical displacements. Other target frameworks are to track targets within pre-recorded videos and extract displacements. [19] analyzes each video frame for detecting the target using Digital Image Correlation. The displacement is then measured in terms of pixel relations. Finally, the algorithm calculates this distance corresponding to real-world space and is validated using a magnetostrictive displacement sensor as ground truth [24]. Similarly, [25] uses attached markers for monitoring bridge displacements with a stationary camera. Using multiple targets was considered to characterize bridge displacements using an algorithm that could predict positions within a millimeter range [19]. This paper achieves a displacement monitoring accuracy of 3 mm error at a 10 m monitoring distance. Such error ranges cannot be tolerated when using form bridge displacements with 2 mm. Moreover, this is achieved in stable, controlled room conditions with a fixed camera and at large displacements ($\pi/8$). In another approach, UAVs were deployed to extract displacements of a multistory laboratory-scale structure [21]. The proposed algorithm applied a high-pass filter to remove the effect of the UAV's rigid body motion. However, this approach requires a strong assumption about the appropriate cutoff frequencies.

Despite promising results, these approaches often require high computational resources, leading to a lack of real-time response. Moreover, most of the techniques utilized stationary cameras for tracking targets. Those who leveraged UAVs have implemented approaches focusing on eliminating drone noise while ignoring putting more effort into advancing the robustness of detecting the targets. None of the above approaches have utilized state-of-the-art fiducial markers (i.e., ARTag, AprilTag, ArUco, and STag), mainly used for localization and tracking in the robotics field [26]. Adopting such frameworks can help set a clear boundary of how accurately computer vision can detect millimeter displacements. In the

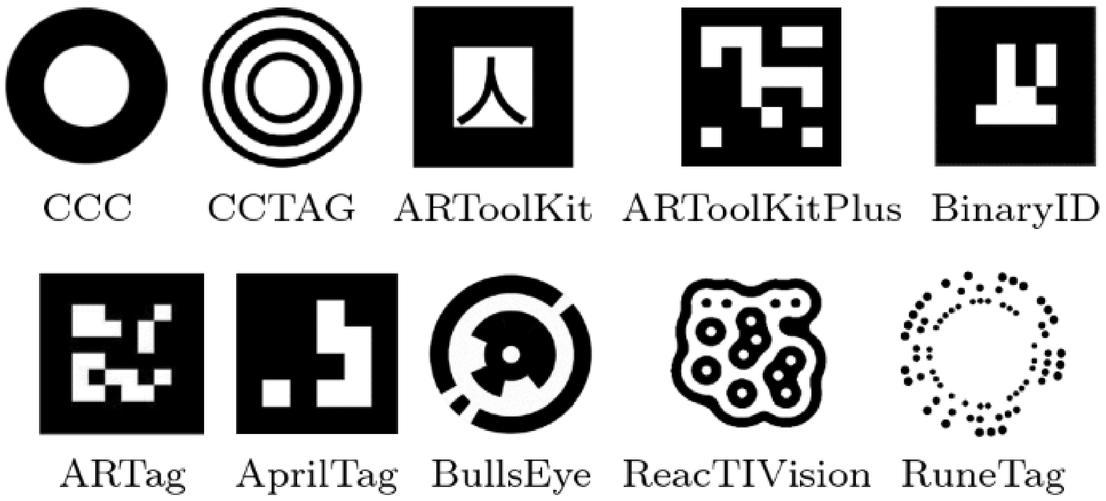


Figure 2.1: Examples of the different patterns of Fiducial Marker Systems [1]

following subsection 2.2, we present prior work done on fiducial markers.

2.2 Fiducial Marker Systems

Fiducial markers are patterned objects printed on paper and placed on the targeted area in the camera's field of view. These markers are mainly utilized for pose estimation, tracking, localization, providing a reference point, or measurement in the scene. In some cases, markers can carry simple messages through their encoding to provide meaning when detected. These patterns are designed to stand out in the environment for easier detection. While these patterns can be in circles, lines, squares, dots, etc., the three major pattern categories are circular [27] [28] [29], square [30] [31] [32], and topological [33] [34], as shown in Figure 2.1. The fiducial marker system comprises a set of valid markers and an algorithm that detects and possibly corrects images [35]. As the applications of fiducial markers range across various disciplines, from medical imaging to PCB manufacturing [26], the current state-of-the-art systems are ARTag, AprilTag, ArUco, and STag [26].

ARTag [30] [36] was the first to introduce a built-in forward error correction for its encoding system. Additionally, it uses a 2D barcode within a six-by-six grid of cells in the interior of the marker to make encoding easier. This made the detection much faster than prior marker systems. In addition, as their detection algorithm was based on the image gradient, it is robust to changes in lighting and occlusion. However, its detection mechanism was outperformed by AprilTags [31] in later studies.

AprilTag [31] fiducial system has undergone two phases of development: the initial phase of introducing the system [31] and advancing the system for robust and efficient detection [37]. It is the first system to introduce a lexicode-based tag generation method to enhance the detection algorithm and reduce false positive detections [38]. AprilTag is primarily used in robotics applications that involve tracking cars by UAVS [39], robotic arms [40], and system calibrations [41]. As AprilTags has been one of the most popular systems, it still has some limitations in computing time and is prone to errors with the angular rotation of cameras. Thus, for tracking and detecting tags in a reduced computing time, the ArUco markers system is introduced [35].

The ArUco system has been widely used in robotics applications for its advantage of reduced computing time, real-time detection approach, and accurate detection and tracking [42]. [43] utilizes ArUco tags to position UAVs during aerial inspections accurately. This work has shown promise in precise positioning using ArUco markers. In addition, the ArUco markers system has outperformed the AprilTags system in accurate UAV landing due to its higher detection framerate, reduced computation, and robust detection in complex environments [44]. In addition, It adapts to non-uniform illumination by utilizing Mixed Integer Linear Programming [45]. One of the main advantages of the ArUco framework is its ability to be tweaked based on the application. For instance, while STag outperforms all systems in robustness to occlusion [46], ArUcoTag has the option to enable robustness to occlusion. These options allow the user to use the same system of ArUco as needed for the application. Robustness to occlusion has the disadvantage of slow processing time and low detection frame rate (10 fps) than required for detecting bridge displacements [47]. Lastly, the open-source code available for ArUco allows for real-time tracking or video tracking.

Based on the previous work, utilizing state-of-the-art fiducial markers as targets for estimating displacements of bridges might outperform existing techniques in detecting such small displacements. This will give us a clear vision of what to expect when implementing a targetless vision framework for detecting displacements. In the following subsection, we will talk about the current targetless vision frameworks.

2.3 Target-less Vision Frameworks

Targetless frameworks essentially focus on utilizing imagery information to spot the differences between pixels and employ that to measure the displacements. For instance, [48] [49] [50] introduce targetless optical flow-based approaches. As these approaches solely rely on the image intensity information, the results are sensitive to different lighting conditions, changes in perspective, and surface conditions. Moreover, [51] uses complex-steerable pyramids to extract phase and amplitude information for each video frame as a phased-based approach. Using phased-based motion estimation and video magnification algorithms. The work in [51] [52] successfully extracted the full-field operational deflected shapes and natural frequencies of different structures (i.e., cantilever beams and wind turbine blades). However, this was done using a stationary camera platform in a controlled, stable environment. Other approaches exploit the structure's natural targets in imageries within each frame. For instance, [53] has employed an object-tracking technique based on the Kanade–Lucas–Tomasi (KLT) tracker to detect the structure's points of interest. This was tested on a laboratory-scale six-story structure.

Several Frameworks, such as [13] [47] [54], utilize laser points to develop a contactless measurement technique. To start, [13] leverages deep learning methods to track bridge features from image sequences collected by multiple UAVs. Laser indicators are utilized to eliminate the motion of UAVs. However, this approach's limitation is the algorithm's performance not meeting the real-time requirement. On the other hand, the reliance on a stationary camera in the approach of [47] makes it challenging to utilize for remote bridges and highly complex structures. In addition, [10] introduces a novel vision-based displacement measurement approach, using only one UAV and a motionless laser spot projected from a distance away as a reference. However, the proposed method depends on tracking a predesigned marker of known size installed on the structure and assumes appropriate frequency.

Reviewing the recent related work on Targetless displacement estimation, There needs to be a robust, ready-to-use, completely targetless UAV framework to extract such small displacements of bridge deformations. Thus, it is essential to develop a target-based framework for UAV capturing and follow it with a targetless one.

Chapter 3

Pipeline Development

This chapter discusses and analyzes the different pipeline components used for measuring displacements. With slight changes in each component depending on the experiment, this pipeline is composed of six main components: capturing videos of the area of interest (3.1), analyzing the captured videos (3.2), getting pixel displacement values (3.2), converting from pixels to inches (3.3), adjusting the results (3.4), and ground truth comparison (3.5). Figure 3.1 shows the different parts of the pipeline to get the inch-displacements data and compare it with ground truth. The following subsections will address each component and discuss the tools used across all experimentations.

3.1 Capturing Videos

During each study, the first step is to capture the displacements of the tag. These tools have been mainly used to capture the displacements of the tag placed on the area of interest. For the Static and Dynamic experimentations, we used an audio signal to start the ground truth capturing sensor, the Canon camcorder, and the Blackfly camera at once. This is to ensure that all platforms are collecting data in sync.

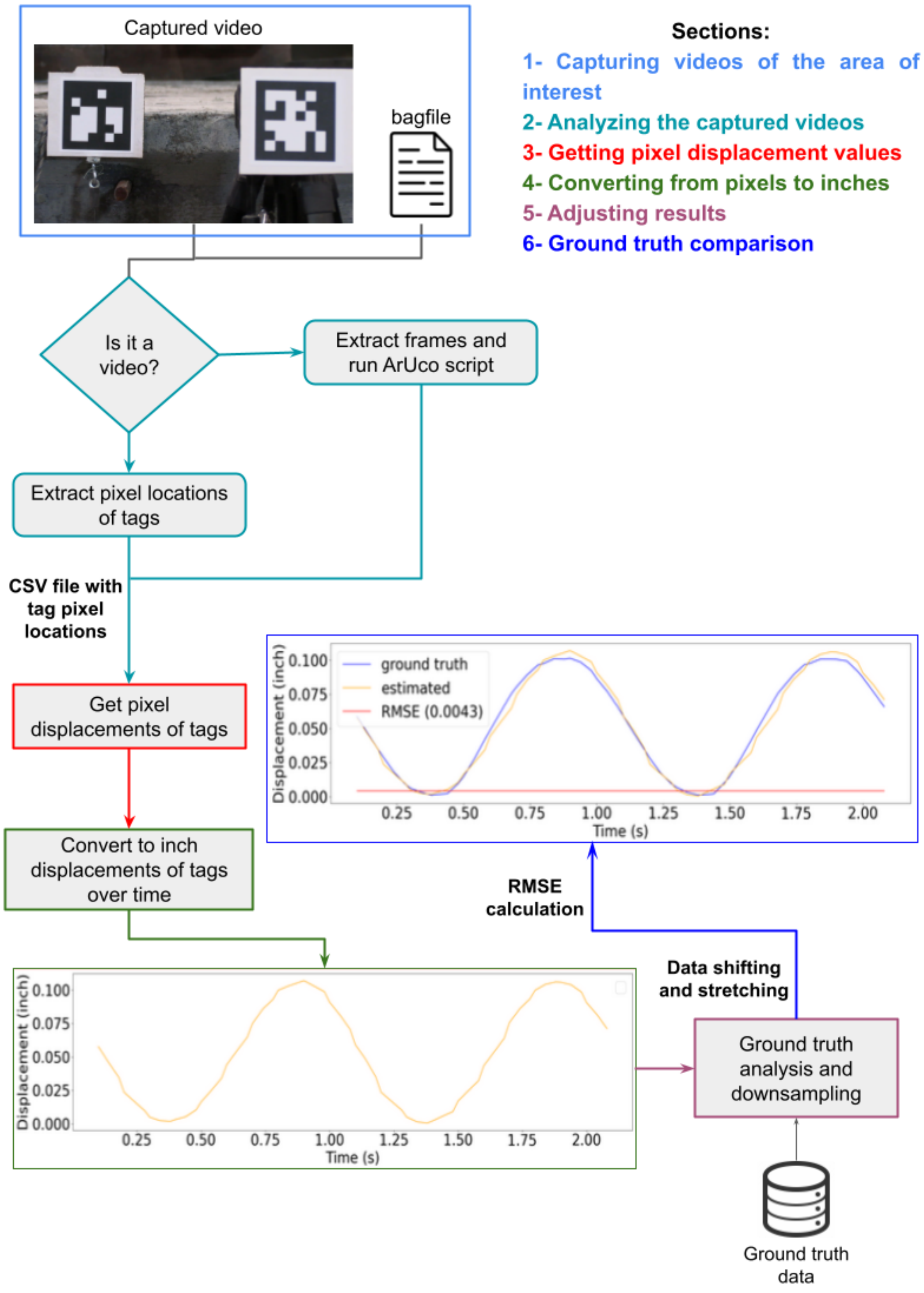

Captured video
bagfile
Sections:
1- Capturing videos of the area of interest
2- Analyzing the captured videos
3- Getting pixel displacement values
4- Converting from pixels to inches
5- Adjusting results
6- Ground truth comparison
Is it a video?
Extract frames and run ArUco script
Extract pixel locations of tags
CSV file with tag pixel locations
Get pixel displacements of tags
Convert to inch displacements of tags over time
ground truth
estimated
RMSE (0.0043)
Displacement (inch)
Time (s)
RMSE calculation
Data shifting and stretching
Ground truth analysis and downsampling
Ground truth data

Figure 3.1: The general flow diagram of the proposed pipeline

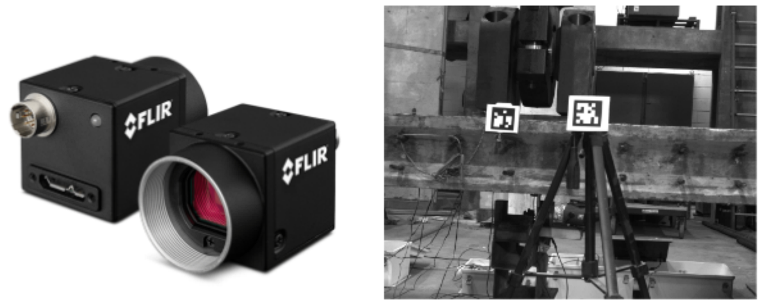


Figure 3.2: **(*Left*)** Blackfly S USB3 Camera Sensor **(*Right*)** An example of picture taken by the Blackfly camera

Although most of the camera platforms were used for video capturing, the Blackfly S camera captured the tag in real-time and only recorded the pixel locations. This is because there was a huge lag when trying to record the video as a whole. This lag caused the detection frame rate to decrease significantly. Thus, we only had rosbag recordings with pixel locations of the markers in the field of view with their IDs. In doing this, we study the effect of real-time detection versus video recordings. After capturing data, we move to the pipeline's second component, analyzing the captured videos.

3.1.1 Tools

Blackfly S USB3 Camera Sensor This Monochrome camera sensor, Figure 3.2, was manufactured by Teledyne FLIR with a 5.0 MP (resolution of 2448 px × 2048 px) equipped with a Sony IMX250 sensor. This camera was integrated with ROS to operate a real-time ArUco detector and record tag displacements using a rosbag package. This camera was leveraged in the Static and Dynamic experiments to capture and record real-time displacement.

Canon Camcorder This Canon VIXIA HF G50 4K30P Camcorder has the capability to capture videos at 4K UHD 30 fps. It also includes a 20x Optical Zoom

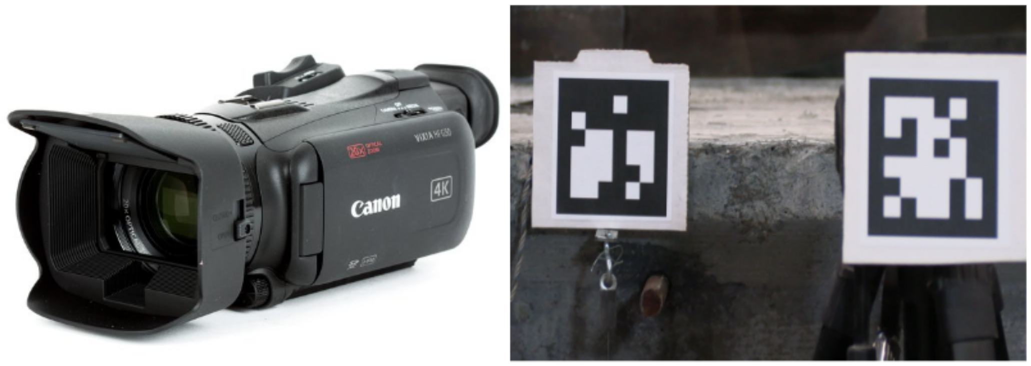


Figure 3.3: **(*Left*)** Canon VIXIA HF G50 4K30P Camcorder **(*Right*)** An example of picture taken by the Canon camera

Lens. This Camcorder has been used to capture tag displacements in Static and Dynamic experiments. It has been used, along with the Blackfly S camera, to study the effect of zoom level and camera resolution on detecting the tag. Figure 3.3 shows a replica of the used camera as well as a frame of the video recorded.

DJI Mavic Pro and DJI Mavic Air 2 Drones Both camera-equipped drones are manufactured by DJI and operated by the DJI Fly and DJI Go mobile applications. The Mavic Air 2 is a more modern drone than the Mavic Pro. The Mavic Air 2 has the capability of capturing videos with 4K resolution at 30 fps with 2x zoom. In addition, the camera's gimbal has a wider angle for mechanical range with a focussing option on the targeted area. While the Mavic pro is a less capable drone with no zoom and lower gimbal robustness, both drones were used to study the effect of the drone's specifications on estimating the tag displacements. Afterward, the more robust drone is utilized for experimentation on displacements at different distances and zoom levels. Figure 3.4 shows pictures of both drones used through this research.

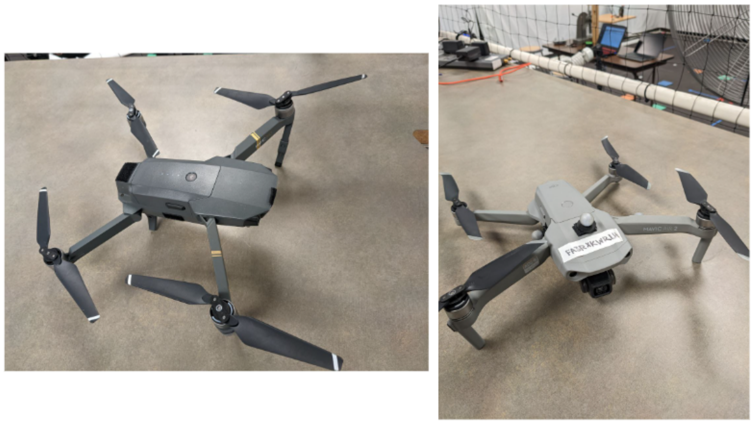


Figure 3.4: ***(Left)*** DJI MAVIC Pro ***(Right)*** DJI MAVIC AIR2

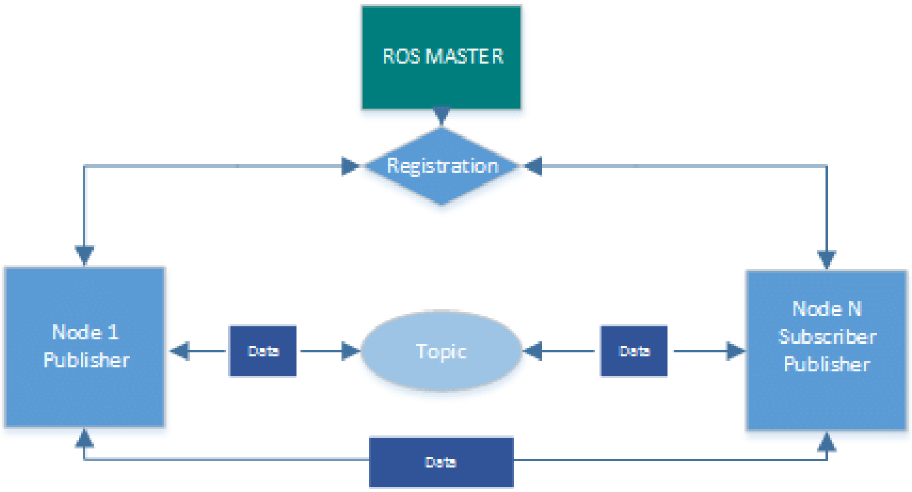


Figure 3.5: ROS communication flow diagram [2]

Robot Operating System (ROS) ROS is an open-source robot operating system that provides a structured communications layer above the host operating system [55]. ROS is widely used across the robotics community due to its convenience and ability to handle several common problems across robotics software development. Figure 3.5 demonstrates the standard communication method in ROS. Our pipeline integrates ROS with the Blackfly S Camera Sensor to capture real-time displacements of tags and record the pixel locations using the ROSBAG package.

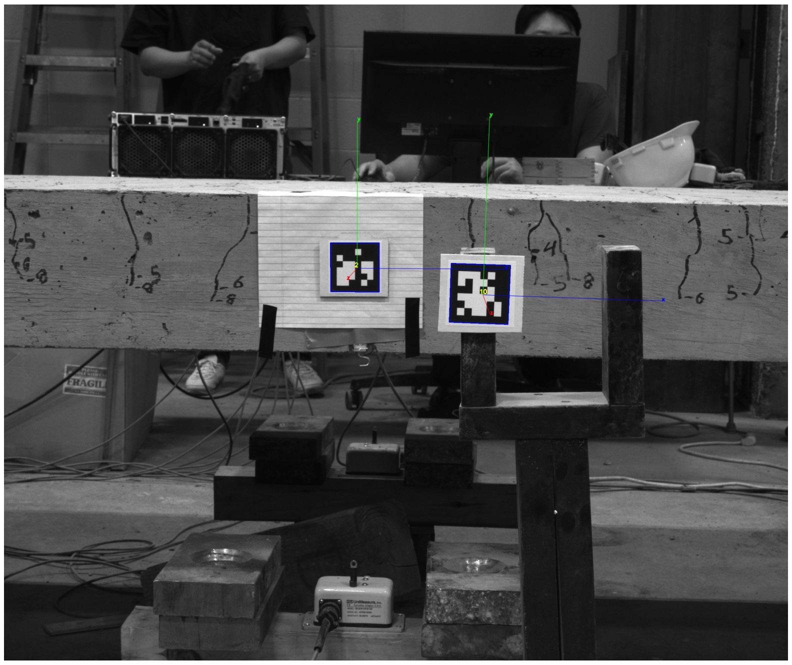


Figure 3.6: A screenshot of real-time ArUco detection using Aruco Real-Time Detector

ArUco Real-Time Detector This software package was developed, by PAL Robotics, as a ROS wrapper based on the open-source ArUco detection code. As this package is compatible with ROS, we use it to integrate with the Blackfly S camera ROS driver to capture ArUco tags in real-time. This package has the advantage of high framerate tracking of the markers ($\approx$ 26 fps). Additionally, it includes enhanced precision tracking with a given tag size and is optimized for minimal perspective ambiguity. Figure 3.6 shows a screenshot of the static test of a real-time tracked tag.

ArUco Tags The ArUco Tags are the targeted markers placed in the camera's field of view to be tracked. Across all studies, we primarily use two tags a stationary tag and a moving tag. The moving tag is placed on the area of interest to capture the tag's displacements, hence, the area's displacement. The stationary tag is used

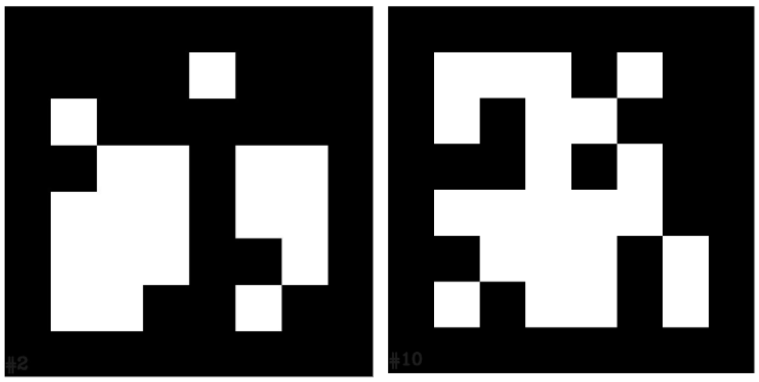


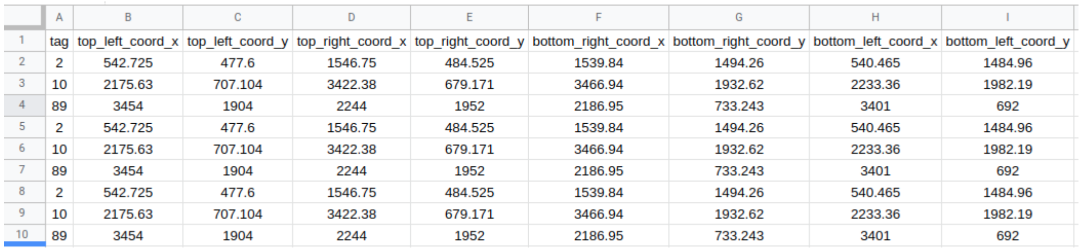
Figure 3.7: ***(Left)*** ArUco Tag with ID 2 ***(Right)*** ArUco Tag with ID 10

as a reference to capture the movements of the camera platforms. If the camera is not moving, it can be used to measure the error distribution of the ArUco detector. To differentiate between them, we use two different encoded ArUco tags. The moving ArUco tag has an ID of 2, while the stationary has an ID of 10. These ids can be identified by the ArUco detector. Figure 3.7 shows two examples of the ArUco tags used in these studies.

3.2 Videos Analysis and Obtaining Pixel Displacements

For analyzing the captured videos by the Canon camera and the UAVs at 29.97 fps, we do several necessary steps to get the pixel displacements of the tags:

1. Marking the starting time of the test by listening to the audio signal in the video. As there is no audio in the videos captured by the UAVs, hand gestures are used in the video to mark the starting point of the experiment.
2. Utilizing FFmpeg software to crop the video from the test starting point to the end. Additionally, FFmpeg extracts 30 frames per second from the trimmed video.



	A	B	C	D	E	F	G	H	I
1	tag	top_left_coord_x	top_left_coord_y	top_right_coord_x	top_right_coord_y	bottom_right_coord_x	bottom_right_coord_y	bottom_left_coord_x	bottom_left_coord_y
2	2	542.725	477.6	1546.75	484.525	1539.84	1494.26	540.465	1484.96
3	10	2175.63	707.104	3422.38	679.171	3466.94	1932.62	2233.36	1982.19
4	89	3454	1904	2244	1952	2186.95	733.243	3401	692
5	2	542.725	477.6	1546.75	484.525	1539.84	1494.26	540.465	1484.96
6	10	2175.63	707.104	3422.38	679.171	3466.94	1932.62	2233.36	1982.19
7	89	3454	1904	2244	1952	2186.95	733.243	3401	692
8	2	542.725	477.6	1546.75	484.525	1539.84	1494.26	540.465	1484.96
9	10	2175.63	707.104	3422.38	679.171	3466.94	1932.62	2233.36	1982.19
10	89	3454	1904	2244	1952	2186.95	733.243	3401	692

Figure 3.8: Screenshot of the ArUco Detector output after cleaning

3. A bash script of the ArUco detector runs on the extracted frames to record the pixel locations of the tags in the fields with their ids and save them in a CSV file.

4. CSV file is imported into a python script to analyze and get the pixel displacements.

As shown in Figure 3.8, the ArUco detector tracks the pixel locations of all four corners of the tag. As the tag displacements are vertical, we choose the y coordinate of the top left point of the tag to get the pixel displacements. Lastly, we loop through pixel locations of that y coordinate and subtract each pixel location, in the y direction, from the maximum. This gives us the pixel differences between each y coordinate of the top left point across all the frames. In other words, now we get the vertical displacements of the tag across a specific video.

For the recorded rosbag files, it takes fewer steps to get the pixel locations as the ArUco ROS package records them:

1. Importing the files into the python script to read and decode using the bagpy python package. This python package outputs a CSV file with the timestamps of the tags ids and the pixel locations of each edge point of the tag.

2. Marking the starting point of the experiment and crop the data captured for experimentation.

3. Looping through these pixel locations in the same fashion as mentioned before to get the pixel locations.

3.2.1 Tools

FFmpeg FFmpeg [56] is a free and open-source software for handling videos, audio, and other multimedia files. For our project, we use the command-line FFmpeg tool to crop our videos and extract them into frames. For cropping, we use a command like:

$$ffmpeg\ -i\ filename\ -ss\ starttime\ -to\ endtime\ -c : v\ copy\ -c : a\ copy\ outputname$$

to crop a video and export the trimmed video into a separate file. To extract frames off of a video at 30 fps, we use this command:

$$ffmpeg\ \ -i\ \ filename\ \ -r\ 30\ test_{\%}02d.png$$

ArUco Detector Bash Script We developed this bash script based on the ArUco open-source project [42] to loop through frames of videos, detect the tags in the frame, and save the pixel locations in a CSV file with the tag id. The bash script helped detect the markers in the frames of the videos, see B.1.

bagpy package This package was developed by The Compositional Systems Lab (CSL) at Vanderbilt University to facilitate the reading of rosbag files based on semantic datatypes. This package helps differentiate between the different ROS topics and the associated messages.

3.3 From Pixels to Inches

After fetching the pixel displacements of the tags, we need to acquire the displacements in inches to compare them with the ground truth values. As the size of each side of the square tag in the video is known, a simple pixel-to-inch ratio is calculated to get the inch displacements. As each row in the CSV file represents a whole frame, we loop through the rows of the CSV file with the pixel locations to get the euclidean distance of the left side of the tag in pixels. As we have the tag left side size in inches, we can get the pixel-to-inch ratio by dividing it by the know tag size. We use the pixel-to-inch ratio in each frame and multiply that by our pixel displacements. Finally, we have the inch displacements of the tag across all frames.

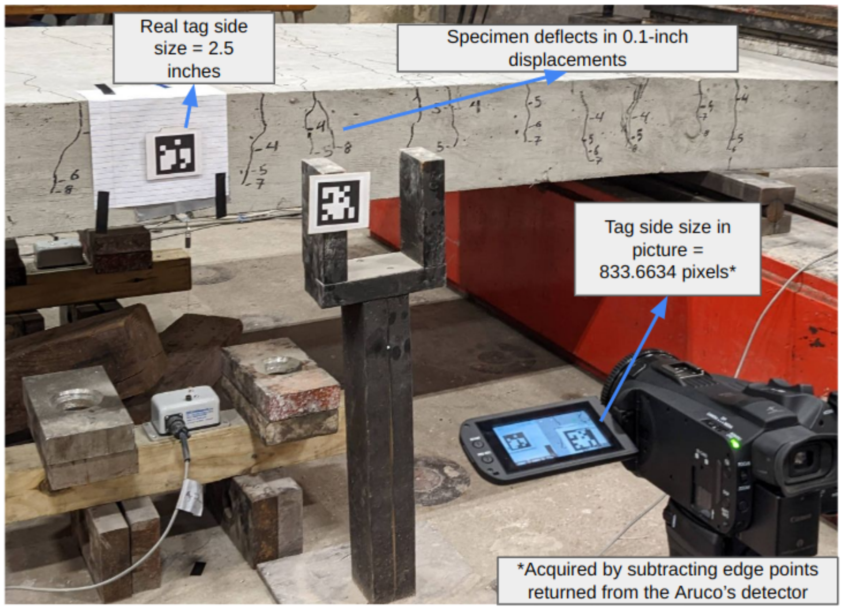

Real tag side size = 2.5 inches
Specimen deflects in 0.1-inch displacements
Tag side size in picture = 833.6634 pixels*
*Acquired by subtracting edge points returned from the Aruco's detector

Figure 3.9: Tag size in inches and pixels in one frame

To explain how this is done for just one frame, Figure 3.9 shows the size of the tag in inches and pixels for one frame of the camera. We get the tag size in pixels by subtracting the pixel coordinate points of the tag returned by the ArUco detector. Afterwards, we get the displacement of the deflection of the specimen in pixels, as shown below:

$$\left[\frac{Tag\ side\ size\ (inches)}{Displacement(inches)}\right] = \left[\frac{Tag\ side\ size\ (pixels)}{Displacement\ (pixels)}\right]$$

$$\left[\frac{2.5_{inches}}{0.1_{inches}}\right] = \left[\frac{833.6634_{pixels}}{Displacement_{pixels}}\right] \tag{3.1}$$

$$\text{Displacement (pixels)} = \left[\frac{833.6634_{pixels} * 0.1_{inches}}{2.5_{inches}}\right] = 33.3465\ pixels$$

$$0.1 \text{ inches } \mathrel{\widehat{=}} 33.3465\ pixels$$

This step is vital as it delivers the pixel resolution required to capture the targeted displacement at the same accuracy level. For instance, one of the tests gives an accuracy of 90% with a pixel resolution of 6 pixels per 0.1 inch using a stationary camera. This means that to detect the motion of x inch, we need a pixel resolution of 6 pixels per x inch to achieve the same accuracy.

3.4 Adjusting Results

So far, the inch displacements for the tags across the captured video are stored in a CSV file. No further analysis or adjustments must be performed on the data captured by the stationary cameras. However, the UAV motion needs to be removed for the data captured by UAVs to get the truly estimated displacements by the ArUco detector. We do that in two approaches. The first way is by subtracting the drone's motion from the moving tag's displacements. The drone motions are captured and reported by the Vicon system. As the drone was moving, the movements of the drone were reflected on the stationary tag. Thus, the second way is subtracting the stationary tag displacements from the moving tag displacements. In later sections, we present and discuss the results of each approach.

3.5 Ground Truth Comparison

For pipeline evaluation, the estimated inch displacements are compared with the ground truth captured for each experiment. Although the approach for capturing ground truth differs per experiment, ground truth data is always captured at a higher frequency than the results. Thus, we make use of a downsampling method to lower the sampling rate to match the captured data's rate. Leveraging the python-pandas-resampling approach reduces the frequency rate of the ground truth to the desired rate to fit the testing data. This method is utilized for frequency conversion and resampling of time series data. In addition, this method applies a lowpass filter to the input sequence to prevent aliasing during resampling. The same downsampling technique is utilized to downsample the UAV's motion before subtracting it from the moving tag displacements. Due to timing errors, further adjustments such as shifting or stretching are performed on the data to achieve better results. After aligning the data, we get the accuracy of the estimated displacements with the ground truth by calculating the data's root mean square error (RMSE), 3.2. RMSE was utilized as the metric reference for this research and prior work as we are interested to see the frequency distribution variance of error magnitudes. Later, we discuss the different ways of obtaining the ground truth in each experiment.

$$RMSE = \sqrt{\Sigma\left(\frac{y_{est} - y_{ref}}{N}\right)^2} \tag{3.2}$$

Chapter 4

Stationary Testing

This chapter answers Research Questions 2 and 3 in section 1.2: "**While fiducial marker systems have proven successful in tracking and localization of several robotics applications, how accurate are they in tracking such minimal displacements?**" and "**What pixel-to-inch ratio is required to accurately detect tags with small displacements? Does the camera platform's resolution, zoom level, distance from the target, and motion on the platform affect the robustness of the estimation?**" by experimenting with the ArUco pipeline with stationary cameras. Additionally, this experiment includes different studies at different distances and zoom levels. The stationary experimentations in this chapter were conducted with markers with a size of 2.5 inches. It is called stationary experiments, as they were conducted with stationary camera platforms.

Before starting the stationary testing, a preliminary investigation was conducted to adjust the ArUco pipeline on the Blackfly camera with manual tag movements. The setup is shown in Figure 4.1. We utilized a Lathe machine to stick a marker on and moved it manually while capturing the displacements with the Blackfly camera. A lathe machine was helpful as it gave a way to manually adjust the tags with known displacements and see the reflection of these displacements in the ArUco pipeline. This was done to finalize camera lens settings or modify any

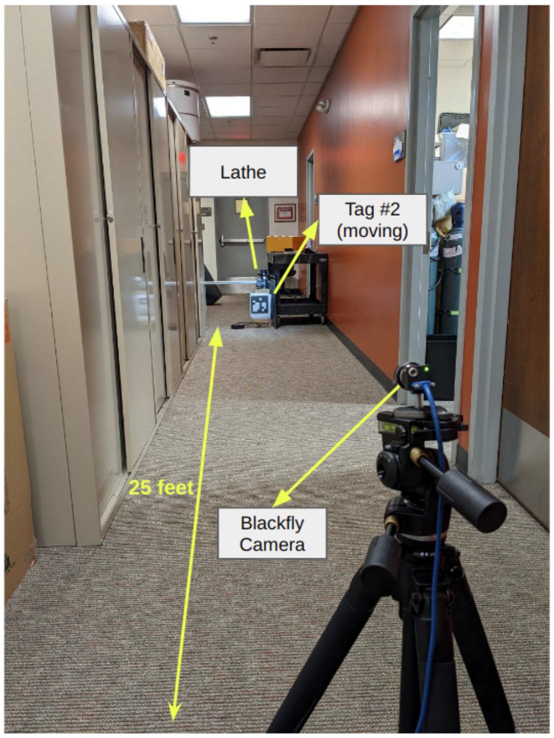


Figure 4.1: Trial experiments of the ArUco pipeline with manual tag movements

needed code in the ArUco ROS node by conducting any experiments.

This chapter is organized as follows: experimentation overview (4.1), experimental setup (4.2), Very Close Test (4.3), Close Test (4.4), Far Test (4.5), and overall conclusion (4.6).

4.1 Experimentation Overview

As the Stationary Experimentation's goal is to imitate actual bridge deformations and estimate them using the proposed pipeline. This experiment has three different tests. Although the experimental setup is the same across all tests, each test has a different distance between the target and the camera with a distinct zoom level. These variations generate various pixel-to-inch ratios with different accuracies. Each test is conducted with the Canon Camcorder and the Blackfly Camera. The

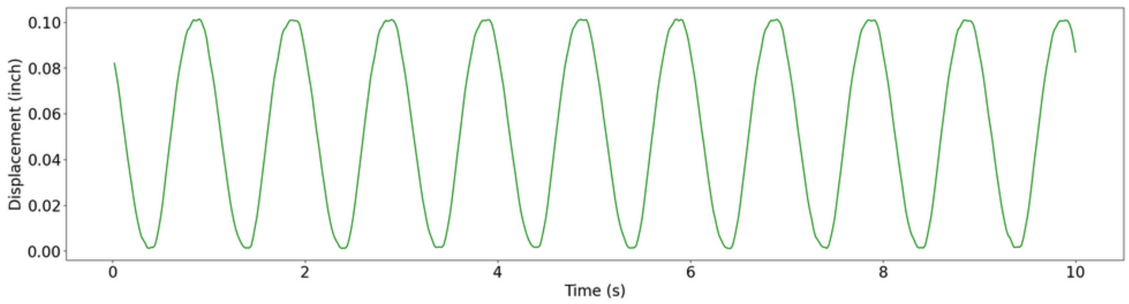


Figure 4.2: Potentiometer data (ground truth) reported for the specimen displacements

following subsection talks about the experimental setup. Figure A.1 shows a copy of the experimentation sheet used for the experimentation day.

4.2 Experimental Setup

To imitate the dynamic movements of a truckload passing over a bridge, we deploy a 220 kip actuator on a concrete rectangular specimen to perform a uniform stress cross-section. The thickness of the specimen was about 8 inches at a 38-inch height from the ground. The goal of the actuary is to exert a 12.00 kip force on the specimen at a frequency of 1 Hz, causing the specimen to deform in the shape of a sinusoidal wave. With a force of 12.00 kip exerted by the actuator on the specimen, the resulting displacements were within 0.1 inch; see Figure 4.2.

These studies aim to capture and report the sinusoidal displacements resulting from the actuator. For ground truth, a potentiometer sensor with a sample rate of 50 is connected to the middle point of the specimen with the moving target stuck on, as shown in Figure 4.3. In addition, a reference stationary tag is placed in the field view of the cameras. The cameras are both placed at the same distance from the target. However, since both cameras have different resolutions, they show different pixel-to-inch ratios. Three tests are conducted for this experimentation:

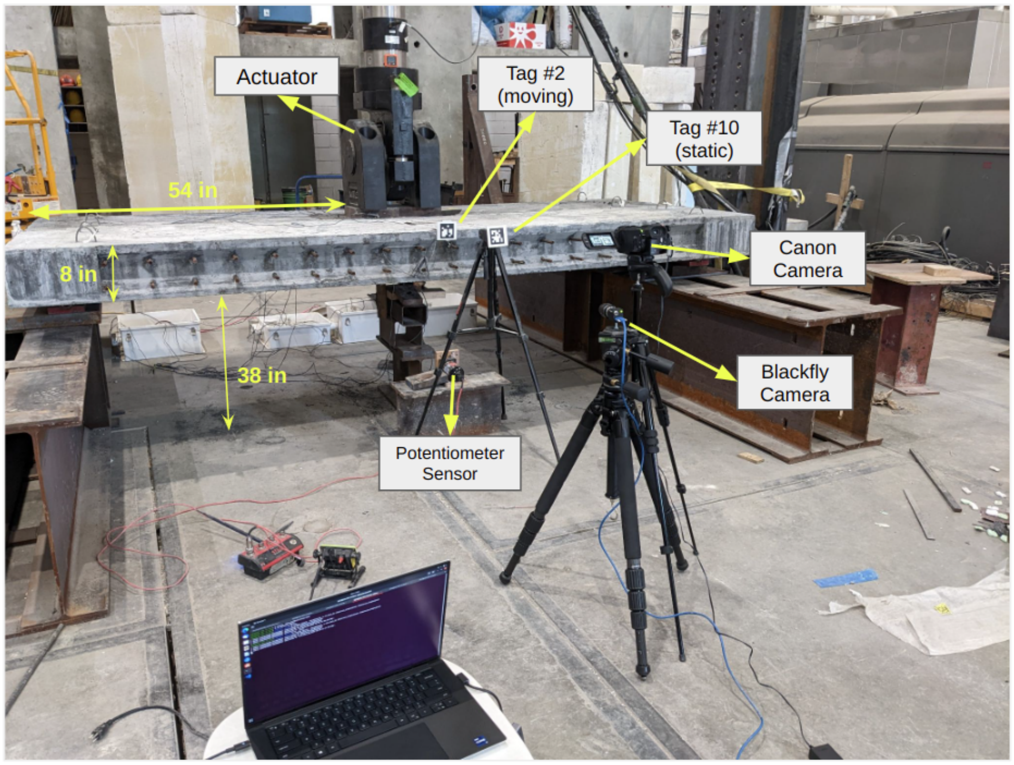


Figure 4.3: Experimentation setup for the Very Close Test

Very Close, Close, and Far. The Close and Very Close tests represent the targeted pixel-to-inch ratios from close distances. However, the Far Test was to investigate the minimal pixel resolution and report the RMSE. Figure 4.3 shows the setup of an experiment for the Very Close Test. The marker placed on the specimen is number 2, while the stationary marker is number 10.

4.3 Very Close Test

4.3.1 Overview

As shown in Figure 4.3, both cameras are placed **55 inches** from the target. However, the Canon Camera is zoomed in 10 times with no zoom on the Blackfly camera. Thus, there is more pixel resolution per 0.1 inch for the videos captured

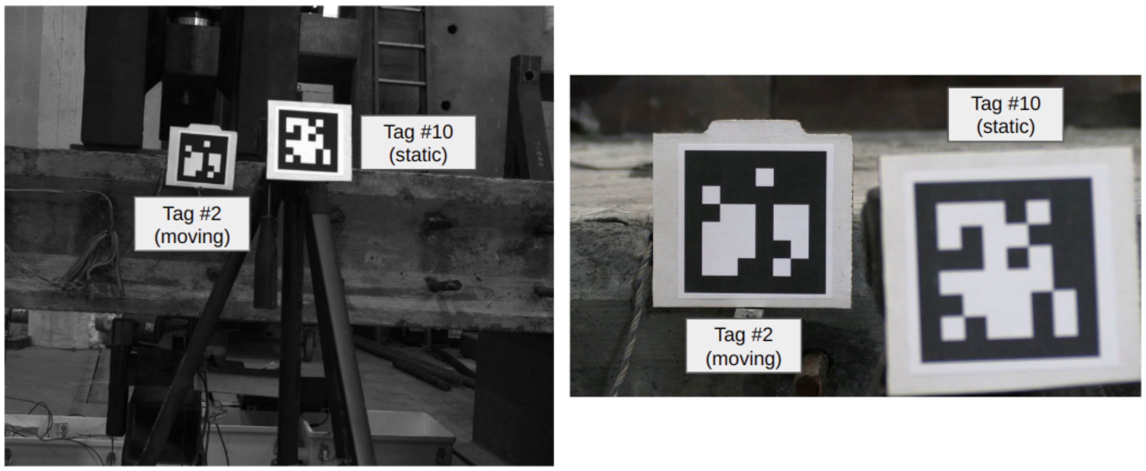


Figure 4.4: Very Close Test - Cameras field of view of tags **(*Left*)** Blackfly Camera **(*Right*)** Canon Camera

by Canon. The Canon Camera had a pixel resolution of **38.1640 pixels per 0.1 inch**, while the Blackfly had **8.5774 pixels per 0.1 inch**. Figure 4.4 shows tags in the field of view of both cameras. The testing duration was seven minutes of moving the specimen.

4.3.2 Results

Utilizing the pipeline mentioned in Chapter 3, Table 4.1 shows the results of running the Very Close Test of capturing 0.1 displacements using Blackfly and Canon Cameras. These results were reported after postprocessing and aligning the data. For comparison with the Canon Data, the ground truth data were downsampled from 50 Hz to 30 Hz to match the 30 fps of the camera. For comparison with the Blackfly data, the ground truth data were downsampled from 50 Hz to 26 Hz, as this is the ArUco real-time detection rate. The downsampling was performed using the steps shown in section 3.5.

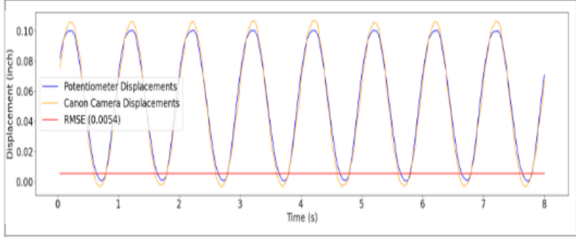
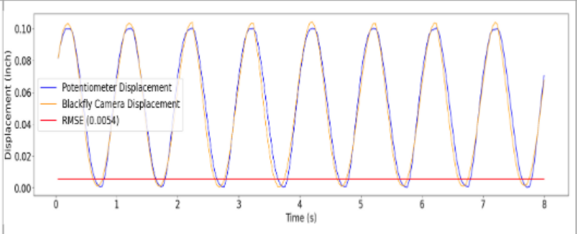
Very Close Test (55 inches away)	**Canon** Inch/pixel ratio — frequency	**Blackfly** Inch/pixel ratio — frequency
0.1 in = 38.1640 px — **30 hz** — **RMSE: 0.0054**		**0.1 in = 8.5774 px** — **25 hz** — **RMSE: 0.0054**
		

Table 4.1: Results of Blackfly and Canon Cameras for the Very Close Test - Tag 2 (moving) Displacements

4.3.3 Discussion

Analyzing the video captured by the Canon has shown blurry detections of tags at tracking times, which resulted from the camera auto-focus recalibration. This blurriness has affected detection accuracy, hence displacement estimation (see Figure 4.5). This blurriness did not happen for the ArUco online detection, as the Blackfly had manual focusing. This caused the RMSE for both cameras to be the same, even though one had more pixel resolution than the other. In an ideal case, Canon would be expected to show better accuracy. In addition, the lower pixel resolution for the Blackfly camera has compensated for the ArUco detection error.

To explain, we show the ArUco detection error in tracking the static tag 10 for both cameras. Looking at Table 4.2, we see that the error rate for the Canon was much lower (within 0.0650 inches). On the other hand, the detection error range for detecting the static with the Blackfly was within 0.0030 inches. This shows that having lower pixel resolution can sometimes be beneficial in compensating for the

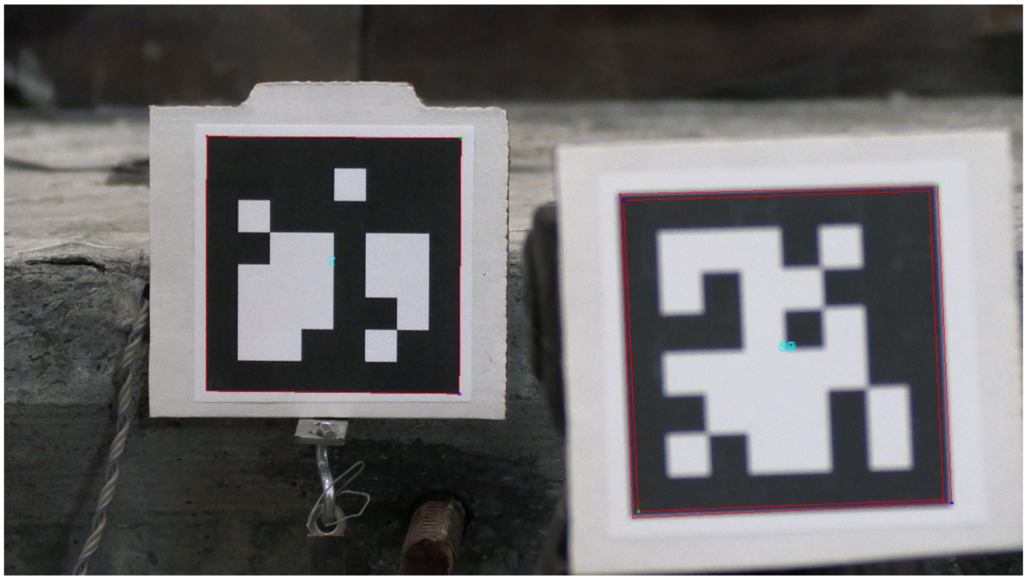


Figure 4.5: A frame of the Canon video showing that the ArUco detector is detecting two markers (10 and 89) on the left even though there is only one marker. This happens due to blurriness

ArUco detection error.

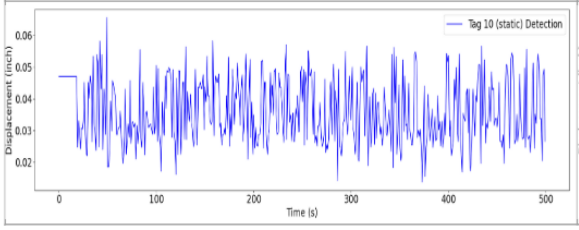
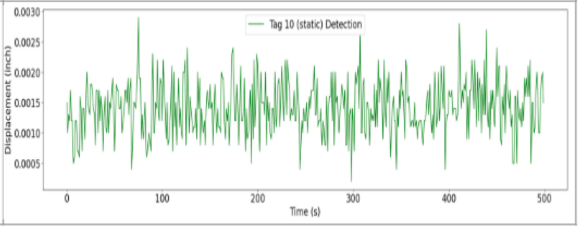
Very Close Test (55 inches away)	**Canon** Inch/pixel ratio — frequency	**Blackfly** Inch/pixel ratio — frequency
	0.1 in = 48.5341 px — **30 hz** — **Range: 0.0650**	**0.1 in = 10.5850 px** — **25 hz** — **Range: 0.0030**
		

Table 4.2: Results of Blackfly and Canon Cameras for the Very Close Test - Tag 10 (static) Displacements

Overall, this pixel-to-inch ratio has shown remarkable results of **0.0054** inches

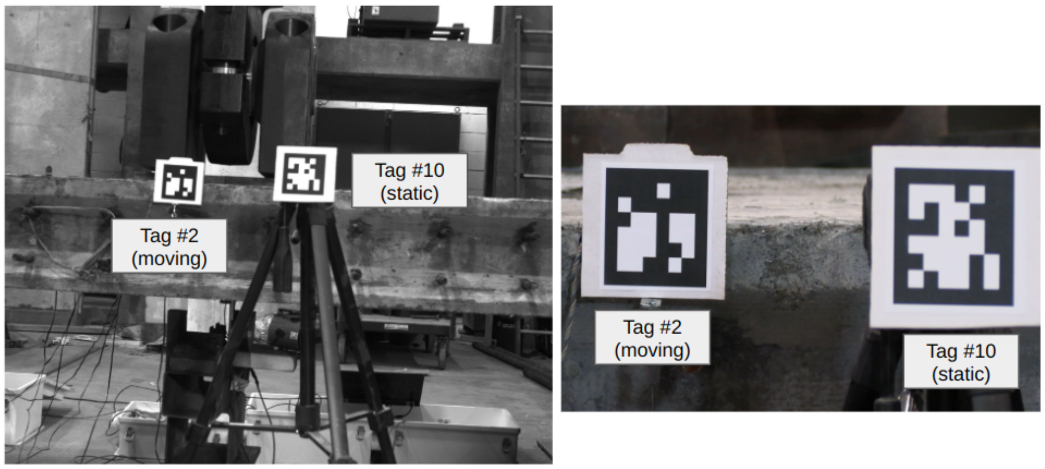


Figure 4.6: Close Test - Cameras field of view of tags **(*Left*)** Blackfly Camera **(*Right*)** Canon Camera

in RMSE for 7 minutes of tracking. This was about **94.6**% accuracy. For the next test, we decide to lower the pixel-to-inch ratio for two reasons: to further challenge the detector and to avoid over-fitting problems. Having a lower pixel-to-inch ratio can sometimes help compensate for the Aruco detection error.

4.4 Close Test

4.4.1 Overview

In this experiment, we adjust the camera's distance from the target and the zoom lens for the Canon to lower the pixel resolution. The camera's distance to the target was **70 inches** with a zoom of $13x$ on the Canon. These adjustments yielded a pixel resolution of **32.8298 pixels per 0.1 inch** for the Canon. On the other hand, the Blackfly had a pixel resolution of **6.0718 pixels per 0.1 inch**. Figure 4.6 shows both tags in the field view of the cameras. Lastly, we change the auto-focus of the Canon to manual focus to avoid auto-focusing in the middle of the video.

4.4.2 Results

The results for this test were conducted similarly to the Very Close Test with the same downsampling rates. Table 4.3 shows the Canon and Blackfly estimation results for capturing 0.1 inch displacements of the specimen.

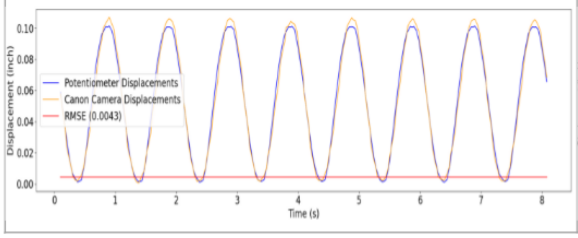
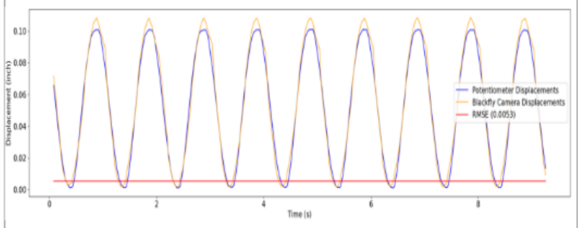
Close Test (70 inches away)	**Canon** Inch/pixel ratio — frequency	**Blackfly** Inch/pixel ratio — frequency
	0.1 in = 32.8298 px — **30 hz** — **RMSE: 0.0043**	**0.1 in = 6.0718 px** — **25 hz** — **RMSE: 0.0053**
		

Table 4.3: Results of Blackfly and Canon Cameras for the Close Test - Tag 2 (moving) Displacements

4.4.3 Discussion

Comparing the Canon results of this test to the previous one, we find that the RMSE has improved to be **0.0043** inches, even with reduced pixel resolution. This raises the pipeline's overall accuracy by 1.1% to be **95.7**%. This is because the blurriness of auto-focusing has been eliminated. However, we find that things did not change for the Blackfly data, as the change in pixel resolution was not significant (**94.7**%). **This test shows that the camera platform's stability significantly influences the overall accuracy of the system.**

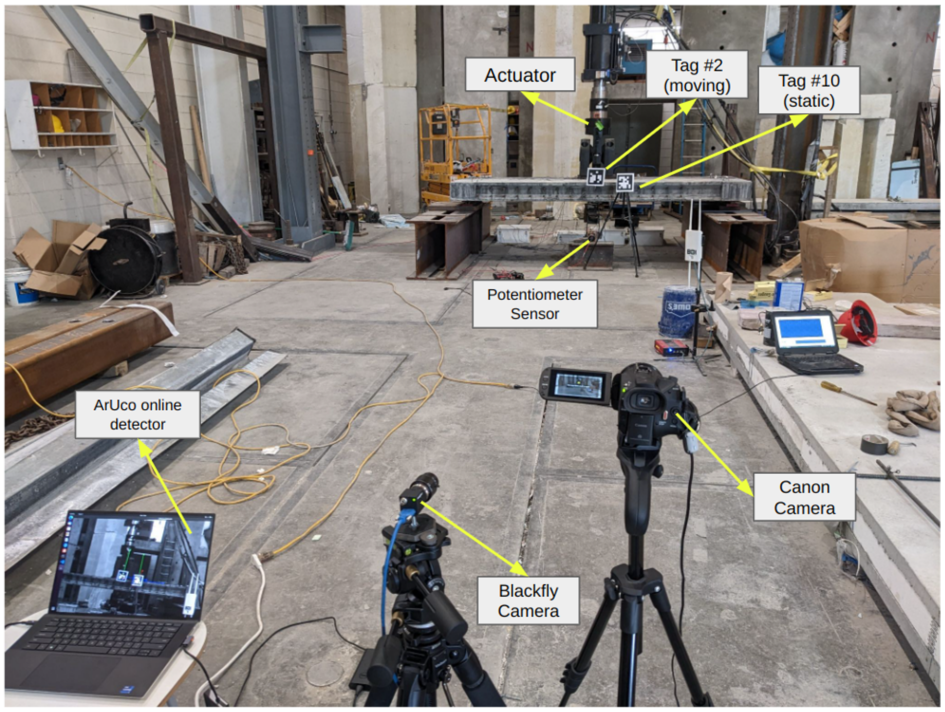


Figure 4.7: Experimentation setup for the Far Test

4.5 Far Test

4.5.1 Overview

This test was conducted to examine the pipeline with a significantly low pixel resolution of **1.0279 pixels per 0.1 inch** for the Canon. And **1.6090 pixels per 0.1 inch** for the Blackfly. These resolutions were achieved by placing both cameras **25 feet** from the target without zooming. To be able to detect tags at such a distance, both tags were swapped with larger tags with a size of 7.5 inches. Figure 4.7 shows the experimental setup for this test. Such scenarios could take place if the camera platform is farther away from the bridge than in previous tests.

4.5.2 Results and Discussion

The results in Table 4.4 are produced using the same steps as the previous two tests. As expected, the RMSE for the Canon and the Blackfly has increased to be **0.0708** and **0.0146**, respectively. This shows that accuracies for Canon and Blackfly are **29.2**% and **85.4**%, respectively. For this test, the Blackfly outperformed the Canon as it had a higher pixel resolution per 0.1 inch. This shows that pixel resolution is vital to achieving high accuracy for estimating displacements. At such long distances, It is also demonstrated that slightly higher pixel resolution, as for the Blackfly, can significantly enhance the accuracy of the estimation.

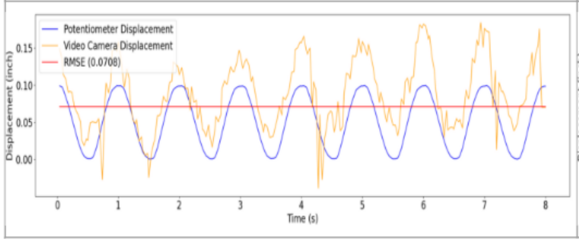
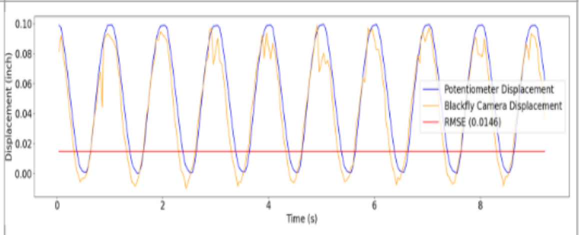
Far Test **(25 feet away)**	**Canon** Inch/pixel ratio — frequency	**Blackfly** Inch/pixel ratio — frequency
	0.1 in = 1.0279 px — **30 hz** — **RMSE: 0.0708**	**0.1 in = 1.6090 px** — **25 hz** — **RMSE: 0.0146**
		

Table 4.4: Results of Blackfly and Canon Cameras for the Far Test - Tag 2 (moving) Displacements

4.6 Overall Conclusion

This chapter shows how accurate the ArUco detector is in estimating bridge-like displacements. We deploy the pipeline to test displacements captured by stationary

	Canon Camcorder		Blackfly camera	
Test Type	Inch/pixel ratio (30 hz)	RMSE	Inch/pixel ratio (25 hz)	RMSE
Very Close Test 55 inches (10x zoom)	0.1 in = 38.1640 px	0.0054	0.1 in = 8.577 px	0.0054
Close Test 70 inches (13x zoom)	0.1 in = 32.8298 px	0.0043	0.1 in = 6.0718 px	0.0053
Far Test 25 feet (no zoom)	0.1 in = 1.0279 px	0.0708	0.1 in = 1.6090 px	0.0146

Table 4.5: RMSE values summary for all stationary tests

cameras. The Stationary Test was conducted to mock bridge deformation in the lab using an Actuator and concrete specimen. The ArUco detection has shown remarkable results through all the performed tests. However, the Close Test showed the best results for both cameras, with an accuracy of **95.7**% for the Canon and **94.7**% for the Blackfly. Eliminating blurriness by changing the Canon to manual focus has improved the estimation accuracy by **1.1**%, even with lower pixel resolution than the Very Close Test. As shown in Table 4.4, the Close Test had **32.8298 pixels per 0.1 inch** for the Canon and **6.0718 pixels per 0.1 inch** for the Blackfly.

Additionally. the Far test has shown the effect of pixel resolution, distance, and zoom level on the accuracy of estimating the structure's displacements. The Far Test had the lowest accuracy results dues to its low pixel resolution ($\approx$ 1 pixel) per 0.1 inches. However, the pipeline still showed excellent results with the Blackfly camera at such a distance from the target (25 feet).

Lastly, to answer the proposed research question 2, the ArUco tags has shown success in measuring small displacements, using a stationary camera platform, with an accuracy above **94**%. For research question 3, this study teaches us that we need at least **32.8298 pixels per 0.1 inch** to estimate small displacements with a targeted accuracy above **94**%. More pixel resolution per inch is required to achieve the best estimation accuracy under the condition of stable video conditions (no blurriness). This can be achieved by adjusting the distance to the target and zoom level.

Chapter 5

Drone Testing

This chapter answers Research Questions 1 and 4 in section 1.2: "**Can we develop a proof-of-concept pipeline that utilizes state-of-the-art computer vision techniques to track bridge displacements captured from camera-equipped drones?**" and "**How can the drone's motion noise be removed to reach the true estimated displacement of the tag?**" by deploying the proposed pipeline to analyze videos of displacements captured by a camera-equipped drone. Additionally, we propose two ways to eliminate the drone's noise from the estimated displacements. The first way is to utilize the reference stationary tag in the scene. The second way is by using the drone's positions of motion. Before deciding on the drone choice, we compare two drones with different capabilities to see the effect of the drone's resolution, zoom level, and stability.

For this chapter, we present two studies. The first study compares the DJI Mavic Pro and the DJI Mavic Air 2 (Figure 3.4) in detecting a stationary tag. Each drone has different capabilities regarding zoom level, resolution, focusing, and stabilization. After showing that the Mavic Air 2 is more robust. We deploy it in a second study to detect the various tag displacements at different distances and zoom levels. The experiments in this chapter are all conducted in a controlled lab environment.

This chapter is organized into two main sections: Drones Comparison (5.1), and Detecting Displacements using Mavic Air 2 (5.2). The later section is split into the following subsections: Experimental Setup (5.2.1), 0.5-inch Sinusoidal Testing (5.2.2), 1.0-inch Sinusoidal Testing (5.2.3), 2.0-inch Sinusoidal Testing (5.2.4), and Stepping Testing (5.2.5). Finally, we present the overall conclusion of these studies (5.2.6).

5.1 Drones Comparison

This study compares the DJI Mavic Pro and the DJI Mavic Air 2 drones in detecting a static tag placed in the camera's field of view. This study aims to show the effect of the stability of the camera-equipped drone on detecting the tag, hence estimating the displacement. The drone's stability will minimize the drone's noise incorporated in the tag displacement estimation. This can facilitate the approach of eliminating the drone's noise using the drone's motion.

Each drone has different capabilities in capturing and stability hardening. The DJI Mavic Air 2 has the option of doubling the zoom lens at 4K video capturing. On the other hand, the DJI Mavic Pro does not have a zoom option at such a resolution. As we learned the importance of increasing the pixel resolution per inch in the studies of Chapter 4, the zoom option in Air 2 will increase the accuracy of the system.

In addition, the DJI Mavic Air 2 has an additional option of stabilizing the camera's gimbal by focusing on a specific area to minimize. This aids the drone in hovering within the same space. This is not available in the Mavic Pro. Lastly, the DJI Mavic Pro had recent crashes that affected the drone's overall calibration, which could affect its stability. On the other hand, the Mavic Air 2 has never

Option	DJI Mavic Pro	DJI Mavic Air 2
$4K$ resolution	Available	Available
Zoom at $4K$ resolution	NA	Available ($2x$)
Gimbal stabilization	NA	Available
Platform stabilization	crashed multiple times	no crashes

Table 5.1: Options available in the DJI MAVIC Air 2 vs the DJI Mavic Pro

crashed. Table 5.1 compares the options offered by each drone.

Results This experiment has two studies. In the first study, we set the distance to the tag at **2 ft**. We place a static tag on the field of view of each drone. Then, we take two videos for each drone: one with a $2x$ zoom and another at $1x$ zoom. We repeat that in the second study but at a **6 ft** distance from the target. For this experiment, we set the altitude for the drone and leave the remote's throttle for the drone to hold its position. Both drones are at an altitude of **4 ft**. We capture a video of each study for 25 seconds and enter it through our pipeline. We then compare the range of the movements and the RMSE for each drone as shown in Table 5.2.

Analyzing Table 5.2, at the 2 ft distance, the Mavic Air 2 has shown better results at $1x$ zoom. In addition, it delivers significantly better results at $2x$ zoom, which is not available in the Mavic Pro. We also see that the range of motion of the drone decreases as the zoom level is lowered. However, both tests show a lower range of motion for the DJI Mavic Air 2 than for the Mavic Pro. This indicates that the Air 2 is more stable and has better accuracy than the Pro.

At the 6 ft distance, the Mavic Air 2 has shown slightly better results (10% less) than the Mavic Pro. In addition, the Mavic Air 2 appeared to be more stable than the Mavic Pro with a lower range of motion. Lastly, the Mavic Air 2 has the

	DJI Mavic Air 2	DJI Mavic Pro
Distance from tag (ft)	**2 ft**	**2 ft**
Resolution / Zoom Level	$4K$ / $2x$ zoom	-
Range of motion (inches)	2.853	-
RMSE (inches)	0.1611	-
Distance from tag (ft)	**2 ft**	**2 ft**
Resolution / Zoom Level	$4K$ / $1x$ zoom	$4K$ / $1x$ zoom
Range of motion (inches)	1.2824	3.0772
RMSE (inches)	0.8259	1.2877
Distance from tag (ft)	**6 ft**	**6 ft**
Resolution / Zoom Level	$4K$ / $2x$ zoom	-
Range of motion (inches)	2.0551	-
RMSE (inches)	1.3031	-
Distance from tag (ft)	**6 ft**	**6 ft**
Resolution / Zoom Level	$4K$ / $1x$ zoom	$4K$ / $1x$ zoom
Range of motion (inches)	2.032	2.5911
RMSE (inches)	1.4613	1.5762

Table 5.2: RMSE and Range of motion results for each drone on capturing a static tag

additional option to have better pixel resolution, which outputs a lower RMSE.

Conclusion Conducting this experiment, we conclude that the Mavic Air 2 has better stability in the air than Mavic Pro. This is shown through the lower range of motion it generates while hovering. This impacts the RMSE of the captured static tag to be lower for the videos captured by the Mavic Air 2 drone. In addition, The Mavic Air 2 has the advantage of achieving higher pixel resolution through its 2x zoom lens. While this is not available for the Mavic Pro, this has significantly improved the RMSE. Based on these conclusions, we deploy the Mavic Air 2 in the following experiment to capture displacements of a moving tag.

5.2 Detecting Displacements using Mavic Air 2

This experiment aims to evaluate the ArUco's pipeline on videos captured by a drone. This experiment deploys the Mavic Air 2 drone to detect an ArUco tag (target) moving at different displacements. In addition, the drone hovers at varying distances from the target. Moreover, this experiment compares two approaches to eliminate the noise in detection caused by the drone's motion. One approach uses a static reference tag placed in the camera's field of view to reflect the drone's movements. We then subtract the reflection of the noise from the target's displacements estimations. The other approach uses the raw drone's motion and removes it from the target's estimations. These two approaches are different as the first one includes the estimation error by the ArUco pipeline while the other doesn't. In addition, the raw drone's motion is relative to the space, while the static tag's pixel position is relative to the moving tag.

5.2.1 Experimental Setup

Unlike previous experiments in Chapter 4, the tag displacements for this experiment are all performed manually using an Arbor Press. As shown in Figure 5.1, the target tag (tag 2) is placed on an Arbor Press. Such a tool is easy to move in specific displacements. The static tag (tag 10) is placed close to the moving tag. The Air 2 drone hovers at different distances to capture the various displacements. We attain the ground truth movements for the Arbor Press and the drone positions using the Vicon motion-capturing system. We place Vicon markers on the drone, as shown in Figure 3.4 **(right)**, and on the Arbor Press, as shown in Figure 5.1. The Vicon cameras track those markers in the region. The Vicon system has millimeter accuracy for position estimation.

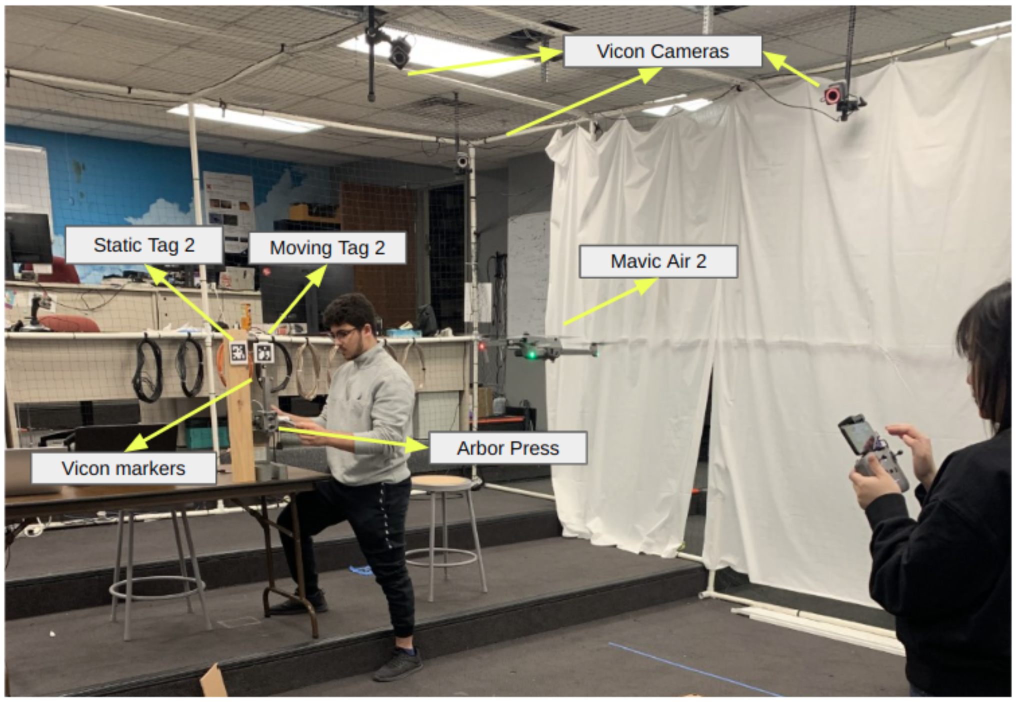


Figure 5.1: Experimentation setup for estimating tag displacement using drones videos

This experiment tests two main types of movements: sinusoidal and stepping. The sinusoidal test is conducted for three different displacements: 0.5-inch, 1.0-inch, and 2.0-inch. The drone captures each displacement at two different distances ranges. Each distance range has two tests, one with 2x zoom and another with 1x zoom. The stepping test has also been captured in the same fashion of distances and zoom levels. The stepping test goes from 0 to 4 inches in 0.5-inch increments. These variations of distances and zoom levels will cause different pixel resolutions per inch. We perform different displacements as this will indicate the accuracy of the ArUco pipeline in estimating different displacements from drone videos. In addition, this will provide help in determining the pixel-to-inch ratio required. For this experiment, all videos are captured at 4K resolution. In addition, we utilize the gimbal stabilization option by focusing the area of interest on the static tag to minimize the drone's noise.

The following tests are for estimating tag displacements in sinusoidal and stepping movements. For each test, we report and compare the RMSE of the displacement estimation of the pipeline with the two approaches of eliminating the drone's noise. We report the pixel-to-inch ratio for each distance range. There are mainly two distance ranges from the target to the drone: 1.8-2.5 feet and 5-6 feet. We capture the displacement for each range with 2x zoom and 1x zoom.

5.2.2 0.5-inch Sinusoidal Test

Analyzing Table 5.3, we see that the table compares the two previously mentioned approaches to eliminating the drone's motion noise from the estimations of the displacements. The left column shows the estimation after eliminating the drone's movements reported by Vicon. The right column shows the estimation after removing the static tag 10 estimations. We compare the results to the ground truth obtained by Vicon and report the RMSE.

We can see that the lowest RMSE reported is for test $a2$ using the static tag approach, which is $\mathbf{0.1903}$. This gives $\mathbf{61.94}\%$ as the best accuracy for this test. To achieve such accuracy, the drone was about 2 feet from the target resulting in a pixel resolution of **112 pixels per 0.5 inch**. This test reported better results than the $a1$ test, which has more pixel resolution. The $a1$ test has an accuracy of $\mathbf{58.52}\%$, which is 3.42% less.

The drone gets unstable as it gets closer to objects due to the obstacle avoidance feature in the drone. Interestingly, the worst overall RMSE was for the $a2$ test using the raw drone's movements approach. This is interesting as it shows how different the two methods of eliminating the drone's noise are. The $a2$ test had the worst RMSE using the drone's motion approach as the drone for this test was the least

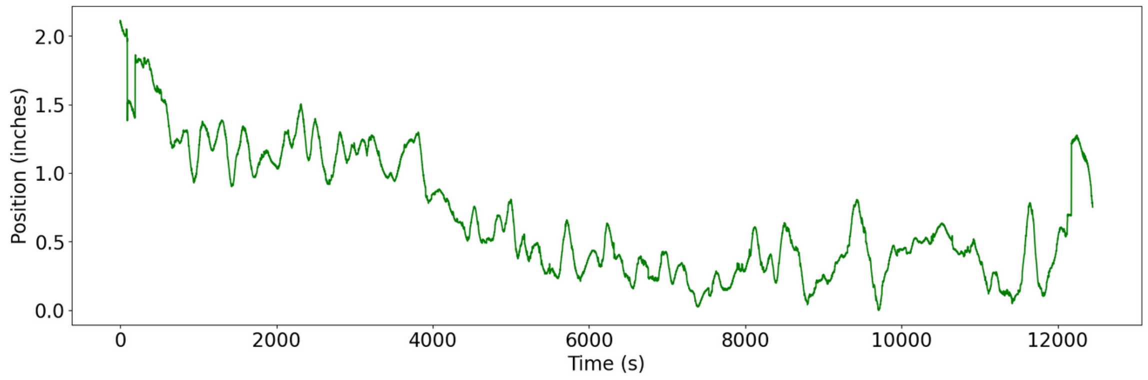


Figure 5.2: 0.5-inch Sinusoidal Test - Mavic Air 2 Inch Positions in Z direction

stable with a motion range of 2.2 inches, see Figure 5.2. The drone was most stable in the *b*2 test, which has the best RMSE using this approach with a range of 0.7 inches. This is because the drone was further from the target in *b*2 than in *a*2.

Throughout this test, the tag 10 approach always reports better RMSE values than the drone's motion approach. The drone's motion did not report accurate results for several reasons:

1. The drone's motion is different from the camera's motion due to the stabilization feature of the camera's gimbal.

2. Using the tag 10 approach includes the ArUco estimation error, which might positively compensate for the drone's motion error.

3. The drone gets unstable as it gets closer to objects due to the obstacle avoidance feature.

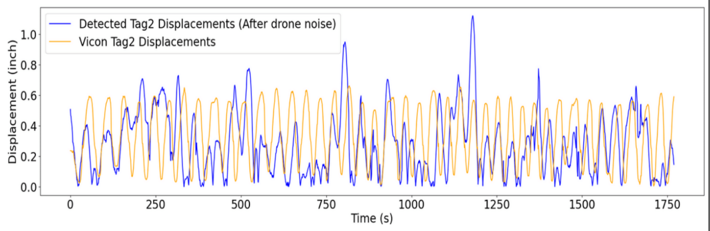
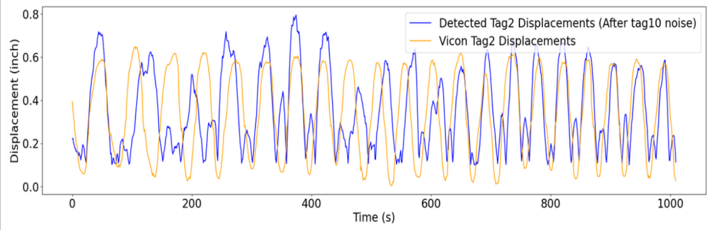
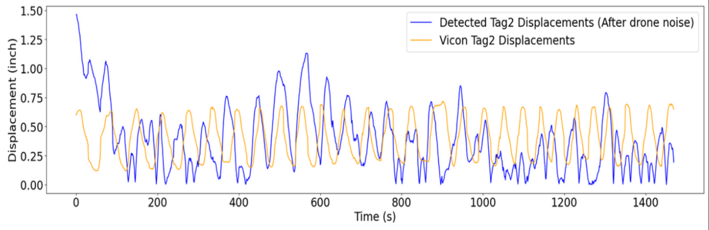
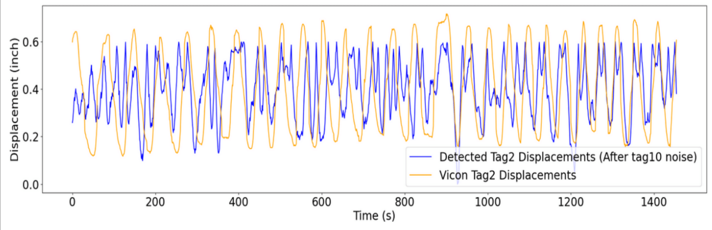
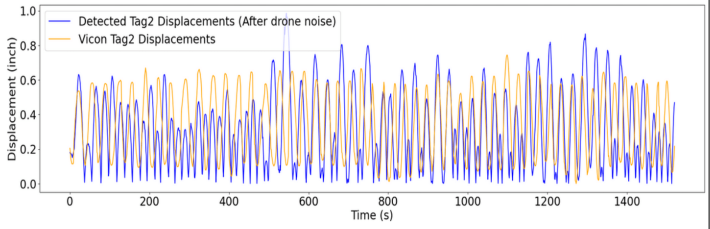
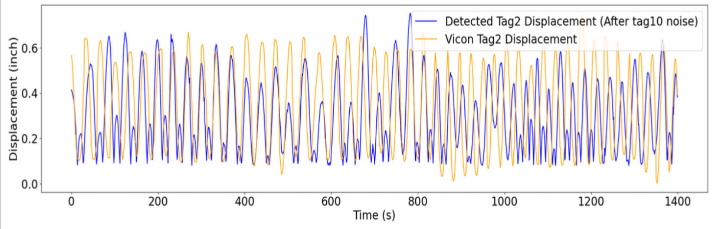
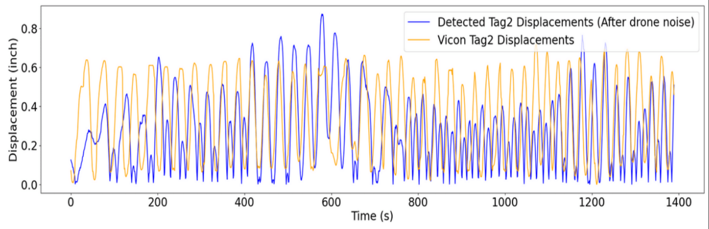
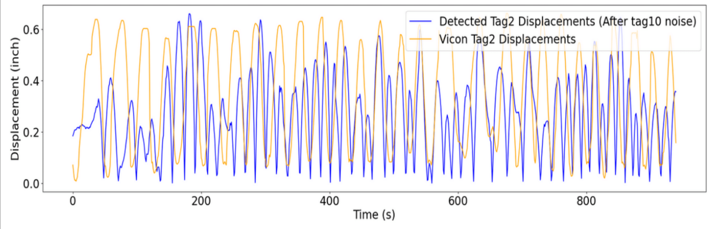
Test #1 (0.5in Sinusoidal)	Tag2 displacement (after drone noise) vs. Ground Truth	Tag2 displacement (after tag10 noise) vs. Ground Truth
a1 Drone was 1.8-2.5ft from target 2x zoom 0.5in = (± 0.5) 157px	**RMSE: 0.2748** 	**RMSE: 0.2074** 
a2 Drone was 1.8-2.5ft from target 1x zoom 0.5in = (± 0.5) 112px	**RMSE: 0.325** 	**RMSE: 0.1903** 
b1 Drone was 5-6ft from target 2x zoom 0.5in = (± 0.5) 57px	**RMSE: 0.2819** 	**RMSE:0.2409** 
b2 Drone was 5-6ft from target 1x zoom 0.5in = (± 0.5) 29px	**RMSE: 0.2667** 	**RMSE:0.257** 

Table 5.3: Results of estimating displacements for the 0.5-inch Sinusoidal Test using tag 10 and drone's movements

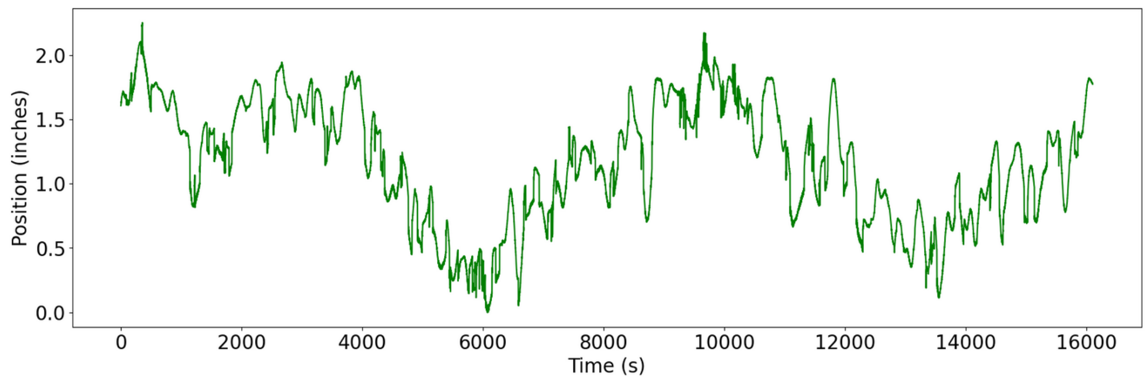


Figure 5.3: 1.0-inch Sinusoidal Test - Mavic Air 2 Inch Positions in Z direction

5.2.3 1.0-inch Sinusoidal Test

For the 1.0-inch sinusoidal test, the $a1$ test was not reported as tags were constantly getting out of the frame and hence were not detected by the ArUco pipeline. The lowest reported RMSE (**0.2581**) was for the $a2$ test using the tag 10 approach, see Table 5.4. This achieves an accuracy of **74.19**%. This accuracy was better than the 0.5-inch test as this $a2$ has a better pixel resolution of **190 pixels per inch**.

Similar to the previous test, the $a2$ test had the worst RMSE using the drone's motion approach as the drone for this test was the least stable with a motion range of 2.2 inches. Not only the range, but the drone movements within this range are also unstable, as seen in Figure 5.3. Again, the drone was most stable in the $b2$ test, which has the best RMSE using this approach with a range of ≈ 0.7 inches.

For this test, using the tag 10 approach to eliminate the drone's noise has shown better results across all sub-tests. However, for $b1$ and $b2$ tests, the difference gap between the RMSEs of the two approaches is less than the gap for the closer tests in $a1$ and $a2$. This is because the drone is further from the target in the $b1$ and $b2$ tests and hence more stable.

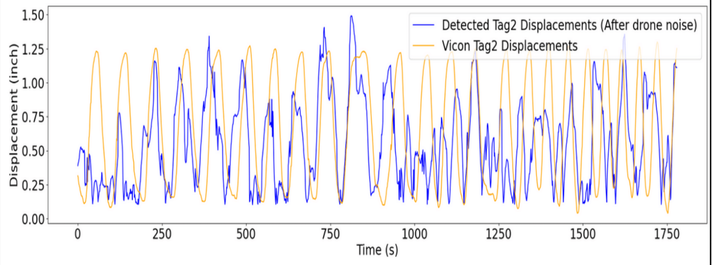
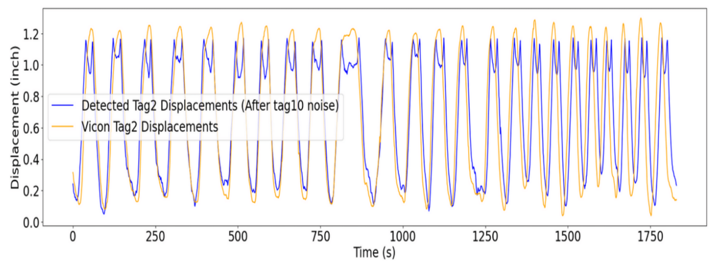
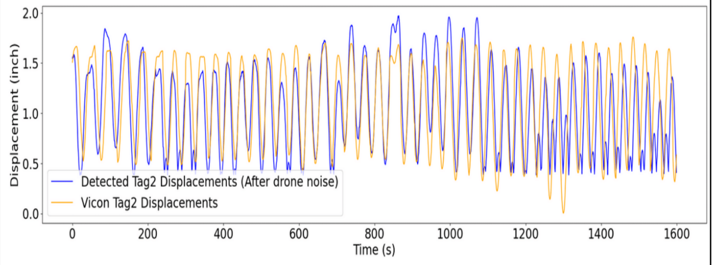
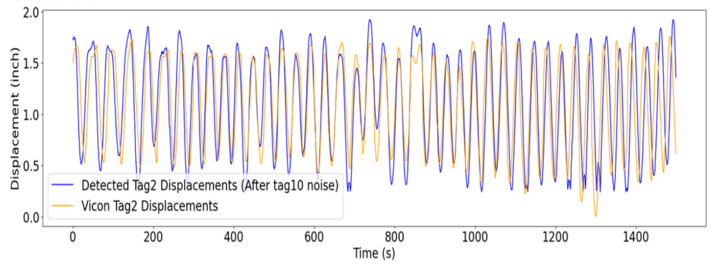
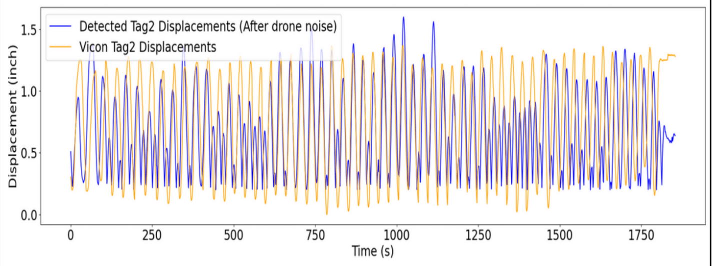
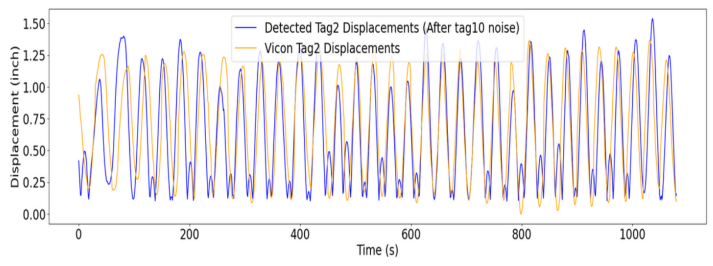
Test #1 (1.0in Sinusoidal)	Tag2 displacement (after drone noise) vs. Ground Truth	Tag2 displacement (after tag10 noise) vs. Ground Truth
a1 Drone was 1.8-2.5ft from target 2x zoom	**NA** (the drone was too close that the tags were out of frame)	**NA** (the drone was too close that the tags were out of frame)
a2 Drone was 1.8-2.5ft from target 1x zoom 1.0in = (± 0.5) 190px	**RMSE: 0.4826** 	**RMSE: 0.2581** 
b1 Drone was 5-6ft from target 2x zoom 1.0in = (± 0.5) 112px	**RMSE: 0.385** 	**RMSE: 0.367** 
b2 Drone was 5-6ft from target 1x zoom 1.0in = (± 0.5) 57px	**RMSE: 0.4408** 	**RMSE: 0.3217** 

Table 5.4: Results of estimating displacements for the 1.0-inch Sinusoidal Test using tag 10 and drone's movements

5.2.4 2.0-inch Sinusoidal Test

This test has shown the best accuracy so far at **90.1**% for the $a2$ test using the tag 10 approach for noise removal; see Table 5.5. The pixel resolution was **264 pixels per 2.0 inches**. This means that in order to achieve the same accuracy for other displacements, we need to achieve a pixel resolution of around 264 pixels. For instance, if we are capturing a displacement of 0.5 inches, we need a pixel resolution of 264 pixels per 0.5 inches in order to get a 90% accuracy. Surprisingly, the $a1$ test did not report a better accuracy even though it had high pixel resolution. However, this shows that more pixel accuracy could sometimes be harmful as it will magnify the ArUco estimation error itself. While less pixel resolution sounds worse, it could be better to cover the estimation error resulting from the ArUco pipeline.

Additionally, this test confirms that using the static tag approach to eliminate the drone's noise shows better results than using the drone's raw data. Using the drone's raw movement positions has reported higher RMSEs than when using the static tag approach.

Lastly, the drone for this test also gets stable as it gets away from the target. Thus, the gap RMSE difference gap between the two approaches decreases for the $b1$ and $b2$ tests.

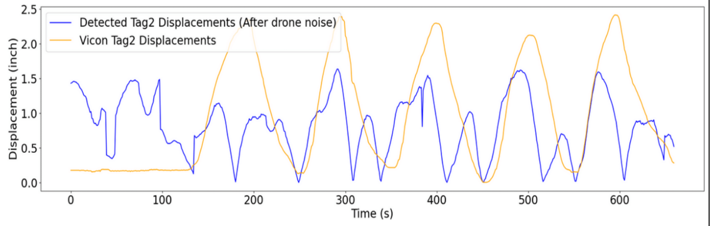
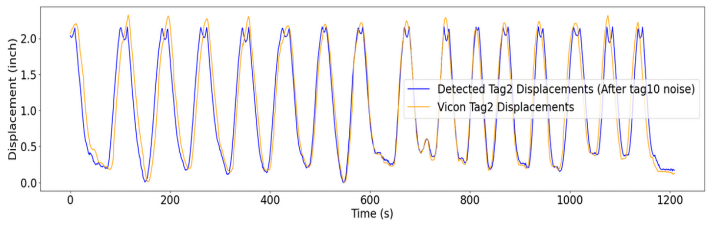
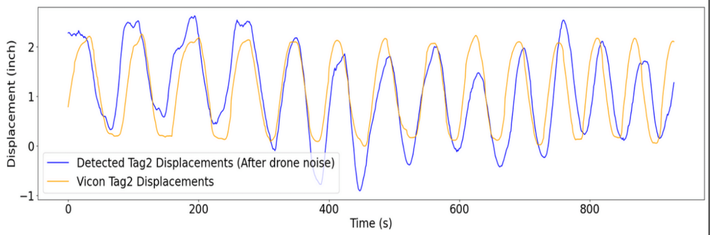
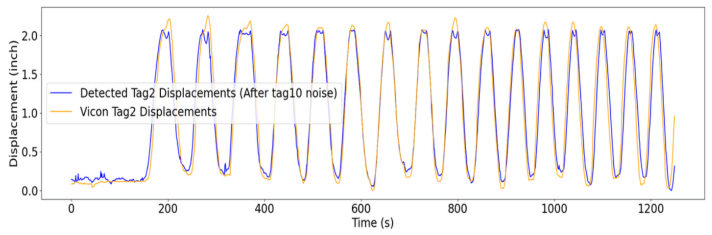
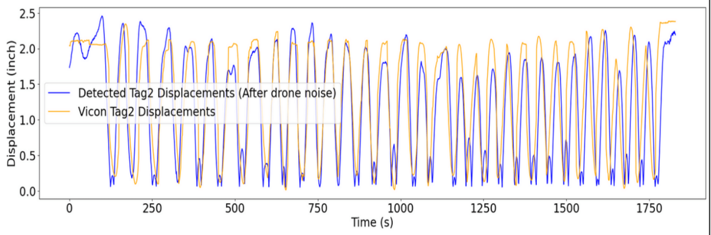
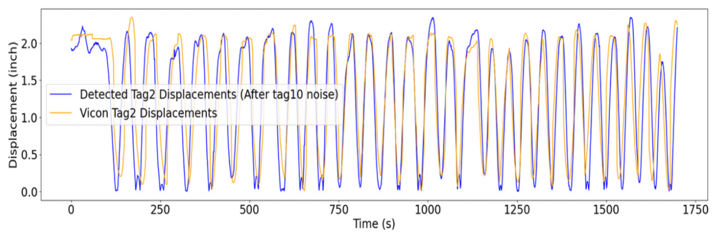
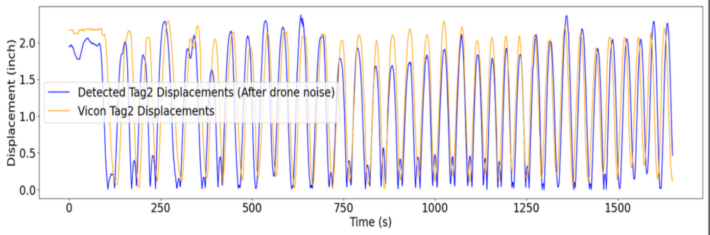
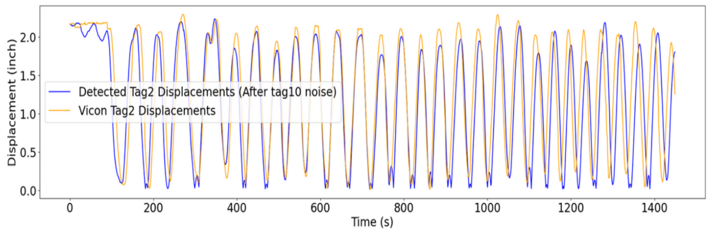
Test #1 (2.0in Sinusoidal)	Tag2 displacement (after drone noise) vs. Ground Truth	Tag2 displacement (after tag10 noise) vs. Ground Truth
a1 Drone was 1.8-2.5ft from target 2x zoom 2.0in = (± 0.5) 447px	**RMSE: 0.8063** 	**RMSE: 0.2489** 
a2 Drone was 1.8-2.5ft from target 1x zoom 2.0in = (± 0.5) 264px	**RMSE: 0.6332** 	**RMSE: 0.1971** 
b1 Drone was 5-6ft from target 2x zoom 2.0in = (± 0.5) 232px	**RMSE: 0.5761** 	**RMSE: 0.5133** 
b2 Drone was 5-6ft from target 1x zoom 2.0in = (± 0.5) 113px	**RMSE: 0.5862** 	**RMSE: 0.4924** 

Table 5.5: Results of estimating displacements for the 2.0-inch Sinusoidal Test using tag 10 and drone's movements

5.2.5 Stepping Testing

For this test, the tag moves in a stepping function instead of a sinusoidal. The tag goes from 0 inches to 4 inches in 0.5 inches increments. This test aims to show if the movement shapes of the tag affect the estimation accuracy. As shown in Table 5.6, the static tag approach still shows better results than using the drone's motion, even though the distance from the target has increased. This test shows that even though utilizing the drone's raw movement has shown better results than previous tests as the drone is further away from the target, the static tag approach still shows better results overall. The lowest RMSE reported is $\mathbf{0.0481}$ for $a1$. This offers an accuracy of $\mathbf{90.38}\%$. While this accuracy is close to that reported for the 2.0-inch test, the pixel resolution is much lower with 108 pixels instead of 264 pixel.

These results show that the type of movements could report different results for using a static tag. This is because a stepping function is more stable in motion than a sinusoidal one. Thus, it was easier for the ArUco pipeline to estimate even with lower pixel resolution. In addition, it would be better to use the static tag to eliminate the drone's noise.

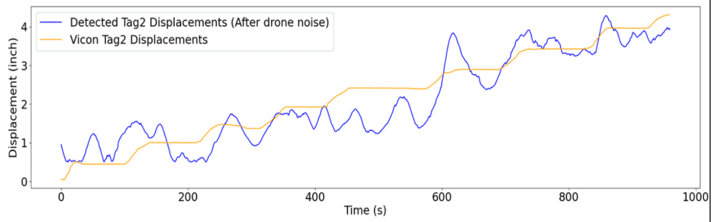
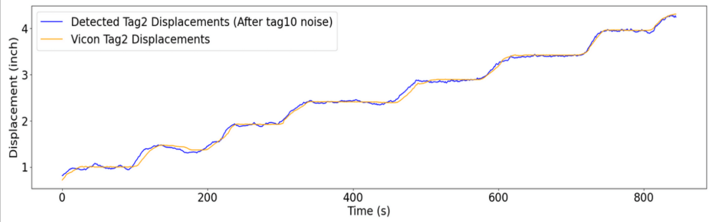
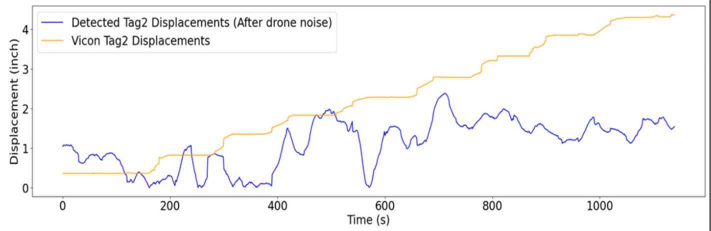
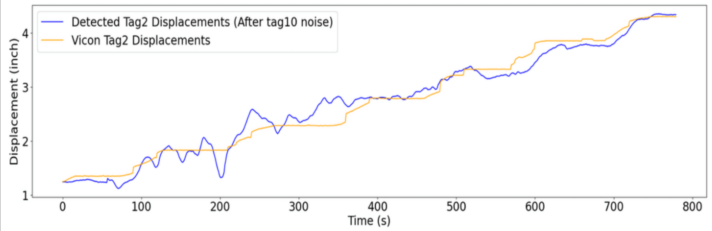
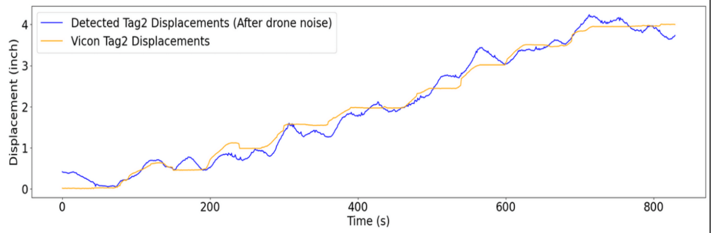
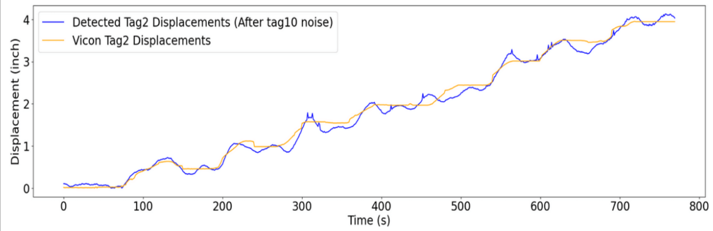
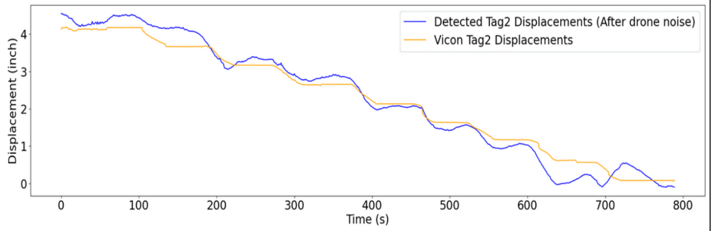
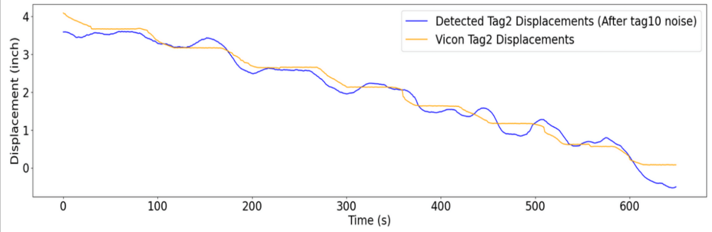
Test #1 (0.5in Stepping)	Tag2 displacement (after drone noise) vs. Ground Truth	Tag2 displacement (after tag10 noise) vs. Ground Truth
a1 Drone was 4ft from target 2x zoom 0.5in = (± 0.5) 108px	**RMSE: 0.4844** 	**RMSE: 0.0481** 
a2 Drone was 4ft from target 1x zoom 0.5in = (± 0.5) 55px	**RMSE: 1.5203** 	**RMSE: 0.173** 
b1 Drone was 5ft from target 2x zoom 0.5in = (± 0.5) 100px	**RMSE: 0.1845** 	**RMSE: 0.1166** 
b2 Drone was 5ft from target 1x zoom 2.0in = (± 0.5) 45px	**RMSE: 0.2712** 	**RMSE: 0.2066** 

Table 5.6: Results of estimating displacements for the Stepping Test using tag 10 and drone's movements

5.2.6 Conclusion

In conclusion, this experiment uses Mavic Air 2 drone to detect tag displacements in sinusoidal and stepping movements. The sinusoidal tests had different tag displacements of 0.5 inches, 1.0 inches, and 2.0 inches. The stepping function was from 0 inches to 4 inches in 0.5-inch increments. We test two approaches to eliminate the noise created by the drone's movements. The first approach uses the drone's raw positions obtained from Vicon and subtracts it from the estimated displacements to get the actual estimations. The second approach removes the reflection of the drone's noise on the static tag estimation from the moving tag. Each test compares and reports the RMSE for the two methods. The ground truth of the tag displacements is also obtained from Vicon.

While the stepping test had the smallest RMSE difference gap between the two approaches, using the static tag to eliminate the drone's noise has shown lower RMSEs across all tests. This is because the drone's motion is not a good representation of the camera's movements because of the gimbal stabilization. The stepping function had better results than previous tests in using the drone's raw positions because the drone was further away from the target. Thus, the drone's motion was close to the camera's motion. However, it was still not as good as using the static tag. In addition, the reflection of the drone's noise on the static tag accounts for the estimation error in the Aruco pipeline.

To answer research question 1, these tests also show the significance of pixel resolution in capturing tag displacements. To capture tag displacements in the sinusoidal form, the experiments show that there needs to be about 264 pixel per displacement to get an accuracy of 90%. For instance, to capture a 0.5-inch displacement, one would need 264 pixel per 0.5 inches to achieve an accuracy of about 90%. This pixel resolution can be achieved by adjusting the zoom, distance from the target, and camera resolution. In addition, these tests show the importance of drone stability in capturing displacements.

To answer research question 4, the drone's motion noise can be removed by utilizing a static reference tag as a reflection of the drone's motion and subtracting the estimated displacements of this tag from the moving tag displacements.

These experiments show that the type of movement of the tag could affect the estimation accuracy. For instance, the pixel resolution required to get a 90% accuracy for the stepping test was much less (108 pixels) than the sinusoidal. This is because the stepping test has a more stable movement to track in the ArUco pipeline than the sinusoidal. The stepping function has a waiting period after each increment until the next increment. This wait helps the ArUco detector to estimate the pixel position and hence get a correct position even with a lower pixel resolution.

Chapter 6

Discussion & Conclusions

This chapter summarizes the discussions presented in the previous chapters and the conclusions derived from the conducted experiments in section 6.1. Additionally, It discusses the assumptions included in building and utilizing the pipeline, along with its limitations, in section 6.2. We also present the final conclusions of this work in section 6.3. Lastly, section 6.4 offers the future direction of this future and its impact.

6.1 Discussion

This section reiterates the proposed research questions and how chapters 3, 4, and 5 attempted to answer them. Each of the proposed research questions is presented in a subsection along with the related findings and discussion.

6.1.1 Can we develop a proof-of-concept pipeline that utilizes state-of-the-art computer vision techniques to track bridge displacements captured from camera-equipped drones?

Findings: We were able to develop a proof-of-concept pipeline that is able to analyze captured videos of moving tags and return the estimated tag displacements in inches. This pipeline was then utilized to analyze videos captured by a camera-equipped drone (Mavic Air 2).

Discussion: Chapter 3 was written to answer this research question by developing the ArUco pipeline to be used to estimate displacements using ArUco tags. Chapter 3 analyzes the different phases of the pipeline and the tools utilized for each phase. The proposed ArUco pipeline is divided into five main components: capturing videos of the area of interest, analyzing the captured videos, getting pixel displacement values, converting from pixels to inches, adjusting the results, and ground truth comparison. This pipeline utilized ArUco tags and detectors to estimate the displacements of a targeted area by placing an ArUco tag on it. The proposed pipeline was tested with stationary (Chapter 4) and dynamic (Chapter 5) camera platforms (drones).

6.1.2 While fiducial marker systems have proven successful in tracking and localization of several robotics applications, how accurate are they in tracking such minimal displacements?

Findings: The lowest reported RMSE was **0.0043** for the Close Test of the stationary experiments using the Canon Camcorder. This test gave the highest accuracy of **95.7**%. For the drone experiments, the lowest RMSE within the sinusoidal tests was **0.1971** for the 2.0-inch test. This RMSE gave an accuracy of **90.1**%.

Discussion: To answer this question, Chapter 4 conducts three experiments to capture videos of an ArUco tag placed on a concrete specimen. The three conducted experiments are: Very Close, Close, and Far. Each experiment had a video recording of 7 minutes. An actuator exerted a force on the specimen generating 0.1-inch displacements. Two stationary camera platforms (Canon Camcorder - Blackfly camera) were used to capture videos of the moving tag. These videos were compared to the ground truth through the pipeline, and the RMSE was reported for each experiment. The highest RMSE reported was **0.0708** for the Far test using the Canon Camcorder. This gave an accuracy of **29.2**%. On the other hand, the lowest RMSE was **0.0043** for the Close Test with an accuracy of **95.7**%.

Chapter 5 deploys the Mavic Air 2 drone to conduct three experiments of capturing different tag displacements in sinusoidal movements. The three sinusoidal experiments are performed for three different displacements: 0.5-inch, 1.0-inch, and 2.0-inch. An additional experiment was conducted to capture the tag displacements in stepping movements from 0 to 4 inches in 0.5-inch increments. Each experiment has four tests with different pixel resolutions. The highest RMSE reported was **0.325** for the 0.5-inch. This gave an accuracy of **35**%. On the other hand, the lowest RMSE was **0.1971** for the 2.0-inch Test with an accuracy of **90.1**%.

6.1.3 What pixel-to-inch ratio is required to accurately detect tags with small displacements? Does the camera platform's resolution, zoom level, distance from the target, and motion on the platform affect the robustness of the estimation?

Findings: To reach an accuracy of **95.7**% using a stationary camera, the pixel resolution needed is **32.8298 pixels per 0.1 inch**. Using Mavic Air 2, the pixel-to-inch ratio needed is **264 pixels per 2.0 inches** to get an accuracy of **90.1**%.

Discussion: To answer this question, each experiment in chapters 4 and 5 had a different pixel resolution per inch to show the impact of pixel resolution on the overall accuracy. The pixel-to-inch resolution resulted from the camera's distance from the target and the zoom level. These experiments showed that there are several factors to impact the resulting accuracy. These factors are the blurriness of the lens, pixel resolution per inch, and camera resolution.

In addition, Chapter 5 conducts a comparison between two drones with different capabilities to capture the displacements of a static tag. These two drones are the Mavic Pro and the Mavic Air 2. We compare the estimation errors resulting from the videos captured by both drones. This is done by entering the videos from both drones into the ArUco pipeline. The Mavic Air 2 outperforms the Mavic Pro in terms of zoom level, stability (a small range of motion while hovering), and camera resolution.

6.1.4 How can the drone's motion noise be removed to reach the true estimated displacement of the tag?

Findings: We find that the best way to remove the estimation noise created by the drone's motion is by utilizing a static reference tag in the scene. Then, subtract the displacements estimation of the static tag from the estimation of the moving tag.

Discussion: To answer this question, Chapter 5 proposes two approaches to eliminate the drone's noise. The first approach removes the drone's raw motion from the ArUco estimation. The second approach utilizes a stationary tag (tag 10) placed in the camera's field of view to reflect the drone's noise. We compare the two approaches for each experiment and report the RMSE values. All experiments show that the second approach has outperformed the first approach in displacement estimation. The lowest RMSE within the sinusoidal tests was **0.1971** for the 2.0-

inch test. This RMSE gave an accuracy of **90.1%** using the second approach of the stationary tag with a pixel resolution of **264 pixels per 2.0 inches**. Throughout this test, the tag 10 approach always reports better RMSE values than the drone's raw motion approach. The drone's raw motion did not report accurate results for several reasons:

1. The drone's motion is different from the camera's motion due to the stabilization feature of the camera's gimbal.
2. Using the tag 10 approach includes the ArUco estimation error, which might positively compensate for the drone's motion error.
3. The drone gets unstable as it gets closer to objects due to the obstacle avoidance feature.

6.2 Assumptions & Limitations

Although this work has shown progress over some related work in utilizing state-of-the-art technologies, several assumptions were made, and considerable limitations were discovered along the way. We go over some of these notes in this list:

1. Tags utilized in this work must be placed on the area of interest. This can be challenging in remote bridge sites. As the process is simply sticking the tag on the bridge, clever approaches could be followed. However, there is the step of placing a target in the scene.
2. The different light conditions affect the ArUco tag detection system. Occlusion due to light conditions could be a roadblock in several scenarios. However, this could be tackled with the trade of lower detection frequency.

3. We assume that no challenges will be faced using a stationary camera when conducting experiments outdoors. However, when the pipeline was briefly tried outdoors using the Canon Camcorder, several challenges were found. For instance, stepping around the stationary camera while placed in an uneven space will affect the camera's stability and hence the estimation accuracy. Additionally, rain and wind might affect the camera's stability directly on indirectly. As shown in Figure 6.1, the umbrella tied to the camera was moving due to the wind which affected the camera's stability. Lack of accurate ground truth can also be a challenge when reporting the RMSE of the proposed pipeline outdoors. These challenges all led to an unsuccessful experiment.

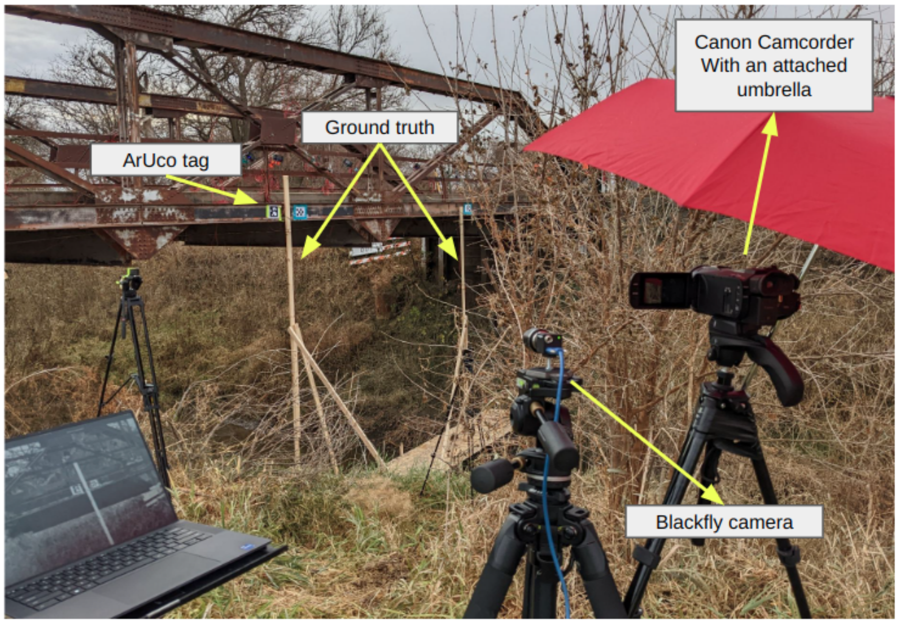


Figure 6.1: An outdoor setup for testing the proposed pipeline

4. As shown in Chapter 5, a stationary tag needs to be placed in the scene to eliminate the drone's noise. This extra step is not always easy to achieve if the bridge is remotely located. As shown in Figure 6.1, there is no static place

in the scene to hang a stationary tag on. Although a way around it would be to locate a stationary number of pixels in the video and use it as a reference, this adds an extra step to the pipeline.

5. We assume we can use the Mavic Air 2 for all applications. However, this drone might not always be used for security purposes.

6. As the drone experiments were conducted indoors, the drone stability was excellent. However, for outdoor testing with wind currents, the drone's stability will differ from indoor experiments.

6.3 Conclusions

A novel proof-of-concept pipeline was proposed to utilize bridge displacements using ArUco tags. The pipeline was successfully developed, tested, and showed promise using stationary and dynamic (drone) camera platforms. Stationary tests consisted of three experiments, each with two stationary camera platforms. Dynamic experiments had a total of 16 tests. These experiments exhibit the pipeline robustness and ability to achieve accuracy of **95.7**% and **90.1**% using a Canon camcorder and a Mavic Air 2, respectively. The pixel resolution for the stationary test was **32.8298 pixels per 0.1 inch**. The pixel resolution for the drone test was **264 pixels per 2.0 inches**. These experiments ensure the significance of high pixel resolution per the targeted displacement, stability of the platform, and camera resolution. However, it is important not to have a higher pixel resolution than required, as this might magnify the error estimation caused by the ArUco pipeline. It has been proven empirically that using a reference tag is more robust than using the drone's raw motion to eliminate drone noise. One of the main reasons is that the drone's motion is different from the camera's motion due to the stabilization feature of the camera's gimbal. Additionally, using the tag 10 approach includes the ArUco estimation error, which might positively compensate for the drone's motion error. Lastly, the obstacle avoidance feature makes the drone unstable as it gets closer to objects.

In conclusion, the key contributions of this work are:

1. Development of a novel pipeline leveraging the state-of-the-art ArUco fiducial marker system to measure bridge displacements in videos captured by stationary camera platforms as well as camera-equipped UAVs. This pipeline returns the bridge displacements in inches to compare with ground truth with accuracy of **95.7**% using a stationary camera and **90.1**% using a camera-equipped drone.

2. Laboratory experiments and results to demonstrate the accuracy level of estimating displacements by stationary cameras and drones. These studies include the effect of zoom level, camera resolution, and the distance to the target on the pipeline's accuracy. In addition, these studies deliver the pixel-to-inch ratio required to get the accuracy level needed to capture the targeted displacement.

3. Development of an approach compensating for UAV's motion to accurately estimate the targeted area's displacements using a static reference tag. This development is a result of a comparison study conducted to compare two different techniques to clear the noise created by the UAV's motions and vibrations.

6.4 Future Work

As this research tested ArUco tags in detecting small displacements as a state-of-the-art localization method, it indicates what to expect as the potential of using the targetless computer vision algorithms to detect small displacements. Moving forward, the use of computer vision in estimating bridge displacements using camera-equipped drones can be further improved. A targetless solution would be the goal of tracking bridge features without relying on reference points, such as laser points. This will allow for a more instant estimation without any preliminary steps to prepare the targeted area. In addition, such a solution will not face the same occlusion problem caused by the lighting conditions. However, high pixel resolution will be required for such a platform to report low RMSE values. Moreover, adding a fully automated pipeline to track outstanding features of the bridge and report the displacements online would be an ideal solution for estimating displacements.

Bibliography

[1] O. Araar, I. E. Mokhtari, and M. Bengherabi, "Pdcat: a framework for fast, robust, and occlusion resilient fiducial marker tracking," *Journal of Real-Time Image Processing*, vol. 18, no. 3, pp. 691–702, 2021. (document), 2.1

[2] P. Tsarouchi, S. Makris, and G. Chryssolouris, "Human–robot interaction review and challenges on task planning and programming," *International Journal of Computer Integrated Manufacturing*, vol. 29, no. 8, pp. 916–931, 2016. (document), 3.5

[3] Y. An, B. F. Spencer Jr, and J. Ou, "A test method for damage diagnosis of suspension bridge suspender cables," *Computer-Aided Civil and Infrastructure Engineering*, vol. 30, no. 10, pp. 771–784, 2015. 1.1

[4] H. Qarib and H. Adeli, "Recent advances in health monitoring of civil structures," *Scientia Iranica*, vol. 21, no. 6, pp. 1733–1742, 2014. 1.1

[5] F. Moreu, H. Jo, J. Li, R. Kim, C. S., A. Kimmle, S. Scola, H. Le, B. Spencer, and J. M. LaFave, "Dynamic assessment of timber railroad bridges using displacements." ASCE, 2014. 1.1, 2

[6] T. Nagayama and B. F. Spencer Jr, "Structural health monitoring using smart sensors," Newmark Structural Engineering Laboratory. University of Illinois at Urbana . . . , Tech. Rep., 2007. 1.1, 2

[7] G. W. Roberts, C. J. Brown, X. Meng, O. Ogundipe, C. Atkins, and B. Colford, “Deflection and frequency monitoring of the forth road bridge, scotland, by gps,” in *Proceedings of the Institution of Civil Engineers-Bridge Engineering*, vol. 165, no. 2. Thomas Telford Ltd, 2012, pp. 105–123. 1.1

[8] S. Jiang and J. Zhang, “Real-time crack assessment using deep neural networks with wall-climbing unmanned aerial system,” *Computer-Aided Civil and Infrastructure Engineering*, vol. 35, no. 6, pp. 549–564, 2020. 1.1

[9] S. Kim and J. Irizarry, “Exploratory study of user-perceived effectiveness of unmanned aircraft system (uas) integration in visual inspections of transportation agency,” *Innovative Infrastructure Solutions*, vol. 5, no. 3, pp. 1–17, 2020. 1.1

[10] Y. Han, G. Wu, and D. Feng, “Vision-based displacement measurement using an unmanned aerial vehicle,” *Structural Control and Health Monitoring*, vol. 29, no. 10, p. e3025, 2022. 1.1, 2.3

[11] J. Yang, J. Li, and G. Lin, “A simple approach to integration of acceleration data for dynamic soil–structure interaction analysis,” *Soil dynamics and earthquake engineering*, vol. 26, no. 8, pp. 725–734, 2006. 2

[12] A. I. Ozdagli, F. Moreu, J. A. Gomez, P. Garp, and S. Vemuganti, “Data fusion of accelerometers with inclinometers for reference-free high fidelity displacement estimation,” in *Proceedings of the 8th European Workshop on Structural Health Monitoring (EWSHM 2016), Bilbao, Spain*, 2016, pp. 5–8. 2

[13] S. Zhuge, X. Xu, L. Zhong, S. Gan, B. Lin, X. Yang, and X. Zhang, “Noncontact deflection measurement for bridge through a multi-uavs system,” *Computer-*

Aided Civil and Infrastructure Engineering, vol. 37, no. 6, pp. 746–761, 2022. 2, 2.3

[14] X. Bai and M. Yang, "Uav based accurate displacement monitoring through automatic filtering out its camera's translations and rotations," *Journal of Building Engineering*, vol. 44, p. 102992, 2021. 2

[15] G. Stephen, J. Brownjohn, and C. Taylor, "Measurements of static and dynamic displacement from visual monitoring of the humber bridge," *Engineering Structures*, vol. 15, no. 3, pp. 197–208, 1993. 2

[16] B. F. Spencer Jr, V. Hoskere, and Y. Narazaki, "Advances in computer vision-based civil infrastructure inspection and monitoring," *Engineering*, vol. 5, no. 2, pp. 199–222, 2019. 2

[17] Y. Xu and J. M. Brownjohn, "Review of machine-vision based methodologies for displacement measurement in civil structures," *Journal of Civil Structural Health Monitoring*, vol. 8, no. 1, pp. 91–110, 2018. 2

[18] D. Feng and M. Q. Feng, "Computer vision for shm of civil infrastructure: From dynamic response measurement to damage detection–a review," *Engineering Structures*, vol. 156, pp. 105–117, 2018. 2, 2.1

[19] S. Sony, S. Laventure, and A. Sadhu, "A literature review of next-generation smart sensing technology in structural health monitoring," *Structural Control and Health Monitoring*, vol. 26, no. 3, p. e2321, 2019. 2, 2.1

[20] C.-Z. Dong, O. Celik, F. N. Catbas, E. J. O'Brien, and S. Taylor, "Structural displacement monitoring using deep learning-based full field optical flow

methods," *Structure and Infrastructure Engineering*, vol. 16, no. 1, pp. 51–71, 2020. 2

[21] L. Duque, J. Seo, and J. Wacker, "Bridge deterioration quantification protocol using uav," *Journal of Bridge Engineering*, vol. 23, no. 10, p. 04018080, 2018. 2, 2.1

[22] J. Seo, L. Duque, and J. P. Wacker, "Field application of uas-based bridge inspection," *Transportation Research Record*, vol. 2672, no. 12, pp. 72–81, 2018. 2

[23] W. Li, X. S. Feng, K. Zha, S. Li, and H. S. Zhu, "Summary of target detection algorithms," in *Journal of Physics: Conference Series*, vol. 1757, no. 1. IOP Publishing, 2021, p. 012003. 2.1

[24] X. Ye, T.-H. Yi, C. Dong, and T. Liu, "Vision-based structural displacement measurement: System performance evaluation and influence factor analysis," *Measurement*, vol. 88, pp. 372–384, 2016. 2.1

[25] J. Lee, K.-C. Lee, S. Cho, and S.-H. Sim, "Computer vision-based structural displacement measurement robust to light-induced image degradation for in-service bridges," *Sensors*, vol. 17, no. 10, p. 2317, 2017. 2.1

[26] M. Kalaitzakis, B. Cain, S. Carroll, A. Ambrosi, C. Whitehead, and N. Vitzilaios, "Fiducial markers for pose estimation," *Journal of Intelligent & Robotic Systems*, vol. 101, no. 4, pp. 1–26, 2021. 2.1, 2.2

[27] L. Calvet, P. Gurdjos, C. Griwodz, and S. Gasparini, "Detection and accurate localization of circular fiducials under highly challenging conditions," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2016, pp. 562–570. 2.2

[28] L. B. Gatrell, W. A. Hoff, and C. W. Sklair, "Robust image features: Concentric contrasting circles and their image extraction," in *Cooperative Intelligent Robotics in Space II*, vol. 1612. SPIE, 1992, pp. 235–244. 2.2

[29] F. Bergamasco, A. Albarelli, E. Rodola, and A. Torsello, "Rune-tag: A high accuracy fiducial marker with strong occlusion resilience," in *CVPR 2011*. IEEE, 2011, pp. 113–120. 2.2

[30] M. Fiala, "Designing highly reliable fiducial markers," *IEEE Transactions on Pattern analysis and machine intelligence*, vol. 32, no. 7, pp. 1317–1324, 2009. 2.2, 2.2

[31] E. Olson, "Apriltag: A robust and flexible visual fiducial system," in *2011 IEEE international conference on robotics and automation*. IEEE, 2011, pp. 3400–3407. 2.2, 2.2

[32] H. Kato and M. Billinghurst, "Marker tracking and hmd calibration for a video-based augmented reality conferencing system," in *Proceedings 2nd IEEE and ACM International Workshop on Augmented Reality (IWAR'99)*. IEEE, 1999, pp. 85–94. 2.2

[33] C. N. Klokmose, J. B. Kristensen, R. Bagge, and K. Halskov, "Bullseye: High-precision fiducial tracking for table-based tangible interaction," in *Proceedings of the Ninth ACM international conference on interactive tabletops and surfaces*, 2014, pp. 269–278. 2.2

[34] M. Kaltenbrunner and R. Bencina, "reactivision: a computer-vision framework for table-based tangible interaction," in *Proceedings of the 1st international conference on Tangible and embedded interaction*, 2007, pp. 69–74. 2.2

[35] S. Garrido-Jurado, R. Muñoz-Salinas, F. J. Madrid-Cuevas, and M. J. Marín-Jiménez, "Automatic generation and detection of highly reliable fiducial markers under occlusion," *Pattern Recognition*, vol. 47, no. 6, pp. 2280–2292, 2014. 2.2, 2.2

[36] M. Fiala, "Artag, a fiducial marker system using digital techniques," in *2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05)*, vol. 2, 2005, pp. 590–596 vol. 2. 2.2

[37] J. Wang and E. Olson, "Apriltag 2: Efficient and robust fiducial detection," in *2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2016, pp. 4193–4198. 2.2

[38] M. Krogius, A. Haggenmiller, and E. Olson, "Flexible layouts for fiducial tags," in *2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2019, pp. 1898–1903. 2.2

[39] J. Wang, C. Sadler, C. F. Montoya, and J. C. Liu, "Optimizing ground vehicle tracking using unmanned aerial vehicle and embedded apriltag design," in *2016 International Conference on Computational Science and Computational Intelligence (CSCI)*, 2016, pp. 739–744. 2.2

[40] C. Nissler, S. B×üttner, Z.-C. Marton, L. Beckmann, and U. Thomasy, "Evaluation and improvement of global pose estimation with multiple apriltags for industrial manipulators," in *2016 IEEE 21st International Conference on Emerging Technologies and Factory Automation (ETFA)*, 2016, pp. 1–8. 2.2

[41] D. Tang, T. Hu, L. Shen, Z. Ma, and C. Pan, "AprilTag array-aided extrinsic calibration of camera-laser multi-sensor system," *Robotics Biomim.*, vol. 3, no. 1, p. 13, July 2016. 2.2

[42] F. J. Romero-Ramirez, R. Muñoz-Salinas, and R. Medina-Carnicer, "Speeded up detection of squared fiducial markers," *Image and vision Computing*, vol. 76, pp. 38–47, 2018. 2.2, 3.2.1

[43] C. B. Schmidt, "Uav positioning data determined via aruco tags for aircraft surface inspection," 2022. 2.2

[44] Y. Zhang, Y. Yu, S. Jia, and X. Wang, "Autonomous landing on ground target of uav by using image-based visual servo control," in *2017 36th Chinese Control Conference (CCC)*, 2017, pp. 11 204–11 209. 2.2

[45] S. Garrido-Jurado, R. Muñoz-Salinas, F. J. Madrid-Cuevas, and R. Medina-Carnicer, "Generation of fiducial marker dictionaries using mixed integer linear programming," *Pattern recognition*, vol. 51, pp. 481–491, 2016. 2.2

[46] B. Benligiray, C. Topal, and C. Akinlar, "Stag: A stable fiducial marker system," *Image and Vision Computing*, vol. 89, pp. 158–169, 2019. 2.2

[47] M. A. Vicente, D. C. Gonzalez, J. Minguez, and T. Schumacher, "A novel laser and video-based displacement transducer to monitor bridge deflections," *Sensors*, vol. 18, no. 4, p. 970, 2018. 2.2, 2.3

[48] Y. Yang, C. Dorn, T. Mancini, Z. Talken, G. Kenyon, C. Farrar, and D. Mascareñas, "Blind identification of full-field vibration modes from video measurements with phase-based video motion magnification," *Mechanical Systems and Signal Processing*, vol. 85, pp. 567–590, 2017. 2.3

[49] Y. Yang, C. Dorn, T. Mancini, Z. Talken, S. Nagarajaiah, G. Kenyon, C. Farrar, and D. Mascareñas, "Blind identification of full-field vibration modes of output-only structures from uniformly-sampled, possibly temporally-aliased

(sub-nyquist), video measurements," *Journal of Sound and Vibration*, vol. 390, pp. 232–256, 2017. 2.3

[50] Y. Yang, C. Dorn, C. Farrar, and D. Mascareñas, "Blind, simultaneous identification of full-field vibration modes and large rigid-body motion of output-only structures from digital video measurements," *Engineering Structures*, vol. 207, p. 110183, 2020. 2.3

[51] N. Wadhwa, M. Rubinstein, F. Durand, and W. T. Freeman, "Phase-based video motion processing," *ACM Transactions on Graphics (TOG)*, vol. 32, no. 4, pp. 1–10, 2013. 2.3

[52] J. G. Chen, N. Wadhwa, Y.-J. Cha, F. Durand, W. T. Freeman, and O. Buyukozturk, "Modal identification of simple structures with high-speed video using motion magnification," *Journal of Sound and Vibration*, vol. 345, pp. 58–71, 2015. 2.3

[53] H. Yoon, H. Elanwar, H. Choi, M. Golparvar-Fard, and B. F. Spencer Jr, "Target-free approach for vision-based structural system identification using consumer-grade cameras," *Structural Control and Health Monitoring*, vol. 23, no. 12, pp. 1405–1416, 2016. 2.3

[54] P. Garg, R. Nasimi, A. I. Ozdagli, S. Zhang, D. D. L. Mascarenas, M. Reda Taha, and F. Moreu, "Measuring transverse displacements using unmanned aerial systems laser doppler vibrometer (uas-ldv): Development and field validation," *Sensors*, vol. 20, no. 21, p. 6051, 2020. 2.3

[55] M. Quigley, K. Conley, B. Gerkey, J. Faust, T. Foote, J. Leibs, R. Wheeler, A. Y. Ng, *et al.*, "Ros: an open-source robot operating system," in *ICRA workshop on open source software*, vol. 3, no. 3.2. Kobe, Japan, 2009, p. 5. 3.1.1

[56] S. Tomar, "Converting video formats with ffmpeg," *Linux Journal*, vol. 2006, no. 146, p. 10, 2006. 3.2.1

Appendix A

Research Media:

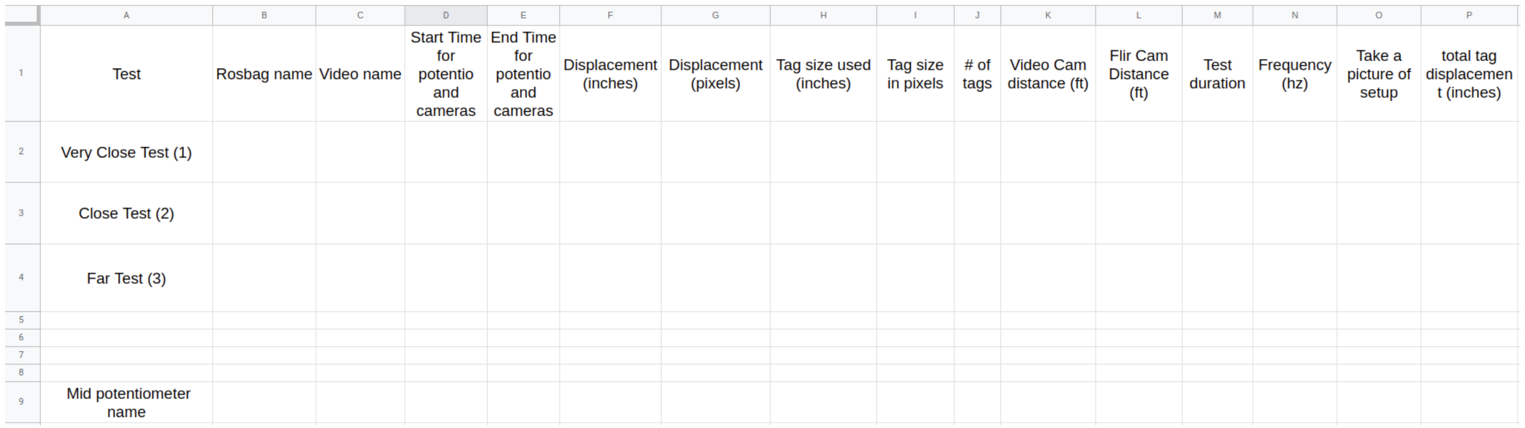
A.1 Pipeline Experiments

Stationary Testing - Close Test:

https://youtu.be/i818gew7g6o

Drone Testing - 2.0 Sinusoidal Experiment:

https://youtu.be/SFM3tAFJ7C0



	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P
1	Test	Rosbag name	Video name	Start Time for potentio and cameras	End Time for potentio and cameras	Displacement (inches)	Displacement (pixels)	Tag size used (inches)	Tag size in pixels	# of tags	Video Cam distance (ft)	Flir Cam Distance (ft)	Test duration	Frequency (hz)	Take a picture of setup	total tag displacemen t (inches)
2	Very Close Test (1)															
3	Close Test (2)															
4	Far Test (3)															
5																
6																
7																
8																
9	Mid potentiometer name															

Figure A.1: A screenshot of the dynamic testing sheet used during stationary testing

Appendix B

Code Scripts:

B.1 ArUco Detector Bash Script

```
#!/bin/bash

for f in ~/vid_frames/*.png
  do
    echo "Processing $f file..."
    # take action on each file. $f store current file name
    ./aruco-3.1.12/build/utils/aruco_simple "$f" >>
       DJI_0092_alone.csv
  done
```

B.2 Code File for Drone Testing - 2.0 Sinusoidal test

Drone Testing - 2.0 Sinusoidal test

December 26, 2022

0.1 Import

```
[1]: import sys
     print(sys.executable)
     import math
     import pandas as pd
     import numpy as np
     import bagpy
     from bagpy import bagreader
     import matplotlib.pyplot as plt
     import plotly.express as px
     import datetime
     import time
     from sklearn.metrics import mean_squared_error
```

```
/home/juba/.local/share/virtualenvs/juba-6MYV_K5R/bin/python
```

0.2 Vicon

```
[2]: vicon = bagreader('/home/juba/displacement tests/drone_tests/
       ↪mavic_air2_displacement_indoor_testing/vicon/test2.c2_2022-11-17-11-06-47.
       ↪bag')
```

```
[INFO]  Data folder /home/juba/displacement tests/drone_tests/mavic_air2_displac
ement_indoor_testing/vicon/test2.c2_2022-11-17-11-06-47 already exists. Not
creating.
```

```
[3]: vicon.topic_table
```

```
[3]:              Topics                              Types  Message Count   Frequency
     0  /vicon/air2/air2  geometry_msgs/TransformStamped          13848  202.242345
     1    vicon/tag2/tag2  geometry_msgs/TransformStamped          13848  201.697716
```

```
[4]: mavic_messages = vicon.message_by_topic(topic='/vicon/air2/air2')
     tag2_messages = vicon.message_by_topic(topic='vicon/tag2/tag2')
```

```
[5]: mavic = pd.read_csv(mavic_messages)
     tag2 = pd.read_csv(tag2_messages)
```

```
[6]: mavic['corrected_time'] = [datetime.datetime.fromtimestamp(x).strftime('%H:%M:
       ↪%S') for x in mavic['Time']]
     tag2['corrected_time'] = [datetime.datetime.fromtimestamp(x).strftime('%H:%M:
       ↪%S') for x in tag2['Time']]
```

```
[7]: mavic = mavic.round(decimals = 4)
     tag2 = tag2.round(decimals = 4)
```

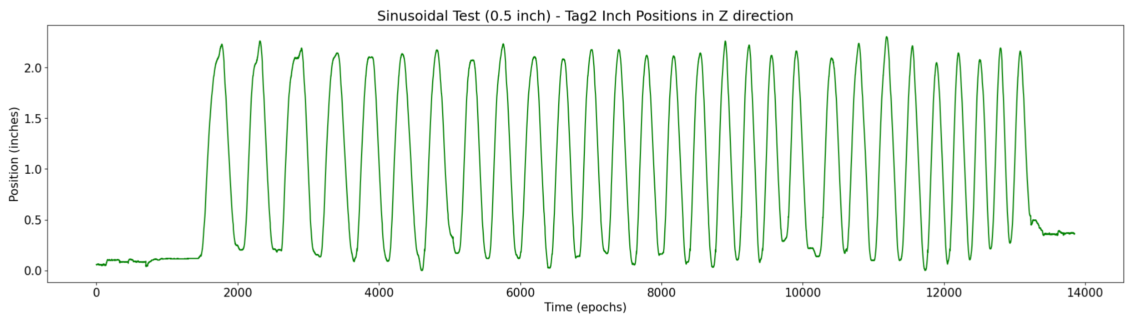
```
[8]: mavic['displacement_meter'] =  mavic['transform.translation.z'].max() -␣
       ↪mavic['transform.translation.z']
     mavic['displacement_inch'] =  (mavic['displacement_meter'] * 39.3701 * -1) + 0.
       ↪06319999999999992

     tag2['displacement_meter'] =  tag2['transform.translation.z'].max() -␣
       ↪tag2['transform.translation.z']
     tag2['displacement_inch'] =  tag2['displacement_meter'] * 39.3701
```

```
[9]: (mavic['displacement_meter'] * -1).min()
```

```
[9]: -0.06319999999999992
```

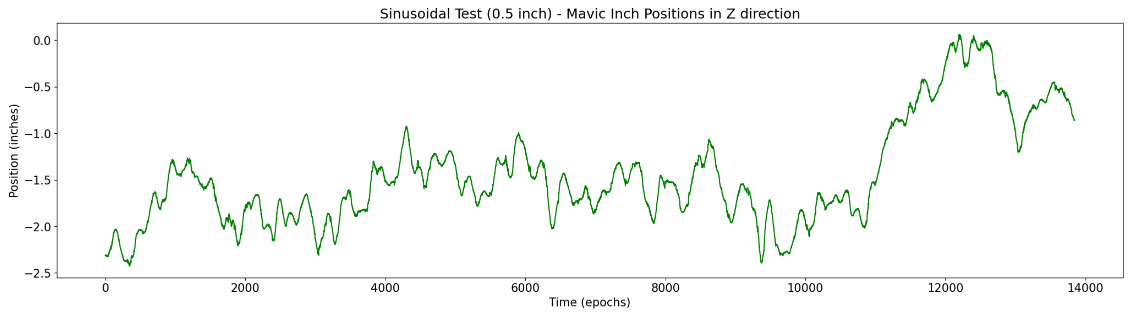
```
[10]: plt.rcParams.update({'font.size': 15})
      plt.rcParams['figure.figsize'] = [25, 6]
      plt.title('Sinusoidal Test (0.5 inch) - Tag2 Inch Positions in Z direction')
      plt.xlabel('Time (epochs)')
      plt.ylabel('Position (inches)')
      plt.plot(tag2.index.values, tag2['displacement_inch'], color='green')
```

```
[10]: [<matplotlib.lines.Line2D at 0x7f6a288b5a30>]
```



```
[11]: plt.rcParams.update({'font.size': 15})
      plt.rcParams['figure.figsize'] = [25, 6]
      plt.title('Sinusoidal Test (0.5 inch) - Mavic Inch Positions in Z direction')
      plt.xlabel('Time (epochs)')
      plt.ylabel('Position (inches)')
plt.plot(mavic.index.values, mavic['displacement_inch'], color='green')
```

```
[11]: [<matplotlib.lines.Line2D at 0x7f6a28898be0>]
```



0.3 Drone

```
[12]: drone = pd.read_csv("/home/juba/displacement tests/drone_tests/
      ↪mavic_air2_displacement_indoor_testing/csv_files/DJI_0098 - DJI_0098.csv")
```

```
[13]: drone.shape
```

```
[13]: (4230, 9)
```

```
[14]: drone
```

```
[14]:       tag  top_left_coord_x  top_left_coord_y  top_right_coord_x  \
      0      10           1574.80           560.808            1908.26
      1      10           1572.63           561.276            1905.87
      2      10           1571.58           562.610            1904.64
      3      10           1569.73           565.573            1902.54
      4      10           1568.37           569.400            1901.36
      …     …                 …                 …                  …
      4225   10           1619.01           713.323            1950.81
      4226    2           2248.02           798.816            2578.38
      4227   10           1618.34           713.494            1949.77
      4228    2           2247.83           799.054            2577.79
      4229   10           1617.93           713.425            1949.78

            top_right_coord_y  bottom_right_coord_x  bottom_right_coord_y  \
      0               552.043               1917.06               886.461
      1               552.216               1915.41               886.506
      2               554.332               1914.18               887.734
      3               556.671               1911.79               890.416
      4               561.288               1910.95               893.726
      …                   …                    …                     …
```

```
4225          704.258          1960.25          1036.520
4226          797.144          2577.68          1129.490
4227          705.125          1959.35          1036.900
4228          797.209          2577.40          1128.930
4229          705.212          1959.17          1036.890

      bottom_left_coord_x  bottom_left_coord_y
0                 1585.21              895.411
1                 1583.43              895.610
2                 1582.88              897.719
3                 1581.18              900.328
4                 1580.12              904.056
…                    …                    …
4225              1629.70             1046.690
4226              2247.87             1128.810
4227              1628.44             1046.990
4228              2247.48             1127.830
4229              1628.54             1046.830

[4230 rows x 9 columns]
```

```
[15]: drone = drone[119:4080] #from second 4 to 1:08
```

```
[16]: drone.tag.value_counts()
```

```
[16]: 10     1951
      2      1951
      89       32
      170      24
      162       2
      0         1
      Name: tag, dtype: int64
```

```
[17]: drone.loc[drone['tag']==2].index[0]
```

```
[17]: 120
```

```
[18]: drone = drone[['tag', 'top_left_coord_x', 'top_left_coord_y',␣
       ↪'bottom_left_coord_x', 'bottom_left_coord_y']]
```

```
[19]: drone.tag.value_counts()
```

```
[19]: 10     1951
      2      1951
      89       32
      170      24
      162       2
```

```
0         1
Name: tag, dtype: int64
```

```
[20]: drone_tag2 = drone[drone["tag"] == 2].reset_index().drop(columns=['index'])
drone_tag10 = drone[drone["tag"] == 10].reset_index().drop(columns=['index'])
```

```
[21]: drone_tag2['displacement_pixel_y'] =  drone_tag2['top_left_coord_y'].max() -␣
  ↪drone_tag2['top_left_coord_y']
px_list = []
displacement_of_target = 2
tag_size = 2.5
for index, row in drone_tag2.iterrows():
    px_euclidean_dist = math.dist([row['top_left_coord_x'],␣
  ↪row['top_left_coord_y']], [row['bottom_left_coord_x'],␣
  ↪row['bottom_left_coord_y']])
    px = (px_euclidean_dist * displacement_of_target) / tag_size
    px_list.append(px)
drone_tag2['px_list'] = px_list
print(px_list[0:10])
drone_tag2['displacement_inch_y'] = ((drone_tag2['displacement_pixel_y'] *␣
  ↪displacement_of_target) / drone_tag2['px_list'])
drone_tag2 = drone_tag2.drop(['px_list'], axis=1)
```

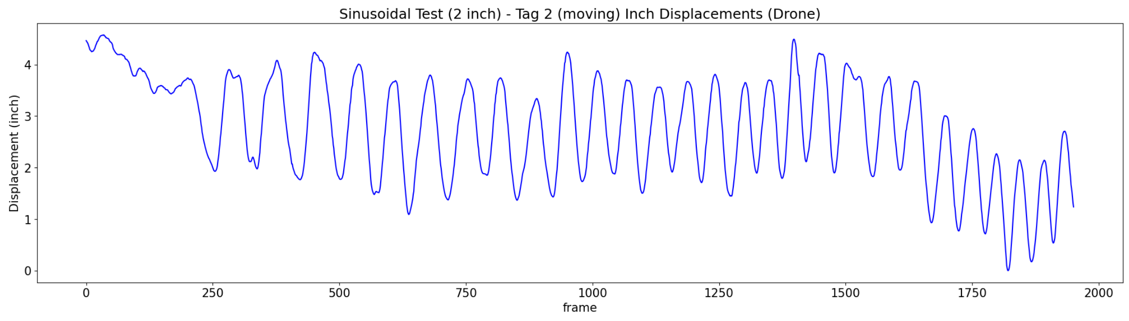
```
[264.5186672326927, 264.22000048444477, 264.4056533725404, 264.05128834648775,
263.99812800427196, 264.3969017863863, 264.0896347482044, 264.6930670566193,
264.2157436774728, 263.79280194091723]
```

```
[22]: drone_tag10['displacement_pixel_y'] =  drone_tag10['top_left_coord_y'].max() -␣
  ↪drone_tag10['top_left_coord_y']
px_list = []
displacement_of_target = 2
tag_size = 2.5
for index, row in drone_tag10.iterrows():
    px_euclidean_dist = math.dist([row['top_left_coord_x'],␣
  ↪row['top_left_coord_y']], [row['bottom_left_coord_x'],␣
  ↪row['bottom_left_coord_y']])
    px = (px_euclidean_dist * displacement_of_target) / tag_size
    px_list.append(px)
drone_tag10['px_list'] = px_list
print(px_list[0:10])
drone_tag10['displacement_inch_y'] = ((drone_tag10['displacement_pixel_y'] *␣
  ↪displacement_of_target) / drone_tag10['px_list'])
drone_tag10 = drone_tag10.drop(['px_list'], axis=1)
```

```
[267.9975808320665, 267.7540960938599, 267.9548156536844, 267.92466375277957,
267.25753202901507, 267.3965550365973, 267.60407140221173, 267.50362174265985,
267.27532634165834, 267.4368777095634]
```

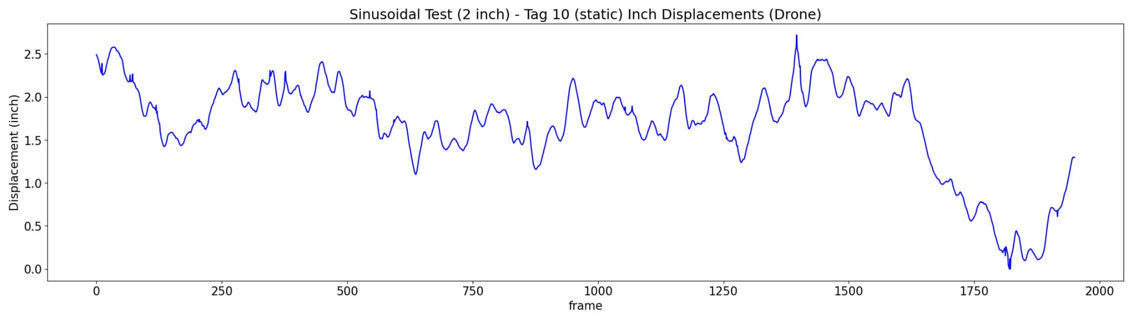
```
[23]: plt.rcParams.update({'font.size': 15})
      plt.rcParams['figure.figsize'] = [25, 6]
      plt.title('Sinusoidal Test (2 inch) - Tag 2 (moving) Inch Displacements␣
        ↪(Drone)')
      plt.xlabel('frame')
      plt.ylabel('Displacement (inch)')
      plt.plot(drone_tag2.index.values, drone_tag2['displacement_inch_y'],␣
        ↪color='blue')
```

```
[23]: [<matplotlib.lines.Line2D at 0x7f6a28435d00>]
```



```
[24]: plt.rcParams.update({'font.size': 15})
      plt.rcParams['figure.figsize'] = [25, 6]
      plt.title('Sinusoidal Test (2 inch) - Tag 10 (static) Inch Displacements␣
        ↪(Drone)')
      plt.xlabel('frame')
      plt.ylabel('Displacement (inch)')
      plt.plot(drone_tag10.index.values, drone_tag10['displacement_inch_y'],␣
        ↪color='blue')
```

```
[24]: [<matplotlib.lines.Line2D at 0x7f6a283b99a0>]
```



0.4 Tag2 detection minus Drone Vicon movements

[25]:
```
mavic
```

[25]:
```
                Time  header.seq  header.stamp.secs  header.stamp.nsecs  \
0       1.668705e+09      880454         1668705369           408511171
1       1.668705e+09      880455         1668705369           413375300
2       1.668705e+09      880456         1668705369           418451141
3       1.668705e+09      880457         1668705369           423347992
4       1.668705e+09      880458         1668705369           428365845
…                …         …               …                   …
13843   1.668705e+09      894297         1668705438           624199413
13844   1.668705e+09      894298         1668705438           629113220
13845   1.668705e+09      894299         1668705438           634266531
13846   1.668705e+09      894300         1668705438           639154165
13847   1.668705e+09      894301         1668705438           644133367

      header.frame_id   child_frame_id  transform.translation.x  \
0              /world  vicon/air2/air2                   1.3099
1              /world  vicon/air2/air2                   1.3099
2              /world  vicon/air2/air2                   1.3099
3              /world  vicon/air2/air2                   1.3100
4              /world  vicon/air2/air2                   1.3101
…                 …              …                       …
13843          /world  vicon/air2/air2                   1.3238
13844          /world  vicon/air2/air2                   1.3236
13845          /world  vicon/air2/air2                   1.3235
13846          /world  vicon/air2/air2                   1.3234
13847          /world  vicon/air2/air2                   1.3232

       transform.translation.y  transform.translation.z  transform.rotation.x  \
0                       0.9538                   1.4023               -0.0114
1                       0.9538                   1.4022               -0.0113
2                       0.9537                   1.4022               -0.0121
3                       0.9536                   1.4023               -0.0147
4                       0.9537                   1.4023               -0.0141
…                          …                        …                    …
13843                   0.9517                   1.4392               -0.0134
13844                   0.9518                   1.4391               -0.0139
13845                   0.9519                   1.4391               -0.0153
13846                   0.9520                   1.4391               -0.0162
13847                   0.9522                   1.4391               -0.0156

       transform.rotation.y  transform.rotation.z  transform.rotation.w  \
0                    0.0254                0.0128                0.9995
1                    0.0245                0.0119                0.9996
2                    0.0243                0.0116                0.9996
```

```
3                    0.0251              0.0137              0.9995
4                    0.0238              0.0113              0.9996
…                     …                  …                  …
13843                0.0330              0.0271              0.9990
13844                0.0329              0.0252              0.9990
13845                0.0337              0.0251              0.9990
13846                0.0337              0.0238              0.9990
13847                0.0335              0.0227              0.9991

      corrected_time  displacement_meter  displacement_inch
0           11:06:47              0.0603          -2.310817
1           11:06:47              0.0604          -2.314754
2           11:06:47              0.0604          -2.314754
3           11:06:47              0.0603          -2.310817
4           11:06:47              0.0603          -2.310817
…                …                 …                …
13843       11:07:57              0.0234          -0.858060
13844       11:07:57              0.0235          -0.861997
13845       11:07:57              0.0235          -0.861997
13846       11:07:57              0.0235          -0.861997
13847       11:07:57              0.0235          -0.861997

[13848 rows x 16 columns]
```

```
[26]: mavic.loc[mavic['corrected_time']=="11:06:47"].index
```

```
[26]: Int64Index([0, 1, 2, 3, 4, 5, 6, 7], dtype='int64')
```

```
[27]: mavic.loc[mavic['corrected_time']=="11:06:50"].index
```

```
[27]: Int64Index([412, 413, 414, 415, 416, 417, 418, 419, 420, 421,
            …
            602, 603, 604, 605, 606, 607, 608, 609, 610, 611],
           dtype='int64', length=200)
```

```
[28]: mavic.loc[mavic['corrected_time']=="11:07:54"].index
```

```
[28]: Int64Index([13212, 13213, 13214, 13215, 13216, 13217, 13218, 13219, 13220,
            13221,
            …
            13402, 13403, 13404, 13405, 13406, 13407, 13408, 13409, 13410,
            13411],
           dtype='int64', length=200)
```

```
[29]: #print(mavic.loc[[10000]])
```

```
[30]: #mavic[10000:]
```

```
[31]: mavic_vicon = pd.DataFrame()
      mavic_vicon = mavic[412:13411].reset_index().drop(columns=['index'])

      detected_tag2 = pd.DataFrame()
      detected_tag2 = drone_tag2
```

```
[32]: mavic_vicon
```

```
[32]:                Time  header.seq  header.stamp.secs  header.stamp.nsecs  \
      0      1.668705e+09      880866         1668705371           468516877
      1      1.668705e+09      880867         1668705371           473439946
      2      1.668705e+09      880868         1668705371           478482658
      3      1.668705e+09      880869         1668705371           483391483
      4      1.668705e+09      880870         1668705371           488490676
      …               …           …                 …                   …
      12994  1.668705e+09      893860         1668705436           439314416
      12995  1.668705e+09      893861         1668705436           444317318
      12996  1.668705e+09      893862         1668705436           449307145
      12997  1.668705e+09      893863         1668705436           454350234
      12998  1.668705e+09      893864         1668705436           459369901

            header.frame_id   child_frame_id  transform.translation.x  \
      0              /world  vicon/air2/air2                   1.3400
      1              /world  vicon/air2/air2                   1.3401
      2              /world  vicon/air2/air2                   1.3402
      3              /world  vicon/air2/air2                   1.3402
      4              /world  vicon/air2/air2                   1.3403
      …                  …               …                      …
      12994          /world  vicon/air2/air2                   1.3170
      12995          /world  vicon/air2/air2                   1.3171
      12996          /world  vicon/air2/air2                   1.3171
      12997          /world  vicon/air2/air2                   1.3170
      12998          /world  vicon/air2/air2                   1.3170

             transform.translation.y  transform.translation.z  transform.rotation.x  \
      0                       0.9510                   1.4032                0.0038
      1                       0.9509                   1.4033                0.0039
      2                       0.9510                   1.4033                0.0062
      3                       0.9508                   1.4036                0.0047
      4                       0.9507                   1.4037                0.0053
      …                           …                        …                     …
      12994                   0.9503                   1.4443               -0.0324
      12995                   0.9503                   1.4443               -0.0317
      12996                   0.9500                   1.4442               -0.0327
      12997                   0.9503                   1.4443               -0.0324
      12998                   0.9504                   1.4442               -0.0318
```

```
       transform.rotation.y  transform.rotation.z  transform.rotation.w  \
0                    0.0193                0.0403                0.9990
1                    0.0196                0.0405                0.9990
2                    0.0181                0.0395                0.9990
3                    0.0207                0.0415                0.9989
4                    0.0194                0.0405                0.9990
…                       …                      …                      …
12994                0.0188                0.0282                0.9989
12995                0.0191                0.0277                0.9989
12996                0.0190                0.0258                0.9990
12997                0.0198                0.0281                0.9989
12998                0.0188                0.0274                0.9989

      corrected_time  displacement_meter  displacement_inch
0           11:06:50              0.0594          -2.275384
1           11:06:50              0.0593          -2.271447
2           11:06:50              0.0593          -2.271447
3           11:06:50              0.0590          -2.259636
4           11:06:50              0.0589          -2.255699
…                …                    …                 …
12994       11:07:54              0.0183          -0.657273
12995       11:07:54              0.0183          -0.657273
12996       11:07:54              0.0184          -0.661210
12997       11:07:54              0.0183          -0.657273
12998       11:07:54              0.0184          -0.661210

[12999 rows x 16 columns]
```

```
[33]: detected_tag2['sec_num'] = detected_tag2.index // 30 + 1
      mavic_vicon['sec_num'] = mavic_vicon.index // 202.242345 + 1
```

```
[34]: detected_tag2 = detected_tag2.assign(time=detected_tag2.groupby('sec_num').
        ↪cumcount())
```

```
[35]: print(detected_tag2.groupby(['time']).size())
```

```
time
0      66
1      65
2      65
3      65
4      65
5      65
6      65
7      65
8      65
9      65
```

```
10    65
11    65
12    65
13    65
14    65
15    65
16    65
17    65
18    65
19    65
20    65
21    65
22    65
23    65
24    65
25    65
26    65
27    65
28    65
29    65
dtype: int64
```

[36]:
```
detected_tag2.drop(detected_tag2.loc[detected_tag2['sec_num']==66].index,␣
 ↪inplace=True)


detected_tag2 = detected_tag2.dropna()
detected_tag2 = detected_tag2.reset_index().drop(columns=['index'])
```

[37]:
```
print(detected_tag2.groupby(['time']).size())
```

```
time
0     65
1     65
2     65
3     65
4     65
5     65
6     65
7     65
8     65
9     65
10    65
11    65
12    65
13    65
14    65
15    65
```

```
16     65
17     65
18     65
19     65
20     65
21     65
22     65
23     65
24     65
25     65
26     65
27     65
28     65
29     65
dtype: int64
```

```
[38]: # detected_tag2.loc[detected_tag2['sec_num']==58].index
```

```
[39]: # detected_tag2.drop(detected_tag2.index[1710:1890], inplace=True)
      # detected_tag2 = detected_tag2.dropna()
      # detected_tag2 = detected_tag2.reset_index().drop(columns=['index'])
```

```
[40]: detected_tag2.shape
```

```
[40]: (1950, 9)
```

```
[41]: detected_tag2
```

```
[41]:       tag  top_left_coord_x  top_left_coord_y  bottom_left_coord_x  \
      0       2           2271.54           519.506              2271.07
      1       2           2270.60           522.100              2270.62
      2       2           2269.78           524.172              2269.99
      3       2           2268.65           526.775              2268.38
      4       2           2267.17           530.332              2266.51
      …     …               …                 …                    …
      1945    2           2250.81           892.741              2253.00
      1946    2           2255.00           901.000              2254.99
      1947    2           2259.00           913.000              2257.64
      1948    2           2262.00           925.000              2259.51
      1949    2           2261.17           939.817              2261.13

            bottom_left_coord_y  displacement_pixel_y  displacement_inch_y  sec_num  \
      0                 850.154               590.714             4.466331        1
      1                 852.375               588.120             4.451745        1
      2                 854.679               586.048             4.432946        1
      3                 856.839               583.445             4.419179        1
      4                 860.329               579.888             4.393122        1
```

```
…                …                 …             …     …
1945             1218.000          217.479        1.671545    65
1946             1227.780          209.220        1.600618    65
1947             1237.320          197.220        1.520244    65
1948             1251.150          185.220        1.419704    65
1949             1264.770          170.403        1.310982    65

      time
0        0
1        1
2        2
3        3
4        4
…     …
1945    25
1946    26
1947    27
1948    28
1949    29

[1950 rows x 9 columns]
```

[ ]:

```

```

[42]:

```
#mavic_vicon = mavic_vicon.assign(time=mavic_vicon.groupby('sec_num').
 ↪cumcount())
```

[43]:

```
#print(mavic_vicon.groupby(['time']).size())
```

[44]:

```
mavic_vicon.shape
```

[44]:

```
(12999, 17)
```

[45]:

```
t = pd.to_timedelta(mavic_vicon.Time, unit='T')
s = mavic_vicon.set_index(t).groupby('sec_num').resample('1.9S').last().
 ↪reset_index(drop=True)
s = s.assign(time=s.groupby('sec_num').cumcount())
mavic_vicon = s
```

[46]:

```
print(mavic_vicon.groupby(['time']).size())
```

```
time
0.0     65
1.0     65
2.0     65
3.0     65
4.0     65
```

```
5.0     65
6.0     65
7.0     65
8.0     65
9.0     64
10.0    64
11.0    64
12.0    64
13.0    64
14.0    64
15.0    64
16.0    64
17.0    64
18.0    64
19.0    64
20.0    64
21.0    64
22.0    64
23.0    64
24.0    64
25.0    64
26.0    64
27.0    64
28.0    64
29.0    64
30.0    63
31.0    60
32.0     5
dtype: int64
```

[47]:
```
after = mavic_vicon.groupby('sec_num').size()
after[after < 30]
```

[47]:
```
sec_num
65.0    9
dtype: int64
```

[48]:
```
mavic_vicon.drop(mavic_vicon.loc[mavic_vicon['time']==30.0].index, inplace=True)
mavic_vicon.drop(mavic_vicon.loc[mavic_vicon['time']==31.0].index, inplace=True)
mavic_vicon.drop(mavic_vicon.loc[mavic_vicon['time']==32.0].index, inplace=True)
mavic_vicon.drop(mavic_vicon.loc[mavic_vicon['sec_num']==65.0].index,␣
 ↪inplace=True)
# mavic_vicon.drop(mavic_vicon.loc[mavic_vicon['time']==37.0].index,␣
 ↪inplace=True)
# mavic_vicon.drop(mavic_vicon.loc[mavic_vicon['time']==38.0].index,␣
 ↪inplace=True)
```

```
# mavic_vicon.drop(mavic_vicon.loc[mavic_vicon['time']==39.0].index,␣
 ↪inplace=True)
# mavic_vicon.drop(mavic_vicon.loc[mavic_vicon['time']==40.0].index,␣
 ↪inplace=True)


mavic_vicon = mavic_vicon.dropna()
mavic_vicon = mavic_vicon.reset_index().drop(columns=['index'])
```

[49]:
```
print(mavic_vicon.groupby(['time']).size())
```

```
time
0.0     64
1.0     64
2.0     64
3.0     64
4.0     64
5.0     64
6.0     64
7.0     64
8.0     64
9.0     64
10.0    64
11.0    64
12.0    64
13.0    64
14.0    64
15.0    64
16.0    64
17.0    64
18.0    64
19.0    64
20.0    64
21.0    64
22.0    64
23.0    64
24.0    64
25.0    64
26.0    64
27.0    64
28.0    64
29.0    64
dtype: int64
```

[50]:
```
# mavic_vicon.drop(mavic_vicon.index[1953:2134], inplace=True)
# mavic_vicon = mavic_vicon.dropna()
# mavic_vicon = mavic_vicon.reset_index().drop(columns=['index'])
```

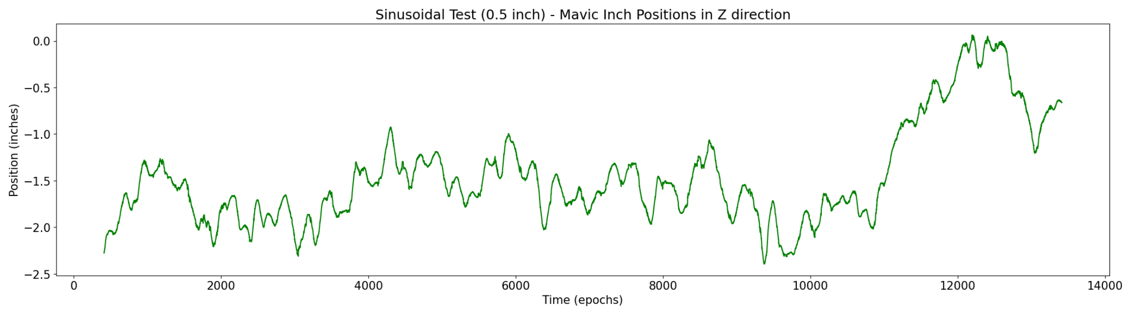
```
[51]: mavic_vicon
```

```
[51]:               Time  header.seq  header.stamp.secs  header.stamp.nsecs  \
      0     1.668705e+09    880872.0       1.668705e+09         498460392.0
      1     1.668705e+09    880878.0       1.668705e+09         528511001.0
      2     1.668705e+09    880884.0       1.668705e+09         558477856.0
      3     1.668705e+09    880891.0       1.668705e+09         593477174.0
      4     1.668705e+09    880897.0       1.668705e+09         623439306.0
      …              …           …                …                   …
      1915  1.668705e+09    893772.0       1.668705e+09         999365777.0
      1916  1.668705e+09    893778.0       1.668705e+09          29421214.0
      1917  1.668705e+09    893785.0       1.668705e+09          64311930.0
      1918  1.668705e+09    893791.0       1.668705e+09          94353273.0
      1919  1.668705e+09    893798.0       1.668705e+09         129403384.0

           header.frame_id   child_frame_id  transform.translation.x  \
      0             /world  vicon/air2/air2                   1.3404
      1             /world  vicon/air2/air2                   1.3402
      2             /world  vicon/air2/air2                   1.3401
      3             /world  vicon/air2/air2                   1.3399
      4             /world  vicon/air2/air2                   1.3398
      …                  …                …                      …
      1915          /world  vicon/air2/air2                   1.3194
      1916          /world  vicon/air2/air2                   1.3190
      1917          /world  vicon/air2/air2                   1.3185
      1918          /world  vicon/air2/air2                   1.3181
      1919          /world  vicon/air2/air2                   1.3175

            transform.translation.y  transform.translation.z  transform.rotation.x  \
      0                      0.9505                   1.4039                0.0043
      1                      0.9521                   1.4046                0.0117
      2                      0.9515                   1.4057                0.0041
      3                      0.9510                   1.4070                0.0014
      4                      0.9501                   1.4077               -0.0030
      …                         …                        …                     …
      1915                   0.9486                   1.4429               -0.0254
      1916                   0.9488                   1.4432               -0.0271
      1917                   0.9488                   1.4436               -0.0308
      1918                   0.9490                   1.4440               -0.0327
      1919                   0.9491                   1.4443               -0.0338

            transform.rotation.y  transform.rotation.z  transform.rotation.w  \
      0                   0.0197                0.0397                0.9990
      1                   0.0161                0.0169                0.9997
      2                   0.0204                0.0127                0.9997
      3                   0.0171                0.0058                0.9998
      4                   0.0184                0.0043                0.9998
```

```
…                    …                    …                    …
1915                 0.0181               0.0209               0.9993
1916                 0.0190               0.0222               0.9992
1917                 0.0213               0.0234               0.9990
1918                 0.0222               0.0231               0.9990
1919                 0.0237               0.0236               0.9989

     corrected_time  displacement_meter  displacement_inch  sec_num  time
0          11:06:50              0.0587          -2.247825      1.0   0.0
1          11:06:50              0.0580          -2.220266      1.0   1.0
2          11:06:50              0.0569          -2.176959      1.0   2.0
3          11:06:50              0.0556          -2.125778      1.0   3.0
4          11:06:50              0.0549          -2.098218      1.0   4.0
…                …                 …                 …       …     …
1915       11:07:54              0.0197          -0.712391     64.0  25.0
1916       11:07:54              0.0194          -0.700580     64.0  26.0
1917       11:07:54              0.0190          -0.684832     64.0  27.0
1918       11:07:54              0.0186          -0.669084     64.0  28.0
1919       11:07:54              0.0183          -0.657273     64.0  29.0

[1920 rows x 18 columns]
```

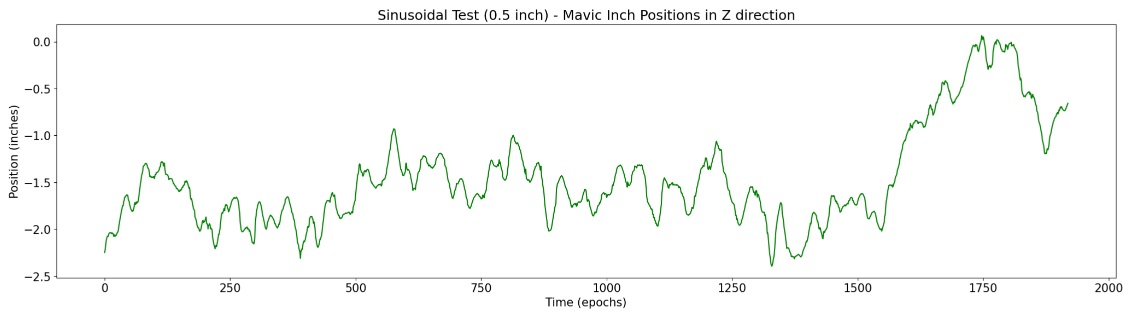
```
[52]: plt.rcParams.update({'font.size': 15})
      plt.rcParams['figure.figsize'] = [25, 6]
      plt.title('Sinusoidal Test (0.5 inch) - Mavic Inch Positions in Z direction')
      plt.xlabel('Time (epochs)')
      plt.ylabel('Position (inches)')
      plt.plot(mavic.index.values[412:13411], mavic['displacement_inch'][412:13411],␣
       ↪color='green')
```

```
[52]: [<matplotlib.lines.Line2D at 0x7f6a21edb640>]
```



```
[53]: plt.rcParams.update({'font.size': 15})
      plt.rcParams['figure.figsize'] = [25, 6]
      plt.title('Sinusoidal Test (0.5 inch) - Mavic Inch Positions in Z direction')
      plt.xlabel('Time (epochs)')
      plt.ylabel('Position (inches)')
      plt.plot(mavic_vicon.index.values, mavic_vicon['displacement_inch'],␣
      ↪color='green')
```

```
[53]: [<matplotlib.lines.Line2D at 0x7f6a21e933a0>]
```



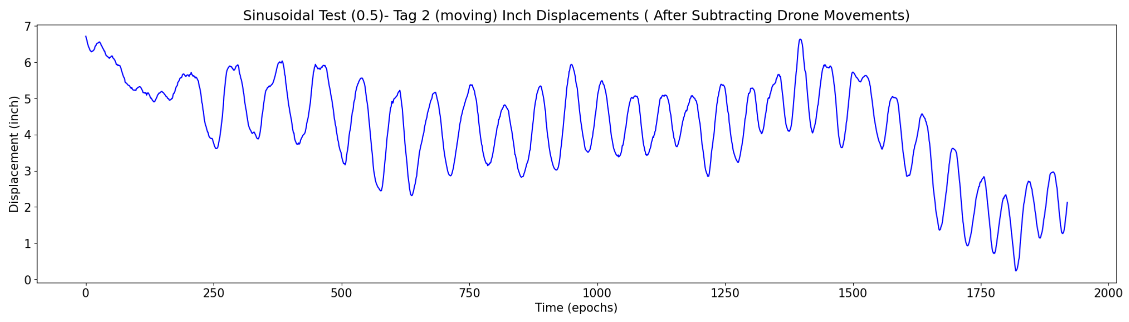
```
[54]: true_tag2_frm_drone = pd.DataFrame()
#true_tag2_frm_drone['Time'] = mavic_vicon['Time']
true_tag2_frm_drone['tag2_disp'] = detected_tag2['displacement_inch_y']
true_tag2_frm_drone['drone_disp'] = mavic_vicon['displacement_inch']

true_tag2_frm_drone['true_tag2_disp'] = ((true_tag2_frm_drone['tag2_disp'] -␣
↪true_tag2_frm_drone['drone_disp'] ).abs())
true_tag2_frm_drone = true_tag2_frm_drone.dropna()
true_tag2_frm_drone = true_tag2_frm_drone.reset_index().drop(columns=['index'])
```

```
[55]: (true_tag2_frm_drone['true_tag2_disp']).min()
```

```
[55]: 0.23816964170694668
```

```
[56]: plt.rcParams.update({'font.size': 15})
plt.rcParams['figure.figsize'] = [25, 6]
plt.title('Sinusoidal Test (0.5)- Tag 2 (moving) Inch Displacements ( After␣
↪Subtracting Drone Movements)')
plt.xlabel('Time (epochs)')
plt.ylabel('Displacement (inch)')
plt.plot(true_tag2_frm_drone.index.values␣
↪,true_tag2_frm_drone['true_tag2_disp'], color='blue')
```

```
[56]: [<matplotlib.lines.Line2D at 0x7f6a21e191f0>]
```



0.5 Aligning Tag2 (after subtracting drone movements) with vicon tag2

```
[57]: true_tag2_frm_drone_1 = pd.DataFrame()
      true_tag2_frm_drone_1 = true_tag2_frm_drone
```

```
[58]: true_tag2_frm_drone_1
```

```
[58]:       tag2_disp  drone_disp  true_tag2_disp
      0      4.466331   -2.247825        6.714156
      1      4.451745   -2.220266        6.672011
      2      4.432946   -2.176959        6.609905
      3      4.419179   -2.125778        6.544957
      4      4.393122   -2.098218        6.491341
      …           …          …              …
      1915   0.878611   -0.712391        1.591002
      1916   1.013957   -0.700580        1.714537
      1917   1.137326   -0.684832        1.822158
      1918   1.305313   -0.669084        1.974397
      1919   1.470916   -0.657273        2.128189

      [1920 rows x 3 columns]
```

```
[59]: vicon_tag2 = pd.DataFrame()
      vicon_tag2 = tag2[412:13411].reset_index().drop(columns=['index'])
      vicon_tag2['sec_num'] = vicon_tag2.index // 201.697716 + 1
```

```
[60]: t = pd.to_timedelta(vicon_tag2.Time, unit='T')
      s = vicon_tag2.set_index(t).groupby('sec_num').resample('1.9S').last().
       ↪reset_index(drop=True)
      s = s.assign(time=s.groupby('sec_num').cumcount())
      vicon_tag2 = s
```

```
[61]: print(vicon_tag2.groupby(['time']).size())
```

```
time
```

```
0.0     65
1.0     65
2.0     65
3.0     65
4.0     65
5.0     65
6.0     65
7.0     65
8.0     65
9.0     65
10.0    65
11.0    65
12.0    65
13.0    65
14.0    65
15.0    64
16.0    64
17.0    64
18.0    64
19.0    64
20.0    64
21.0    64
22.0    64
23.0    64
24.0    64
25.0    64
26.0    64
27.0    64
28.0    64
29.0    64
30.0    64
31.0    62
32.0     4
dtype: int64
```

[62]:
```
after = vicon_tag2.groupby('sec_num').size()
after[after < 30]
```

[62]:
```
sec_num
65.0    15
dtype: int64
```

[63]:
```
vicon_tag2.drop(vicon_tag2.loc[vicon_tag2['sec_num']==65.0].index, inplace=True)
vicon_tag2.drop(vicon_tag2.loc[vicon_tag2['time']==30.0].index, inplace=True)
vicon_tag2.drop(vicon_tag2.loc[vicon_tag2['time']==31.0].index, inplace=True)
vicon_tag2.drop(vicon_tag2.loc[vicon_tag2['time']==32.0].index, inplace=True)
```

```
vicon_tag2 = vicon_tag2.dropna()
vicon_tag2 = vicon_tag2.reset_index().drop(columns=['index'])
```

```
[64]: print(vicon_tag2.groupby(['time']).size())
```

```
time
0.0     64
1.0     64
2.0     64
3.0     64
4.0     64
5.0     64
6.0     64
7.0     64
8.0     64
9.0     64
10.0    64
11.0    64
12.0    64
13.0    64
14.0    64
15.0    64
16.0    64
17.0    64
18.0    64
19.0    64
20.0    64
21.0    64
22.0    64
23.0    64
24.0    64
25.0    64
26.0    64
27.0    64
28.0    64
29.0    64
dtype: int64
```

```
[65]: true_tag2_frm_drone_1 = true_tag2_frm_drone_1[20::]
      true_tag2_frm_drone_1 = true_tag2_frm_drone_1.reset_index().
        ↪drop(columns=['index'])
```

```
[66]: true_tag2_frm_drone_1.shape
```

```
[66]: (1900, 3)
```

[67]:
```
vicon_tag2.shape
```

[67]:
```
(1920, 18)
```

[68]:
```
vicon_tag2_1 = vicon_tag2[0::]
vicon_tag2_1 = vicon_tag2.reset_index().drop(columns=['index'])
```

[69]:
```
true_tag2_frm_drone_1 = true_tag2_frm_drone_1[170:1100]
true_tag2_frm_drone_1 = true_tag2_frm_drone_1.reset_index().
  ↪drop(columns=['index'])

vicon_tag2_1 = vicon_tag2_1[170:1100]
vicon_tag2_1 = vicon_tag2_1.reset_index().drop(columns=['index'])
```

[70]:
```
vicontag2_tag2afterdone = pd.DataFrame()
#vicontag2_tag2afterdone['Time'] = vicon_tag2['Time']
vicontag2_tag2afterdone['tag2_minus_drone'] =␣
  ↪(true_tag2_frm_drone_1['true_tag2_disp'] * 0.95) - 3.1
vicontag2_tag2afterdone['vicon'] = vicon_tag2_1['displacement_inch']
```

[71]:
```
vicontag2_tag2afterdone = vicontag2_tag2afterdone.dropna()
vicontag2_tag2afterdone = vicontag2_tag2afterdone.reset_index().
  ↪drop(columns=['index'])
```

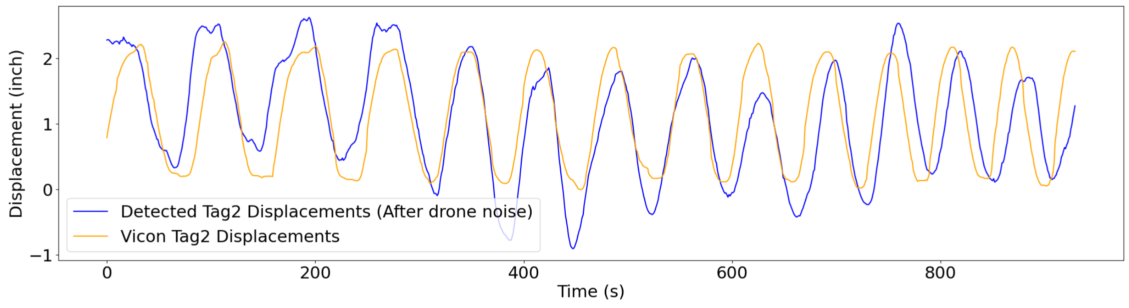
[72]:
```
vicontag2_tag2afterdone['rmse'] = np.
  ↪sqrt(mean_squared_error(vicontag2_tag2afterdone['vicon'],␣
  ↪vicontag2_tag2afterdone['tag2_minus_drone']))
vicontag2_tag2afterdone['diff'] = vicontag2_tag2afterdone['vicon'].
  ↪sub(vicontag2_tag2afterdone['tag2_minus_drone'], axis = 0)
vicontag2_tag2afterdone = vicontag2_tag2afterdone.round(decimals = 4)
```

[73]:
```
vicontag2_tag2afterdone.describe().round(decimals = 4)
```

[73]:
```
       tag2_minus_drone     vicon      rmse      diff
count          930.0000  930.0000  930.0000  930.0000
mean             1.0874    1.1286    0.6332    0.0411
std              0.8801    0.7679    0.0000    0.6322
min             -0.9021    0.0000    0.6332   -1.7423
25%              0.4048    0.2894    0.6332   -0.3797
50%              1.0854    1.1418    0.6332    0.0352
75%              1.8184    1.9242    0.6332    0.4857
max              2.6284    2.2559    0.6332    1.4730
```

[74]:
```
#(vicontag2_tag2afterdone['tag2_minus_drone']* -1).min() + 1.9277
```

[75]:
```
plt.rcParams.update({'font.size': 22})
plt.rcParams['figure.figsize'] = [25, 6]
# plt.title('Sinusoidal Test (0.5) - Vicon Tag2 vs Drone Tag2 detection (after␣
↪clearing drone noise)')
plt.xlabel('Time (s)')
plt.ylabel('Displacement (inch)')
plt.plot(vicontag2_tag2afterdone.index.values,␣
↪vicontag2_tag2afterdone['tag2_minus_drone'], color='blue', label = "Detected␣
↪Tag2 Displacements (After drone noise)")
plt.plot(vicontag2_tag2afterdone.index.values,␣
↪vicontag2_tag2afterdone['vicon'], color='orange', label = "Vicon Tag2␣
↪Displacements")
#plt.plot(vicontag2_tag2afterdone['Time'], vicontag2_tag2afterdone['rmse'],␣
↪color='red', label = "RMSE (0.4189)")
#plt.plot(vicontag2_tag2afterdone['Time'], vicontag2_tag2afterdone['diff'],␣
↪color='green', label = "Difference")
plt.legend()
plt.show()
```



[ ]:
```
```

0.6 Tag2 detection minus Tag10 movements

[76]:
```
detected_tag2_1 = pd.DataFrame()
detected_tag2_1 = drone_tag2
detected_tag10 = pd.DataFrame()
detected_tag10 = drone_tag10
```

[77]:
```
detected_tag10['sec_num'] = detected_tag10.index // 30 + 1
detected_tag10 = detected_tag10.assign(time=detected_tag10.groupby('sec_num').
↪cumcount())
```

[78]:
```
print(detected_tag10.groupby(['time']).size())
```

```
time
```

```
0     66
1     65
2     65
3     65
4     65
5     65
6     65
7     65
8     65
9     65
10    65
11    65
12    65
13    65
14    65
15    65
16    65
17    65
18    65
19    65
20    65
21    65
22    65
23    65
24    65
25    65
26    65
27    65
28    65
29    65
dtype: int64
```

```
[79]: detected_tag10.drop(detected_tag10.loc[detected_tag10['sec_num']==66].index,␣
        ↪inplace=True)
```

```
[80]: detected_tag10.shape
```

```
[80]: (1950, 9)
```

```
[81]: detected_tag2_1['sec_num'] = detected_tag2_1.index // 30 + 1
      detected_tag2_1 = detected_tag2_1.assign(time=detected_tag2_1.
        ↪groupby('sec_num').cumcount())
```

```
[82]: print(detected_tag2_1.groupby(['time']).size())
```

```
time
0     66
1     65
```

```
2     65
3     65
4     65
5     65
6     65
7     65
8     65
9     65
10    65
11    65
12    65
13    65
14    65
15    65
16    65
17    65
18    65
19    65
20    65
21    65
22    65
23    65
24    65
25    65
26    65
27    65
28    65
29    65
dtype: int64
```

[83]:
```
detected_tag2_1.drop(detected_tag2_1.loc[detected_tag2_1['sec_num']==66].index,␣
 ↪inplace=True)
```

[84]:
```
detected_tag2_1.shape
```

[84]: (1950, 9)

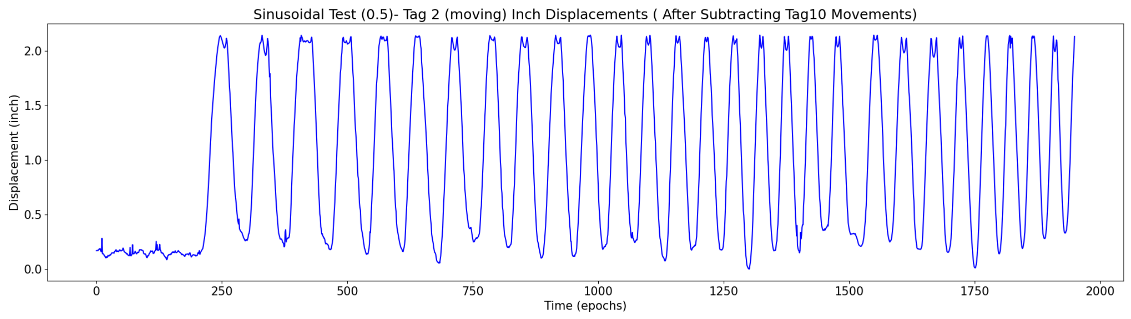
[85]:
```
true_tag2_frm_tag10 = pd.DataFrame()
true_tag2_frm_tag10['tag2'] = detected_tag2_1['displacement_inch_y']
true_tag2_frm_tag10['tag10'] = detected_tag10['displacement_inch_y']


true_tag2_frm_tag10['true_tag2_disp'] = ((true_tag2_frm_tag10['tag2'] -␣
 ↪true_tag2_frm_tag10['tag10'] ).abs()* -1) + 2.1422534784496516

true_tag2_frm_tag10 = true_tag2_frm_tag10.dropna()
true_tag2_frm_tag10 = true_tag2_frm_tag10.reset_index().drop(columns=['index'])
```

[86]:
```
(true_tag2_frm_tag10['true_tag2_disp']*-1).max()
```

[86]:
```
-0.0
```

[87]:
```
plt.rcParams.update({'font.size': 15})
plt.rcParams['figure.figsize'] = [25, 6]
plt.title('Sinusoidal Test (0.5)- Tag 2 (moving) Inch Displacements ( After␣
    ↪Subtracting Tag10 Movements)')
plt.xlabel('Time (epochs)')
plt.ylabel('Displacement (inch)')
plt.plot(true_tag2_frm_tag10.index.values␣
    ↪,true_tag2_frm_tag10['true_tag2_disp'], color='blue')
#plt.plot(true_tag2_frm_tag10.index.values ,true_tag2_frm_tag10['tag10'],␣
    ↪color='red')
#plt.plot(true_tag2_frm_tag10.index.values ,true_tag2_frm_tag10['tag2'],␣
    ↪color='green')
```

[87]:
```
[<matplotlib.lines.Line2D at 0x7f6a21d83d60>]
```



0.7 Aligning Tag2 (after subtracting tag10 movements) with vicon tag2

[88]:
```
vicon_tag2_1 = pd.DataFrame()
true_tag2_frm_tag10_1 = pd.DataFrame()
vicon_tag2_1 = vicon_tag2
true_tag2_frm_tag10_1 = true_tag2_frm_tag10
```

[89]:
```
vicon_tag2_1.shape
```

[89]:
```
(1920, 18)
```

[90]:
```
true_tag2_frm_tag10_1.shape
```

[90]:
```
(1950, 3)
```

[91]:
```
true_tag2_frm_tag10_1
```

[91]:
```
          tag2     tag10  true_tag2_disp
0     4.466331  2.491478        0.167400
1     4.451745  2.478311        0.168820
2     4.432946  2.460049        0.169357
3     4.419179  2.444478        0.167552
4     4.393122  2.425294        0.174425
…          …         …             …
1945  1.671545  1.273231        1.743939
1946  1.600618  1.289776        1.831411
1947  1.520244  1.299370        1.921379
1948  1.419704  1.300326        2.022875
1949  1.310982  1.298484        2.129756

[1950 rows x 3 columns]
```

[92]:
```
true_tag2_frm_tag10_1
```

[92]:
```
          tag2     tag10  true_tag2_disp
0     4.466331  2.491478        0.167400
1     4.451745  2.478311        0.168820
2     4.432946  2.460049        0.169357
3     4.419179  2.444478        0.167552
4     4.393122  2.425294        0.174425
…          …         …             …
1945  1.671545  1.273231        1.743939
1946  1.600618  1.289776        1.831411
1947  1.520244  1.299370        1.921379
1948  1.419704  1.300326        2.022875
1949  1.310982  1.298484        2.129756

[1950 rows x 3 columns]
```

[93]:
```
vicon_tag2_1
```

[93]:
```
              Time  header.seq  header.stamp.secs  header.stamp.nsecs  \
0     1.668705e+09   1016941.0       1.668705e+09         503463584.0
1     1.668705e+09   1016947.0       1.668705e+09         533425073.0
2     1.668705e+09   1016953.0       1.668705e+09         563456402.0
3     1.668705e+09   1016960.0       1.668705e+09         598455899.0
4     1.668705e+09   1016966.0       1.668705e+09         628527595.0
…              …          …                 …                   …
1915  1.668705e+09   1029806.0       1.668705e+09         829342841.0
1916  1.668705e+09   1029813.0       1.668705e+09         864349289.0
1917  1.668705e+09   1029819.0       1.668705e+09         894387350.0
1918  1.668705e+09   1029825.0       1.668705e+09         924375108.0
1919  1.668705e+09   1029831.0       1.668705e+09         954350954.0
```

```
     header.frame_id   child_frame_id  transform.translation.x  \
0             /world  vicon/tag2/tag2                   1.3883
1             /world  vicon/tag2/tag2                   1.3883
2             /world  vicon/tag2/tag2                   1.3883
3             /world  vicon/tag2/tag2                   1.3883
4             /world  vicon/tag2/tag2                   1.3884
…                  …                …                        …
1915          /world  vicon/tag2/tag2                   1.3882
1916          /world  vicon/tag2/tag2                   1.3882
1917          /world  vicon/tag2/tag2                   1.3880
1918          /world  vicon/tag2/tag2                   1.3880
1919          /world  vicon/tag2/tag2                   1.3880

      transform.translation.y  transform.translation.z  transform.rotation.x  \
0                      1.7267                   1.3459                0.0113
1                      1.7267                   1.3459                0.0107
2                      1.7267                   1.3458                0.0104
3                      1.7267                   1.3458                0.0121
4                      1.7266                   1.3458                0.0119
…                           …                        …                     …
1915                   1.7263                   1.3355                0.0010
1916                   1.7262                   1.3354                0.0019
1917                   1.7264                   1.3355               -0.0014
1918                   1.7262                   1.3355                0.0019
1919                   1.7262                   1.3358                0.0024

      transform.rotation.y  transform.rotation.z  transform.rotation.w  \
0                  -0.0206               -0.0773                0.9967
1                  -0.0213               -0.0772                0.9967
2                  -0.0204               -0.0778                0.9967
3                  -0.0210               -0.0767                0.9968
4                  -0.0206               -0.0772                0.9967
…                        …                     …                     …
1915               -0.0276               -0.0890                0.9957
1916               -0.0266               -0.0885                0.9957
1917               -0.0292               -0.0878                0.9957
1918               -0.0273               -0.0888                0.9957
1919               -0.0274               -0.0883                0.9957

     corrected_time  displacement_meter  displacement_inch  sec_num  time
0          11:06:50              0.0021           0.082677      1.0   0.0
1          11:06:50              0.0021           0.082677      1.0   1.0
2          11:06:50              0.0022           0.086614      1.0   2.0
3          11:06:50              0.0022           0.086614      1.0   3.0
4          11:06:50              0.0022           0.086614      1.0   4.0
…                 …                   …                  …        …     …
1915       11:07:54              0.0125           0.492126     64.0  25.0
```

```
1916        11:07:54              0.0126           0.496063     64.0   26.0
1917        11:07:54              0.0125           0.492126     64.0   27.0
1918        11:07:54              0.0125           0.492126     64.0   28.0
1919        11:07:54              0.0122           0.480315     64.0   29.0

[1920 rows x 18 columns]
```

```
[94]: true_tag2_frm_tag10_1 = true_tag2_frm_tag10_1[53:1950]
      true_tag2_frm_tag10_1 = true_tag2_frm_tag10_1.reset_index().
        ↪drop(columns=['index'])
```

```
[95]: true_tag2_frm_tag10_1.shape
```

```
[95]: (1897, 3)
```

```
[96]: vicon_tag2_1 = vicon_tag2_1[0:1897]
      vicon_tag2_1 = vicon_tag2_1.reset_index().drop(columns=['index'])
```

```
[97]: true_tag2_frm_tag10_1 = true_tag2_frm_tag10_1[5:1255]
      true_tag2_frm_tag10_1 = true_tag2_frm_tag10_1.reset_index().
        ↪drop(columns=['index'])
      vicon_tag2_1 = vicon_tag2_1[0:1250]
```

```
[98]: vicontag2_tag2aftertag10 = pd.DataFrame()
      #vicontag2_tag2aftertag10['Time'] = vicon_tag2_1['Time']
      vicontag2_tag2aftertag10['tag2_minus_tag10'] =␣
        ↪true_tag2_frm_tag10_1['true_tag2_disp'] * 0.97
      vicontag2_tag2aftertag10['vicon'] = vicon_tag2_1['displacement_inch']

      vicontag2_tag2aftertag10 = vicontag2_tag2aftertag10.dropna()
      vicontag2_tag2aftertag10 = vicontag2_tag2aftertag10.reset_index().
        ↪drop(columns=['index'])

      vicontag2_tag2aftertag10['rmse'] = np.
        ↪sqrt(mean_squared_error(vicontag2_tag2aftertag10['vicon'],␣
        ↪vicontag2_tag2aftertag10['tag2_minus_tag10']))
      vicontag2_tag2aftertag10['diff'] = vicontag2_tag2aftertag10['vicon'].
        ↪sub(vicontag2_tag2aftertag10['tag2_minus_tag10'], axis = 0)
      vicontag2_tag2aftertag10 = vicontag2_tag2aftertag10.round(decimals = 4)
```

```
[99]: vicontag2_tag2aftertag10
```

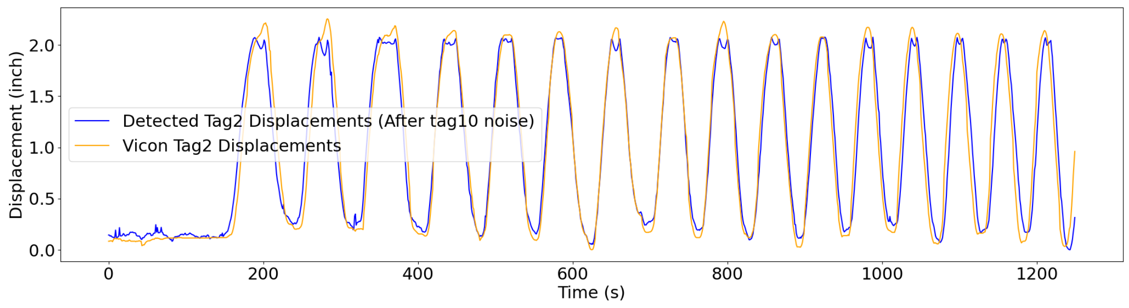
```
[99]:       tag2_minus_tag10   vicon    rmse    diff
      0               0.1435  0.0827  0.1971 -0.0609
      1               0.1420  0.0827  0.1971 -0.0593
      2               0.1330  0.0866  0.1971 -0.0464
      3               0.1368  0.0866  0.1971 -0.0502
```

```
4                0.1281  0.0866  0.1971 -0.0415
…                 …       …       …       …
1245             0.0593  0.4291  0.1971  0.3698
1246             0.1034  0.5591  0.1971  0.4557
1247             0.1694  0.7205  0.1971  0.5511
1248             0.2281  0.8425  0.1971  0.6145
1249             0.3148  0.9606  0.1971  0.6458

[1250 rows x 4 columns]
```

[100]:
```
vicontag2_tag2aftertag10.describe().round(decimals = 4)
```

[100]:
```
       tag2_minus_tag10      vicon       rmse       diff
count         1250.0000  1250.0000  1250.0000  1250.0000
mean             0.9784     0.9755     0.1971    -0.0029
std              0.7475     0.7902     0.0000     0.1971
min              0.0000     0.0000     0.1971    -0.5246
25%              0.2402     0.1732     0.1971    -0.0911
50%              0.8165     0.8189     0.1971    -0.0192
75%              1.7823     1.8140     0.1971     0.0887
max              2.0777     2.2559     0.1971     0.6458
```

[101]:
```
plt.rcParams.update({'font.size': 22})
plt.rcParams['figure.figsize'] = [25, 6]
# plt.title('Sinusoidal Test (0.5) - Vicon Tag2 vs Drone Tag2 detection (after␣
 ↪clearing tag10 noise)')
plt.xlabel('Time (s)')
plt.ylabel('Displacement (inch)')
plt.plot(vicontag2_tag2aftertag10.index.values,␣
 ↪vicontag2_tag2aftertag10['tag2_minus_tag10'], color='blue', label =␣
 ↪"Detected Tag2 Displacements (After tag10 noise)")
plt.plot(vicontag2_tag2aftertag10.index.values,␣
 ↪vicontag2_tag2aftertag10['vicon'], color='orange', label = "Vicon Tag2␣
 ↪Displacements")
#plt.plot(vicontag2_tag2aftertag10.index.values,␣
 ↪vicontag2_tag2aftertag10['rmse'], color='red', label = "RMSE (0.2168)")
#plt.plot(vicontag2_tag2aftertag10.index.values,␣
 ↪vicontag2_tag2aftertag10['diff'], color='green', label = "Difference")
plt.legend()
plt.show()
```


Detected Tag2 Displacements (After tag10 noise)
Vicon Tag2 Displacements
Displacement (inch)
Time (s)
0.0
0.5
1.0
1.5
2.0
0
200
400
600
800
1000
1200

```
[ ]:
```

```
[ ]:
```